

**The Soft Skills of Software Learning Development: The  
Psychological Dimensions of Computing and Security  
Behaviours**



A thesis submitted to Lancaster University in partial fulfilment of the requirements for  
the degree of doctor of Philosophy

**Matthew Ivory, BSc, MSc**

Department of Psychology

Lancaster University

September 2025

### **Declaration**

The thesis contains original work completed solely by the author under the supervision of Professor John Towse, Professor Mark Levine, Doctor Miriam Sturdee, and Professor Bashar Nuseibeh, and has not been submitted in the same form for the award of a higher degree at this institution or elsewhere.

This PhD research was funded through the EPSRC Doctoral Training Centre.

Any sections of the thesis which have been published are clearly identified in the Statement of Authorship.

Name: Matthew Ivory

Signature:

Date: 2025-09-08

## Acknowledgments

I extend my gratitude towards my supervisors, John Towse, Mark Levine, Miriam Sturdee, and Bashar Nuseibeh, without whom this thesis would not be what it is. Your insight and guidance have been appreciated. John, for steering me in the right direction. It did not go unnoticed, and I saw my work improve each time. I plan to continue on this path. Mark, for offering distilled insights. Miriam, thank you for offering time to talk and providing feedback. Bashar, for being the expert software engineer.

To my family, for instilling in me a passion for learning I did not necessarily realise I possessed. My mother, who I don't think understood my PhD but supported me regardless. My father, whose work ethic and academic achievements inspired me more than he knows. Together, they have consistently shown me love and support. I don't think they will ever truly understand their impact, but hopefully this goes some way.

To the Cumbrian fells, for providing space away from the desk when needed. To the friends who accompanied me on these ventures. Between crisp granite mornings and warm rhyolite evenings, you gave these times value.

Finally, and most importantly, to my fiancé Anna, my best friend. Your unwavering support and love have been critical. Your dedication to your work continually impresses me. Thank you for listening to me and offering encouragement. I think you understood some of my research, but I imagine you were glad I didn't ask you to read it.

# **The Soft Skills of Software Learning Development: The Psychological Dimensions of Computing and Security Behaviours**

**Matthew Ivory, BSc, MSc**

## **Abstract**

Security is critical to high-quality software, yet vulnerabilities are routinely identified. Are cognitive and social psychology constructs relevant to software engineers' security behaviours? Empirical data to adjudicate this question is currently sparse. I argue that developers' psychology is a keystone which bridges technical knowledge and successful code. Through a multi-phase, mixed-methods approach and five empirical work packages, I answer questions about which soft skills are valued in software engineering and how latent psychological theories influence security behaviours. Phase one investigates the value of psychologically rooted soft skills through explorations of graduate and educator perceptions. Success is supported by communication, teamwork, problem solving, and critical thinking. These are translated into latent psychological dimensions: Social Identity Theory and Dual Processing Theory of Decision Making. Phase two investigates risk perception around secure coding in line with dual processing, establishing a complex but important relationship between cognitive reflection and unrealistic optimism for risk awareness. Security perceptions are explored through social identities, which modulate how developers represent security, resulting in complex constructions of software and personal responsibility. Identifying with others enhances developers' responsibility, but an absence of shared identities can lead to responsibility being rejected. A final study

utilises dual processing theory to explain why security vulnerabilities are ‘invisible’ to developers, using a groundwork study and power analysis to emphasise the theory’s potential to explain insecure behaviours. Phase one implications and identification of latent psychological dimensions reduce soft skill ephemerality and speak to improving software learning development processes. Phase two implications speak to enhanced theoretical understandings of how social and cognitive psychology can explain secure coding behaviours, practical applications of raising awareness of dangers of intuitive mindsets and improving responsibility through freelance platform gamification, and research implications such as power analyses. This thesis advances our current understanding of the human element in secure coding.

## Contents

<b>Declaration</b>	<b>2</b>
<b>Acknowledgments</b>	<b>3</b>
<b>Abstract</b>	<b>4</b>
<b>Statement of Authorship</b>	<b>9</b>
<b>List of Tables</b>	<b>11</b>
<b>List of Figures</b>	<b>12</b>
<b>1 Introduction</b>	<b>16</b>
1.1 Research Questions . . . . .	17
1.2 Thesis Construction . . . . .	18
1.3 Thesis Creation . . . . .	20
1.4 Rationale for Multi-Part Format . . . . .	20
1.5 Personal Contributions . . . . .	21
<b>2 Literature Review</b>	<b>22</b>
2.1 Current Landscape . . . . .	22
2.2 Software Security . . . . .	24
2.2.1 Tooling . . . . .	25
2.2.2 Artifacts . . . . .	26
2.3 Applications of Psychology . . . . .	27
2.3.1 Cognition . . . . .	28
2.3.2 Social . . . . .	32
2.4 Soft Skills . . . . .	35
2.5 Other Psychological Perspectives . . . . .	37
2.6 Summary . . . . .	39
2.7 Methodological Motivations . . . . .	40
2.7.1 Overarching Methodological Commitments . . . . .	40
2.7.2 Research Designs . . . . .	42
2.7.3 Analytical Methods . . . . .	45
2.8 Open Science . . . . .	49
2.8.1 Preregistrations . . . . .	49
2.8.2 Data Sharing . . . . .	50
2.8.3 Computing Environments . . . . .	51
2.8.4 Preprints . . . . .	51
2.8.5 Open access . . . . .	51
2.8.6 Summary . . . . .	52
<b>3 The Soft Skills of Software Learning Development: The Psychological Dimensions of Computing and Security Behaviours</b>	<b>53</b>
3.1 Statement of Continuous Thesis Summary . . . . .	60

<b>4</b>	<b>What's in an undergraduate Computer Science Degree; Alumni perceptions about soft skills in careers</b>	<b>61</b>
4.1	Statement of Continuous Thesis Summary . . . . .	91
4.1.1	Contribution to Thesis Argument and Forward Trajectory . . . . .	92
<b>5</b>	<b>Everything but Programming; Investigating Academics' Perceptions of Embedded Soft Skills in Computer Science Undergraduate Education</b>	<b>94</b>
5.1	Statement of Continuous Thesis Summary . . . . .	126
5.2	Synthesising Phase 1 . . . . .	126
5.2.1	Communication and Teamwork . . . . .	127
5.2.2	Problem Solving and Critical Thinking . . . . .	129
5.2.3	Contribution to Thesis Argument and Forward Trajectory . . . . .	132
<b>6</b>	<b>Recognising The Known Unknowns; the interaction between reflective thinking and optimism for uncertainty among software developer's security perceptions</b>	<b>134</b>
6.1	Statement of Continuous Thesis Summary . . . . .	151
6.1.1	Contribution to Thesis Argument and Forward Trajectory . . . . .	152
<b>7</b>	<b>Can You Hear The ROAR of Software Security? How Responsibility, Optimism and Risk shape developers' security perceptions</b>	<b>154</b>
7.1	Statement of Continuous Thesis Summary . . . . .	193
7.1.1	Contribution to Thesis Argument and Forward Trajectory . . . . .	194
<b>8</b>	<b>Software Vulnerabilities as Cognitive Blindspots; assessing the suitability of a dual processing theory of decision making for secure coding</b>	<b>196</b>
8.1	Statement of Continuous Thesis Summary . . . . .	222
8.1.1	Contribution to Thesis Argument . . . . .	223
<b>9</b>	<b>General Discussion</b>	<b>225</b>
9.1	Summary of Results . . . . .	226
9.2	Answering Research Questions . . . . .	227
9.3	Theoretical Implications . . . . .	229
9.3.1	Soft Skills and Pedagogy . . . . .	230
9.3.2	Cognitive Psychology Applications . . . . .	230
9.3.3	Social Psychology Applications . . . . .	231
9.4	Practical Implications . . . . .	232
9.4.1	Curriculum Design . . . . .	233
9.4.2	Closing the Temporal Gap . . . . .	234
9.4.3	Rewards for Secure Coding . . . . .	234
9.4.4	Increasing accountability . . . . .	234
9.4.5	Educating engineers on decision making styles . . . . .	235
9.5	Final Thesis Structure . . . . .	235
9.6	Reflections on Methodological Coherence and Theoretical Integration . . . .	237
9.7	Limitations . . . . .	238
9.8	Further Work . . . . .	240

9.9 Conclusion . . . . .	242
<b>Consolidated Bibliography</b>	<b>244</b>



### Statement of Authorship

**Chapter Title:** The Soft Skills of Software Learning Development: The Psychological Dimensions of Computing and Security Behaviours

**Publication Status:** Published

**Published as:** Ivory, M. (2022). The Soft Skills of Software Learning Development: The Psychological Dimensions of Computing and Security Behaviours. Proceedings of the International Conference on Evaluation and Assessment in Software Engineering 2022, 317–322. <https://doi.org/10.1145/3530019.3535344>

**Principal Author:** Matthew Ivory

**Contribution:** Theoretical conceptualization; manuscript development; manuscript revisions based on supervisor and peer-reviewer feedback.

**Chapter Title:** What’s in an undergraduate Computer Science Degree; Alumni perceptions about soft skills in careers

**Publication Status:** Under Review

**Principal Author:** Matthew Ivory

**Contribution:** Theoretical conceptualization; manuscript development; manuscript revisions based on supervisor feedback.

**Chapter Title:** Everything but Programming; Investigating Academics’ Perceptions of Embedded Soft Skills in Computer Science Undergraduate Education

**Publication Status:** Submitted

**Principal Author:** Matthew Ivory

**Contribution:** Theoretical conceptualization; manuscript development; manuscript revisions based on supervisor feedback.

**Chapter Title:** Recognizing the Known Unknowns; the Interaction Between Reflective Thinking and Optimism for Uncertainty Among Software Developer’s Security Perceptions

**Publication Status:** Published

**Published as:** Ivory, M., Towse, J., Sturdee, M., Levine, M., & Nuseibeh, B. (2023). Recognizing the Known Unknowns; the Interaction Between Reflective Thinking and Optimism for Uncertainty Among Software Developer’s Security Perceptions. Technology, Mind, and Behavior, 4(3: Winter 2023). <https://doi.org/10.1037/tmb0000122>

**Principal Author:** Matthew Ivory

**Contribution:** Theoretical conceptualization; manuscript development; manuscript revisions based on supervisor and peer-reviewer feedback.

---

**Chapter Title:** Can you hear the ROAR of software security? How Responsibility, Optimism And Risk shape developers’ security perceptions

**Publication Status:** Under Review

**Principal Author:** Matthew Ivory

**Contribution:** Theoretical conceptualization; manuscript development; manuscript revisions based on supervisor and peer-reviewer feedback.

---

**Chapter Title:** Software Vulnerabilities as Cognitive Blindspots; assessing the suitability of a dual processing theory of decision making for secure coding

**Publication Status:** Submitted

**Principal Author:** Matthew Ivory

**Contribution:** Theoretical conceptualization; manuscript development; manuscript revisions based on supervisor feedback.

---

Signed confirmation that the above information pertaining to author contribution is correct

---

Matthew Ivory	John Towse	Mark Levine	Miriam Sturdee	Bashar Nuseibeh
---------------	------------	-------------	----------------	-----------------

---

## List of Tables

- 3.1. Coefficients, t-values and p-values for the linear regression of CRT predicting OWASP vulnerability.
- 4.1. Values of the factor loadings of the final EFA, ordered by factor eigenvalue and item loading strength.
- 5.1. Relevant participant demographics, including their position within the department and the delivery year of core modules.
- 5.2. The skills identified in Chapters 4 and 5 as being important for software learning development. 5a refers to study one and 5b to study two in Chapter 5.
- 6.1. Reported Ages of Participants Included in the Analysis.
- 6.2. Qualitative Questions Presented to Participants in the Order Listed.
- 6.3. The Raw Percentage Agreement Between the Primary Researcher and the Two Researchers Who Completed the Validation Task.
- 6.4. Transformation of Optimism Task and PURL Scores Toward a Normal Distribution.
- 6.5. Model Coefficients of the Terms Used in the Modelling of the Presence of Uncertainty-Related Language.
- 6.6. Chosen Model for the Second Step Model Using Zero-Truncated PURL as a Dependent Variable.
- 7.1. Reported age of participants.
- 8.1. Demographic breakdown of participant ethnicity.
- 8.2. The presentation order and attributes of the Python puzzles presented to participants. The puzzle ID relates to the original ID given in Brun et al. (2023) and cyclomatic

scores are a measure of code complexity with higher scores indicating more difficult code to comprehend or modify.

8.3. Percentage of correctly solved answers compared to the sample from Frederick (2005).

8.4. Potential effect sizes and samples reported from the simulation and power analysis.

\* = this duplicated measure is the result of it being included in a second model from hypothesis 3, \*\* = the values for cybersecurity are based upon linear regression predictions and assume a linear relationship between power, coefficient, and sample size.

9.1. The skills identified in Chapters 4 and 5 as being important for software learning development. 5a refers to study one, and 5b to study two in Chapter 5. This graph is replicated from the Statement of Continuous Thesis Summary in Chapter 5.

### **List of Figures**

2.1. A bibliometric analysis of the co-occurrence of keywords taken from the collection of research publications spanning the PhD project.

3.1. Outline of research phases within the planned work.

3.2. Transmission of shallow skills can be expected to develop whereby teaching staffs' understanding of which skills are important feed into the skills students pick up on, which are reflected in alumni use of these skills.

3.3. Box plot of mean OWASP vulnerability scores by CRT score split by population.

4.1. Histogram of undergraduate graduation year split by department. Bars are grouped into ten-year intervals.

4.2. Histogram of age and gender splits. Bars are grouped into five-year intervals.

4.3. Survey flow with CS alumni filling out an additional section regarding their expe-

rience working with security. This focussed on whether their employment was linked to security work, and how much of their work is concerned with security.

4.4. Correspondence Analysis plotting skill importance between alumni groups. Dimension one can be interpreted as importance and explains 84.99% of variance, with the second dimension interpreted as differenced in department explaining only 10.19% of variance.

4.5. A representation of the absolute difference in distance and direction between alumni for each skill. As dimension one explains so much variance, omitting dimension two still provides a strong representation of the skills reduced to a single dimension.

4.6. Plot of the fitted probabilities for all skills with significant differences between the two departments. Each individual plot indicates the differences between alumni responses for each rating level.

4.7. Joint correspondence analysis of skills and their perceived originating source as either university, through hobbies or through work, split by department. Only items with an associative strength greater than .25 are included in the plot for clarity. Arrows indicate the angles of association.

4.8. Associations of skills for each source of development. Items plotted closer towards the zero axis (the top) have a smaller associative angle, indicating a stronger association, and items further from the centre are more strongly associated. A strong cluster of positively connected skills are seen in the arc between 0 and 10, indicating their strong association to their respective sources. Skills seen to be closely associated with a source are within the righthand side of the positive angles, and skills that are not associated with a particular source have negative associations and are seen towards the lefthand side.

4.9. Frequencies of each reported skill type per participant.

4.10. Soft skills mentioned within the answers of which technical skills were taught and used in employment. Items mentioned once only are omitted.

4.11. A split-by-skill representation of each skill rated for CS alumni's security usage in employment. Items closer to the centre or each other indicate minimal differences in the way skills are used around security. Dimension one provides over 70% of variance and so items located further along the x-axis indicated greater perceived importance.

4.12. The source of development for the top five skills reported by CS alumni.

4.13. A plot highlighting the skills that had significant differences between the two alumni populations.

5.1. A stacked column plot indicating the most mentioned soft skills across all curricula. Skills mentioned only once in an institution's curriculum were removed.

5.2. Separating the frequencies by soft skill shows the patterns (or absence of) across institutions. Only the top six skills are shown.

5.3. Presenting soft skills split by their institution offers a different perspective, showing how institutions present different skills within their curricula.

5.4. Skills that occur more than three times are included, and only skills that occur more than five times are labelled. This offers a representation of how soft skills are referenced within curricula information. Terms with no contextual value, such as 'computer science' and 'module' are removed as they overwhelm the plot due to their high frequency.

5.5. Network plot of the soft skills and co-occurrences through modules. The connections indicate skills that are taught in the same modules. Yellow nodes are soft skills, and dark purple nodes are individual modules

5.6. Proportion of modules within an institution that includes the top six reported soft

skills.

5.7. Inspection of soft skills by course structure (module years), no distinct pattern is observed across skills.

6.1. Map Displaying Approximated Location/Nationality Data of Participants on a Continent Level.

6.2. Survey Flow Representing the Presentation Order of Measures Given to Participants.

6.3. Distribution of the PURL Scores Before and After Transformation Toward a Normal Distribution.

6.4. Interaction Plot Between CRT Score (the Propensity for Reflective Thinking) OVT Score (Optimism Bias) and Model Predicted Values.

7.1. Map displaying the combined location and nationality data of participants on a continent level. This represents an approximation of both location and nationality of each participant due to typically low immigration movement. Most participants were from North America, Europe, or Asia.

8.1. An example of a puzzle containing a blindspot. All puzzles were formatted similarly, with the context given first, followed by an image of the code, and then asked to describe the code's behaviour. Each puzzle was prefaced with the scenario as context. The context would explain the general setup and use of the code section and provide examples, if necessary, of its use case. It also noted that readers should assume all necessary permissions are given for execution.

8.2. Study pipeline that participants experienced. All participants experienced the same survey flow with randomisation in some sections.

## 1 Introduction

Software has a significant role in modern society and is relied upon for many vital tasks. Financial transactions, medical patient records, and nationally sensitive information depend on software security to conduct actions with integrity and without interference. Software security is invaluable for software, as malicious tampering or accidental access to sensitive information can be damaging and potentially cause personal harm (Palassis et al., 2021). Accordingly, software engineers need to mitigate vulnerabilities within software.

Despite the importance of security, we are constantly identifying insecure software in production-quality software. Over 80% of Android applications contain cryptographic errors (Egele et al., 2013; Weir et al., 2020b), and many of the vulnerabilities are well-known, with 70% of software containing one of the ten most common vulnerabilities (Veracode, 2023), with some vulnerabilities having been common knowledge for almost two decades. Awareness alone does not ensure software security, so other factors must influence its existence (Rauf et al., 2021).

Software is made by people for people, and it can be considered a by-product of human behaviour (John et al., 2005). Software engineers' cognitive and social dimensions influence functionality and security throughout software development. The psychological influences can result in irrational behaviours leading to security not being appropriately accounted for in software. As a result, developers' behaviour can be positioned as a root cause for the presence of vulnerabilities in software.

Decision making and the manifestation of behaviour can be examined through psychological theories and measures, allowing for an increased understanding of influences. Psychology in software development is not a new idea reaching as far back as the 1970s (Weinberg, 1971), with a research focus on secure programming environments more recently (Graff & Wyk, 2003). Despite this, technological developments have typically dom-



inated software engineering (Lenberg et al., 2014), and technical interventions have been deployed in hopes that technology can remove human influence. Exceptional software engineers need more than just technical skills of programming and systems knowledge (Destefanis et al., 2017), requiring interpersonal skills and critical thinking skills as well (Groeneveld et al., 2020b). Further research is still needed to understand the role of psychology within software engineering.

To best explore the role of psychological theories in secure software development, it is beneficial to start from a high-level perspective by exploring the perceptions held by relevant populations. To achieve this aim, the “soft skills” or non-technical skills that support the application of technical skills should be targeted. Soft skills are a more familiar idea in software engineering as a common concept of the required skills outside of programming that contribute to project success. Soft skills are the observable manifestations of latent psychological structures. Once we identify the valuable skills, the psychological structures can be extracted and measured to enhance our understanding of security behaviours.

### **1.1 Research Questions**

This PhD seeks to understand whether theories of decision making and social identity can explain developers’ computing and security behaviours within software engineering. I present two research questions that structure the thesis:

1. What non-technical skills (or soft skills) are valued within Computer Science and software engineering?
2. Can psychological variables and constructs shape security behaviours within software engineering?

To answer these, I present a series of chapters exploring how social and cognitive factors influence secure software development. The thesis is separated into two phases, with phase 1 providing important information about soft skills valued within Computer Science (CS) education, followed by translating these findings into more psychologically

grounded theories. These dimensions are then tested in phase 2.

Phase 1 comprises Chapters 4 and 5, which address the soft skills delivered during a CS higher education. Psychology theories are not well established within CS or software engineering, so it is unclear which theories are best deployed for maximal effect. To mitigate this issue, I leverage valued soft skills within these domains. They are a familiar concept for students and professionals, meaning the skills can be used to identify the relevant latent psychological theories that can help us determine the influence of developer's psychology for secure software development.

Phase 2 (Chapters 6, 7, and 8) targets the second research question. The psychologically motivated ideas derived from phase 1 are the Social Identity Approach (Haslam, 2012) and the dual processing theory of decision making (Evans, 2003), which are applied towards security perceptions and software vulnerability detection. These studies demonstrate the value of these theories within software engineering and contribute to the understanding of the psychological dimensions of computing and security behaviours.

## **1.2 Thesis Construction**

This thesis is structured as follows:

In Chapter 2, I review relevant literature that supports the research chapters. As part of the multi-part format, where each research chapter is an individual manuscript submitted to peer-reviewed venues, there is a need for concision that results in the introduction sections being selective in the literature covered. Depending on the venue audience, certain ideas or concepts are given reduced space simply because they are assumed knowledge.

Chapter 2 addresses three main concepts. I review software security issues in general and how software and its artifacts result in insecure code. I then cover relevant cognitive and social psychological theories investigated within the secure software development domain. Finally, I review soft skills within software engineering and education. I then present a methodology review covering the core research designs, analytical approaches, and the

open science approach adopted throughout the thesis. Again, including methodological reviews are not part of the expected format of typical research papers, so I take the opportunity to provide details that are intentionally omitted from later chapters.

Chapter 3 was presented at a doctoral symposium and served as the primer for the anticipated research to be carried out during the PhD and outlines the proposed work and research justifications. This chapter effectively preregisters the PhD project. The differences between the expected and actual outcomes are discussed in Chapter 9.

Chapter 4 is the first half of phase 1, where I explore the perceptions of CS and Psychology graduates to understand the non-technical skills considered valuable for their careers. Graduates reported where these skills were primarily developed, offering an insight into how non-technical skills fit into their personal development. The Psychology sample is used as a comparative sample, highlighting the differences between the two degrees. The findings suggest that graduates value skills of *problem solving*, *communication*, and *teamwork* above others. They also report that only *problem solving* is seen to be associated with their education. A focus on those working in security roles highlighted *organisational* skills over the general developer population.

Chapter 5 is the second part of phase 1 and explores staff perceptions of how non-technical skills are integrated into their curriculum materials. Through a multi-site interview and text analysis of curriculum data, these two studies explore staff perceptions on soft skills and how they are presented to students. These perceptions are then compared against the soft skills extracted from curriculum content. The findings identify an alignment between employers and educators, suggesting that the proposed soft skills gap (Akdur, 2021) results from how students value soft skills. The findings from Chapters 4 and 5 suggest that the psychological underpinnings of social skills of *groupwork*, *communication*, and cognitive skills of *critical thinking* and *problem solving* warrant further investigation.

The thesis then takes the findings from phase 1 and translates the valued soft skills into their potential latent psychological traits. These traits are explored in phase 2.

Continuing with exploring perceptions, Chapters 6 and 7 deploy a quantitative and qualitative analysis on the same dataset. The study gathered data regarding security perceptions within software development from software engineer freelancers and current CS students. Initially, these analyses were planned to be written as a single chapter, but it was decided that separate chapters were needed to preserve the separate implications. Chapter 6 suggests that a complex relationship between cognitive reflection and optimism exists with increased security awareness. Chapter 7 indicates that developers discuss responsibility, optimism, and risk diversely across secure software development processes.

Chapter 8, the final research chapter, tests the hypothesis that software vulnerabilities exist as blindspots in software developer's cognition. I interpret this through dual processing theory and apply intuitive and reflective thinking measures in a code comprehension task. This chapter uses a modest sample to simulate data for power analyses and assess appropriate samples required for software engineering research. The findings suggest that dual processing theory may be appropriate for secure software engineering.

Chapter 9 concludes the thesis, where I discuss the findings and implications, along with the limitations and future work.

### **1.3 Thesis Creation**

The thesis is fully reproducible, with necessary documents, images, and files stored in an open-access repository: <https://osf.io/2fpbq/>. The repository contains links to all the research chapter components, providing easy access to all the work packages.

### **1.4 Rationale for Multi-Part Format**

The studies presented in Chapters 3-8 are written as research papers; all chapters are published or submitted to peer-reviewed venues.

- Chapter 3 is published in the conference proceedings of The International Conference on Evaluation and Assessment in Software Engineering (EASE) 2022.
- Chapter 4 is under review at ACM Transactions on Computing Education.
- Chapter 5 is under review at ACM Transactions on Computing Education.
- Chapter 6 is published in APA Technology, Mind & Behaviour.
- Chapter 7 is under review at Springer Empirical Software Engineering.
- Chapter 8 is under review at ACM Transactions on Software Engineering and Methodology.

The multi-part format was chosen given the interdisciplinary nature of the research and the valuable contributions each chapter makes, and the chapters are presented as distinct research papers. Together, they provide a cohesive narrative that answers the research questions posed in this thesis. Most chapters target software engineering audiences rather than psychological ones because, whilst the content is very much psychological, the relevant audiences are primarily those carrying out research in software engineering.

### **1.5 Personal Contributions**

For this thesis, I was responsible for the majority of the work undertaken across the thesis; this included theoretical conceptualisation, study design, ethics, data collection, data analysis, primary manuscript writing and drafting, manuscript submissions, and responding to the peer-review process. This was under the guidance of my supervisors, and undoubtedly, the thesis would not be in the state it is without them: Professor John Towse, Professor Mark Levine, Dr Miriam Sturdee (Lancaster University & St. Andrews University), and Professor Bashar Nuseibeh (of Open University & Lero Institute). Contributions outside of this are noted in the appropriate chapter acknowledgements.

## 2 Literature Review

This thesis explores the interdisciplinary space between psychology and software engineering, specifically exploring the social and cognitive influences on secure software development. In this chapter, I contextualise the thesis by reviewing relevant research around secure software engineering, psychology implementation in software engineering, and soft skills in software development. A visual representation of the research is presented by constructing a bibliometric network plot (Figure 1), followed by an in-depth review of specific papers. As the thesis is in the multi-part format, each chapter has its own literature review, so this section intentionally omits content presented in later chapters. I then provide a methodological review, where I discuss paradigms and approaches, followed by open science practices highlighting the less visible components enacted.

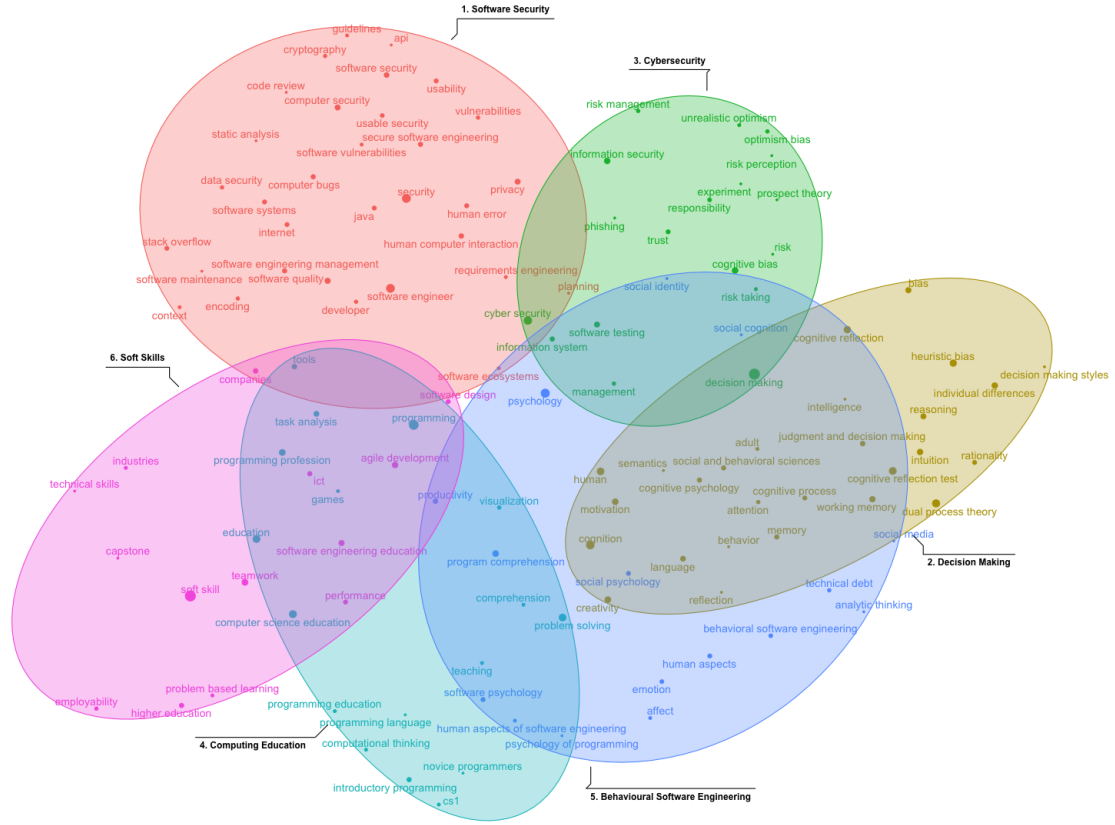
### 2.1 Current Landscape

A bibliometric analysis visualises the current research landscape, offering a quantitative approach to understanding the relationships between publications (Donthu et al., 2021). In doing so, the research is placed in reduced dimensional space, offering a high-level overview of their connectivity.

Keyword co-occurrences were used to represent relationships between publications that share keywords. Co-occurrences were calculated with VOSviewer (Eck & Waltman, 2009) using 1,164 publications collected between October 2020 and December 2023. Keywords were normalised to British English spelling, and similar terms were combined (e.g. “dual process model” and “dual-process theory”). Terms that appeared more than five times were retained, and manual inspection removed terms unrelated to the thesis (e.g., neuroscience terms). The files for analysis reproduction are available at <https://osf.io/2fpbq/>.

The network in Figure 1 contains six clusters. The following observations can be made:

1. The cluster *Software Security* shares minimal variance with others, indicating its relative distinctiveness as a domain based on the publications collected.



**Figure 1**

*A bibliometric analysis of the co-occurrence of keywords taken from the collection of research publications spanning the PhD project*

2. The clusters of *Decision Making* and *Behavioural Software Engineering* share some overlap, but the terms are minimally co-located. That is, terms from *Decision Making* are not interspersed amongst the terms in *Behavioural Software Engineering*; instead, a small number of terms cause the overlap. It highlights that human aspects of software engineering are being carried out using psychological frameworks. *Behavioural software engineering* is closer to *Computing Education* and terms relating to novice programmers.
3. The separation between *Decision Making* and *Software Security* suggests these areas have little co-occurrence or exposure. It does not indicate an absence, as researchers have conducted work in this space (e.g., Brun et al., 2023), but instead suggests

minimal exposure. A higher co-occurrence between key terms would result in more proximal distributions. The distance between these clusters suggests that more work is needed regarding psychological theories applied towards software security.

4. The clusters of *Computing Education* and *Soft Skills* overlap, suggesting soft skills research is applied in educational contexts.

It is worth noting that keywords are author-generated and not a standardised process, meaning that difference in terminology across disciplines may influence the plot. As the collation of research articles was personalised to the thesis, it was shaped by the research questions and how the search and citation acquisition occurred. Despite this, the network shows clustering and relationships between words. The main takeaway is that social and cognitive psychology are poorly associated with software engineering security research. My thesis aims to increase the overlap between the clusters, and in doing so, the research explores uncharted ground.

## 2.2 Software Security

Relevant software security research is split into topics of *Tooling*, which refers to the tools available to software engineers, such as Application Programming Interfaces (APIs) and how developers interact with these, and *Artifacts*, which covers the materials that support software use (such as documentation) and how their presence and usability impact secure coding practices. This section is restricted to focusing on software engineers rather than others in the development process (e.g., users or non-technical stakeholders) to ensure it has maximal relevance in framing the thesis.

As articulated in Ivory et al. (2023c), software vulnerabilities are the unexpected logic flows that create software behaviour that enables unintended access to information or functionality. Vulnerabilities are pervasive within software engineering with serious ramifications such as impact on medical care (Smart, 2018). Many vulnerabilities detected are not new, and 70% of software contains a common vulnerability (Veracode, 2020). This



issue indicates that the support given to software engineers is inadequate, and developers must practise secure coding to reduce the presence of vulnerabilities. Secure coding is defined as the practice that ensures software “does not contain known vulnerabilities” (Rauf et al., 2021).

### ***2.2.1 Tooling***

Tools can improve security practices by raising the salience of security issues, such as interactive tools that highlight insecure code in real-time (Zhu et al., 2014), which reduces the temporal distance between writing and identifying issues (Oliveira et al., 2014). It is important to note that the mere existence of these tools does not automatically mitigate security issues, but they must be used with the intent of achieving secure code (Witschey et al., 2015). Security is just one of many competing interests within the development cycle, with other goals including functionality, usability, and privacy (Rauf et al., 2021). These competing goals may reduce the obligation or intention to ensure security, which can often be more salient to developers.

When developers deviate from tool default behaviour to suit their personal preferences (Danilova et al., 2020), this can result in the suppression of security warnings when developers consider them annoying, irrelevant, or even inaccurate (Gorski et al., 2020). Developers may view this personalisation positively, but if they experience memory lapses and forget to reactivate settings to check security, the software may be assumed to be secure when it is not (Anu et al., 2020).

Tool diffusion occurs through communities (Rogers, 1995), and adoption of new tools is higher when previous experiences were positive (Witschey et al., 2014). Otherwise, developers select recognisable tools regardless of security implementation, indicating a familiarity bias. Tool diffusion relies upon curiosity (Weir et al., 2016), and diffusion can be constrained by institutional procedures that delay procurement (Xiao et al., 2014). Security’s reactive nature is seen in that the motivation to adopt new tools is linked to an increased perception of vulnerability or need for security compliance (Ifinedo, 2012).

Developers are often *assumed* to code securely by default (Wurster & van Oorschot, 2008) but are actually *observed* to prioritise functionality over security (Assal & Chiasson, 2019). This functionality-first mindset is evidenced through the event-driven nature of security implementation (Lopez et al., 2019a). For example, waiting until issues are discovered, or stakeholders requesting fixes (Gutmann, 2002). The mindset is partly borne from a reliance on APIs to enact secure functionality (Lopez et al., 2020a) and an assumption that tools are secure by default (Xie et al., 2011). This assumption can lead to a reduced sense of responsibility, as developers deflect blame towards the API when software vulnerabilities are detected (Wijayarathna & Arachchilage, 2019). Developers are often reluctant to update API libraries in their software (Kula et al., 2018) because of concerns about breaking functionality, which would increase the workload associated with performing security updates (Huang et al., 2019).

### 2.2.2 *Artifacts*

Artifacts support software use, and include manuals and documentation (Nazar et al., 2016). In the context of the thesis, “artifact” relates to the by-products of software and not research which includes prototypes or publications (Winter et al., 2022). In user security studies, system security is constrained by user behaviour. Users are not motivated to engage in security practices in addition to their primary goal (Pattinson et al., 2015), with security being viewed as a barrier (West, 2008). Rather than expect voluntary changes in behaviour, it is better to design software that mandates security by design. Despite these findings, they are infrequently applied to software developers (Green & Smith, 2016), particularly for API developers, who provide the tools for developers (Acar et al., 2017a).

Developers are reliant on API developers to create usable documentation (Nadi et al., 2016). Where documentation usability is low, developers often produce functional but insecure code (Acar et al., 2017c). API developers must recognise that their users – software developers who are not always experienced in secure coding – must be accommo-

dated and not treated as security experts (Amin & Bhowmik, 2021). API documentation should incorporate easy-to-use examples and indicate where default function behaviour is insecure (Patnaik et al., 2022). The difference between expertise expected by API developers and actual user expertise can result in unusable documentation (Pieczul et al., 2017). If official artifacts are not usable, they are an additional strain on the developer’s attention and workload. This strain leads to frustration, which drives developers to seek usable information elsewhere (Gorski et al., 2020).

Q&A forums, such as StackOverflow, offer more readily-used solutions to software tasks, but their community-based nature may mean answers are provided by non-experts, and security professionals do not typically vet these sites. StackOverflow answers offer more insecure code than official documentation (Acar et al., 2016a), with a quarter of questions only receiving insecure solutions (Acar et al., 2017b). These responses are not simply due to well-intentioned novices submitting insecure solutions, as even experienced users submit insecure solutions (Rahman et al., 2019).

The ability of a concerned developer to actively check proposed answers for security is limited, as many accepted answers possess no external links to support the answer (Tahaei et al., 2020b). Answers are often selected based on non-code features, such as description details (van der Linden et al., 2020b), meaning superficial criteria are used for decision making. This behaviour is not unexpected because those asking questions are doing so out of necessity (since they do not have a solution) and likely do not possess the knowledge to assess the technical aspects of an answer, so the issue is with the information provided. Insecure StackOverflow solutions are present in over 15% of Android applications (Fischer et al., 2017), suggesting these are directly copy-pasted into projects without security considerations.

### **2.3 Applications of Psychology**

In recent years, psychology has been increasingly applied within software engineering. Applying psychology to programming is not a novel idea; it was first mentioned in the

1970s, with psychology being applied to software engineering more broadly in the 2000s (Blackwell et al., 2019).

This thesis sits within the growing field of human-centred software engineering, contributing to a tradition that includes work by Blackwell et al. (2001), Petre (2009) and Whittle et al. (2021) on embedding human values into software practice. These foundational contributions have advanced the understanding of how cognition, values, and user impact shape software engineering, with a particular emphasis on tool interaction, design cognition, and ethical alignment. Building on this, the current thesis extends the human-centred software engineering agenda by focusing on psychological theories (specifically dual processing theory and social identity theory) to investigate how developers' cognitive and social makeup influences secure software behaviours. Rather than focusing on previously well-studied concepts such as the cognitive dimensions of notation framework, it adopts a behavioural lens to examine how psychological and social-cognitive mechanisms inform computing and secure computing behaviours. This complements existing human-centred software engineering literature by deepening the behavioural science perspective and offering a dual-theoretical framing - cognitive and social - that remains underexplored in the domain. In doing so, it strengthens our understanding of secure software engineering not only as a socio-technical enterprise but as a psychologically-grounded human activity, offering new empirical and theoretical contributions to the human-centred software engineering landscape.

### ***2.3.1 Cognition***

This section reviews cognitive biases, working memory and cognitive load, cognition-oriented interventions, and early mentions of dual processing theories. Each section overlaps with the thesis research, supporting the claims made. Neighbouring research areas, such as neuroscience programming research, are omitted as they do not contribute to the thrust of the thesis.

**2.3.1.1 Cognitive biases.** Cognitive biases are systematic deviations from expected decision making performance (Evans, 1984). They are borne from cognitive heuristics, the “mental shortcuts” used to arrive quickly at optimal or near-optimal decisions (Beike & Sherman, 1994). Biases result in flawed judgments when heuristics are systematically applied to ill-suited contexts, and software developers are no different from the general population regarding their susceptibility to biased thinking (Ralph, 2013). When errors are missed, they can become vulnerabilities (Patel et al., 2023), and the lack of awareness over biases results in non-optimal decisions being made (De Wit et al., 2021).

Within software engineering, biases have been identified across the development cycle (Ralph, 2013), and mitigating biases is not always straightforward, often requiring unique interventions for specific biases (Mohanani et al., 2020). Biases influence the tools and methods that developers choose, and developers typically prefer familiar tools (Sergeyuk et al., 2023), even when they are ill-suited for a particular task (Anu et al., 2020). No single tool is ideal for every task, and no single tool alone can detect all vulnerabilities (Elder et al., 2022), so preference for familiar tools can lead to biased decisions.

Biases are notoriously persistent and difficult to combat (Petre, 2022). Encouraging developers to think reflectively (Chattopadhyay et al., 2022) can increase attentional focus towards security (Anu et al., 2016). Another method is to reduce the complexity of the environment, requiring fewer variables to be evaluated (Nagaria & Hall, 2020).

**2.3.1.2 Working Memory and Cognitive Load.** Working memory is a cognitive system that holds information for guiding judgment-making (Baddeley, 1986; Jarrold & Towse, 2006). It comprises several mental components controlled by a central executive system that directs attentional focus (Baddeley & Hitch, 1974), effectively managing short- and long-term memory (Chai et al., 2018).

Working memory capacity has been suggested to have a major influence on programming learning (Prat et al., 2020). Working memory is required during code comprehension to

maintain a mental model of the current code state, and as code increases in complexity, developers likely forget or confuse information (Crichton et al., 2021). Working memory capacity and recall are associated with the ability to detect non-localised code errors (Baum et al., 2019), meaning higher capacity allows developers to understand more complex non-linear code. Others find working memory capacity is mediated by programming ability, suggesting a limited role of working memory itself (Bergersen & Gustafsson, 2011).

Humans do not possess a software security-specific working memory component (Oliveira et al., 2014), meaning that to code securely, it must be attended to fully, but security is just one of many competing goals (Rauf et al., 2021). Working memory has been applied to security contexts with mixed results. Oliveira et al. (2018) tested working memory against vulnerability detection in Java APIs and found no meaningful effect; however, the same measures applied to Python code found an association between long-term memory recall and vulnerability detection (Brun et al., 2023), suggesting a possible language-specific association.

Cognitive load is the amount of information that working memory can process at any given time. It comprises three components: intrinsic, extrinsic, and germane load. The inherent difficulty of a task relates to intrinsic load, extrinsic load is the presentation of the information, and germane load is the processing effort needed to translate the task information into internal models (Sweller et al., 1998). Many software tasks have a high intrinsic load due to the nature of software development (Sedano et al., 2017), and so extrinsic and germane loads are the only aspects that can be reduced (Gonales et al., 2021). As software grows, new functionality is often appended to existing code and made to fit rather than optimised, which increases intrinsic load (Helgesson, 2023).

Working memory’s influence on software development may not be as easily detected as it is elsewhere because of the development environment. Tools may be used to reduce

environment complexity (Brachten et al., 2020) by reducing extrinsic load by offloading information to technology that would otherwise overwhelm working memory (Zayour et al., 2013). For example, checklist strategies can reduce cognitive load when conducting code reviews (Gonçalves et al., 2020) or by assigning information to variables.

**2.3.1.3 Interventions.** The success of boosting interventions in software engineering offers support to the notion that cognition plays a significant role in secure coding. When unprompted for security across different experience levels, it is infrequently implemented, but security requests increase secure coding (Naiakshina et al., 2018). The situation is more complex than just “being aware of security means it is implemented” because when asked about insecure solutions submitted, some developers refused to correct vulnerabilities without additional compensation (Naiakshina et al., 2019). Hallett et al. (2021) replicated the success of boosting interventions, indicating that many developers can code securely but (consciously or unconsciously) only do so if adequately motivated. Even when developers indicate personal responsibility for security, it is not addressed unless prompted (Danilova et al., 2021). Lightweight interventions such as workshops can improve security motivations by providing information that may not have previously existed (Weir et al., 2020a), allowing teams to recognise new approaches and reflect on their potential. The broadening of perspectives can be beneficial even in groups without security expertise (Shreeve et al., 2022).

**2.3.1.4 Dual Processing Models.** The dual processing theory of decision making (Evans, 2003) suggests that humans make judgments using one of two systems. The default-interventionist psychological model of dual processing theory proposes two cognitive systems (Evans & Stanovich, 2013): the default and *intuitive* system 1 and a more cognitive and *rational* system 2 that only engages when sufficiently cued (Damjanović et al., 2019). System 1 processing is the default for decision making, driven by heuristics to reduce complex judgments into simpler operations (Kahneman et al., 1974). In contrast, system 2 processing is deployed when individuals seek an optimal solution by using all available information but demands more cognitive effort – and as a result, it is used spar-

ingly. System 2 is interventionist and only overrides system 1 when a need for accuracy is detected. System 1 is liable to generate simpler and less complete mental models than system 2 (Johnson-Laird, 2010), and blindspots can reside in the gaps created.

So far, dual processing theory has been minimally introduced to software engineering, primarily highlighting its theoretical potential (Petre, 2022; Pretorius et al., 2018). Lowe (2019b) reinterpreted previous data through dual processing theory without capturing measures of it and presented a potential application within computing education (Lowe, 2019a). No models of cognition for software engineering have accounted for dual processing, meaning these models do not differentiate between the two processing systems (Robins, 2022). One empirical study has been conducted with student populations for defect detection in code (Buffardi, 2023); it was seen that more intuitive thinking styles correlated with increased acceptance of code containing defects, suggesting that more reflective decision making is related to more frequent detection of defects. To the author’s knowledge, empirical work has yet to be deployed in security contexts.

### **2.3.2 Social**

Here, I review social psychological applications towards software engineering, focusing on social identity. The social identity approach posits that to understand social behaviours and individual perceptions, we cannot focus on the individual as an *individual*, but we should focus on how behaviours are tied to *social identities*, which are a person’s self-definitions of group memberships (Haslam, 2012). Social identity describes group-based behaviours and how individuals ascribe group membership (Tajfel & Turner, 2010). Social identities are self-defined groups that allow individuals to associate with others who share emotionally significant values (Abrams & Hogg, 1990). Group members form perceptions of others who share membership (the ingroup) and those who do not (the outgroup). People typically show more prosocial behaviours towards ingroup members (Turner et al., 1979) and increased negative behaviours towards outgroup members (Brewer, 2017).



**2.3.2.1 Social Identity in Software Engineering.** Software is not developed by individuals in isolation. It is developed in rich social cultures that influence software production (Sharp et al., 2000), requiring extensive communication (Lopez et al., 2020a). Social identities are present within software engineering communities (Rauf et al., 2020), where the identity of “software developer” is distinct even within an organisation (Backevik et al., 2019). Where social identities are strong enough to form self-perceived distinctive groupings, this can influence security behaviours, such as where developers avoid asking security experts for advice (due to perceptions of unshared identities), as they see themselves as wasting the expert’s time and instead seek information from peers (Nicholson et al., 2018). This behaviour can be detrimental, as developers may receive inaccurate information based on perceived social groups and status.

Organisational security cultures are based upon the security-specific values expected to be internalised by employees, and they are essential for the maintenance of security practices (Haney et al., 2018), even in non-security industries (Poller et al., 2016). Where security-positivity is seen as an attribute of a social group, it is perceived as a shared responsibility distributed among group members (Assal & Chiasson, 2018b). An increased security focus can lead to it being assumed a shared value amongst ingroup members (Lopez et al., 2019a), supporting the security culture. Social and institutional structures promote accountability, where responsibility for users heightens a developer’s security sensitivity (Rauf et al., 2020).

In online communities, social interactions can increase membership strength (Mustafa et al., 2023), which can lead to the communities being trusted similarly to in-person interactions for security information (Xiao et al., 2014). On sites such as StackOverflow, the conversations and forum posts promote these relationships (Lopez et al., 2019c). When considered alongside the findings that StackOverflow solutions are frequently insecure (van der Linden et al., 2020b), it is possible this trust is misplaced. Similarly, in software security tasks that involve others (such as seeking assistance or advice), developers some-

times assume others are providing correct information, and fail to critically reflect on the dangers of just trusting others (van der Linden et al., 2020a), particularly where their identities are unknown.

Software ecosystems are large distributed systems of software united by a common technology (e.g., iOS ecosystem). These ecosystems unite diverse populations, including users and developers who may otherwise have few shared values (Manikas & Hansen, 2013). The underlying technology provides a common value across ecosystem users, affording a shared social identity. These social networks promote increased ecosystem engagement, driven by the ecosystem-based identity (de Souza et al., 2016). These shared values encourage ingroup affiliation, and the presence of shared values can improve performance across development tasks where a common goal is identified (Teh et al., 2012).

If software engineers perceive no commonality between themselves and their users, then user values may not be considered during development. Human values, such as diversity, integrity, and responsibility, are poorly represented within software engineering (Whittle et al., 2021), which results in unethical software, such as biased algorithms (Galhotra et al., 2017). When implementing software, there is a limited understanding of the software used within the context of society resulting in values not being integrated (Winter et al., 2019). Where values are ‘breached’, it can lead to user dissatisfaction or abandonment of software (Mougouei et al., 2018), causing issues for developers.

Group influences can affect secure behaviours. When making security decisions in a cybersecurity context, decision making is influenced by the multiple perspectives available, broadening the decision making space (Shreeve et al., 2020). Interactions between individuals can be pivotal in ensuring these perspectives and understandings are shared, resulting in more optimal decisions being made, even for those without cybersecurity expertise (Shreeve et al., 2022). Group interaction can reduce biases, with developers offering more realistic estimates than when independently asked (Moløkken-Østvold &

Jørgensen, 2004).

## 2.4 Soft Skills

Soft skills are domain-agnostic, and are valued across industries, employers, and job roles. Soft skills influence how effectively technical skills can be applied. They are the observable manifestations of latent psychological attributes that complement technical skills (Ahmed et al., 2015). It is helpful to consult research on soft skills as it offers an insight into how psychological dimensions are valued in software engineering through familiar terminology that many practising developers will be familiar with (e.g., problem solving). Employers who consider technical skills comparatively easier to teach (Liebenberg et al., 2014) view soft skills positively. They are typically independent of company size or organisation type (Galster et al., 2022), suggesting that they are not developed in response to specific organisational structures. One issue with soft skills is that they are typically poorly defined, so their definitions are inconsistent across software engineering research. This issue is addressed in Chapter 4.

Within software engineering, soft skills are recognised as a significant distinction between average and exceptional engineers (Capretz & Ahmed, 2018). SWEBOK (Software Engineering Body of Knowledge; Bourque & Fairly, 2014) represents the skills needed for software engineering employment and recognises cognitive, social, and professional skills. Soft skills are used throughout a career and are not limited to software-specific activities. When acclimatising to a new workplace, employees can get overwhelmed by the need to understand the social infrastructure, but soft skills ease transitions (Rabelo et al., 2022), improve team integration, and enhance team productivity (Stevens & Norman, 2016).

Interpersonal skills of communication and collaboration are in high demand. Communication is required for most software roles, including design, programming, and testing (Ahmed et al., 2012b), highlighting the social nature of software engineering. Social interactions are commonly requested in job adverts (Montandon et al., 2021). Cognitive-based skills, such as problem solving are considered critical for effective software development

(Matturro et al., 2019; Mtsweni et al., 2016), as it is a skill closely tied to programming (Ahmed et al., 2012b). Problem solving and critical thinking allow real-world issues to be translated into computational problems to be addressed through software solutions.

Software engineers are also expected to manage their environment, including time management, self-management, and conflict resolution (Matturro et al., 2019). These require consistency in their practice to be considered a soft skill, as they are only beneficial with their continued applications. The ability to move effectively between types of work relates to environment management, and task switching is valuable, mainly when it involves switching between autonomous work and group work (Ahmed et al., 2012a).

Security-focused domains may require different skills than what is needed in typical development, and soft skills should also be valuable within these roles (Furnell & Bishop, 2020). Like general software engineering, communication is crucial (Armstrong et al., 2020), as are interpersonal skills, adaptability, and innovation (Haney & Lutters, 2017). Critical thinking, decision making and problem solving are also highlighted (Sussman, 2021), as well as organisational skills as a requirement for creating effective knowledge bases (Graham & Lu, 2022). Like SWEBOK, CyBOK (Cybersecurity Body of Knowledge; Rashid et al., 2021) highlights the importance of these skills.

It is essential to understand where soft skills are developed and how they are effectively taught to aspiring developers. Employers expect graduates to possess the necessary career skills, including the non-technical (Andersson & Logofatu, 2018). However, a gap between education and employment is reported (Akdur, 2021), resulting in employers perceiving graduates to be seeking employment without the necessary soft skills (Liebenberg et al., 2014).

Delivering soft skills in higher education is essential because, as with students who start their degree with diverse levels of technical skills, it is the same for soft skill competencies. It is subsequently essential to ensure higher education provides the necessary skills

required for software engineering (Balaji & Somashekar, 2009). Exposure to industry-relevant practical experiences can help students recognise the value of soft skills by integrating them into software project coursework (González-Morales et al., 2011). Ensuring that soft skills are embedded in modules is necessary to allow students to constantly use these skills (Hazzan & Har-Shai, 2013). One way of incorporating soft skills is through project-based learning (Zheng et al., 2015). This learning method typically requires continued engagement over an extended period, encouraging the development of soft skills like time and meeting management (Carter, 2011).

## 2.5 Other Psychological Perspectives

Cognitive and social psychological approaches are not the only approaches applicable within software engineering. In this section, I cover two other perspectives that *could* have been taken in the development and synthesis of this thesis, as well as the reasons for why they were not used in favour of more psychological approaches.

One approach is organisational psychology, which focuses on the behaviours and actions that occur within the workplace and other organisational settings (Kraiger, 2001). Organisational psychology has made significant contributions to understanding software engineering and security practices at the team and institutional levels, from motivation (Lenberg et al., 2015), leadership (Haney et al., 2018), organisational culture (Bada et al., 2019), and policy compliance (Parsons et al., 2014). All these factors influence, to some degree, the security position of both organisations and the teams within. For example, leadership advocating for security awareness can filter down and influence how employees interact and think about security (Haney et al., 2018).

Another potential approach is Human-Computer Interaction (HCI), which focuses on the processes through which people interact and engage with computer systems. HCI has played a role in the evolution of *usable* security and secure software engineering. HCI research has demonstrated how users and developers interact with security features (e.g. secure APIs) and how interface design affects secure behaviour (Cranor, 2005; Green &

Smith, 2016). Beyond usability, experience-focused approaches in HCI have drawn attention to how affect, trust, and meaning-making shape security engagement, offering richer accounts of how individuals interpret and respond to security cues in practice (Dourish, 2001).

Despite the value of both organisational psychology and HCI as distinct and independent research fields, neither is specifically oriented toward capturing the more granular cognitive and behavioural processes that underpin an individual's decision-making in security-critical contexts. Organisational psychology contributes insights into structural, cultural, and interpersonal influences on security behaviour across teams and institutions; however, the focus on the organisational group dynamics may overlook the individual and their cognitive and motivational nuances in favour of the wider organisation (Schneier, 2008). Similarly, HCI provides an understanding of usability and interaction, but typically emphasises external interfaces over internal human processes like decision-making or cognitive bias (Camp, 2009). Though both disciplines have strong explanatory power within their respective domains, social and cognitive psychology are potentially better suited to analysing how individuals navigate uncertainty, social influence, and complex decision-making.

It is important to acknowledge that no single discipline can fully account for the complex nature of secure software engineering. Organisational psychology and HCI offer essential perspectives, particularly in institutional, and long-term system contexts. Nonetheless, this thesis draws a deliberate and discrete boundary to focus on the psychological mechanisms closest to individual behaviour. This approach reflects a methodological and conceptual alignment with the thesis aim: whether theories of decision making and social identity can explain security behaviours within software engineering. Social and cognitive psychology provide robust frameworks for modelling cognitive- and socially-mediated decision-making.

It is also worth recognising that the supervision team that oversaw the PhD operate within the intersection of software engineering, social psychology, and cognitive psychology. Much of this work forms a large portion of the existing application of psychology into software engineering and particularly ideas of social identity (Rauf et al., 2020, 2021). As a result, in the nascent stages of the PhD project, it is only natural that the team’s existing work would influence the direction the project would take in terms of theoretical approach.

## 2.6 Summary

Software security was reviewed concerning the software and the artifacts available to software developers. It was followed by social and cognitive psychology applications in software security and soft skills research and their relevance in software engineering and CS contexts, before discussing two other relevant methods for interpreting secure software behaviour. In exploring these topics and revisiting the bibliometric analysis, what is observed is that psychology, particularly theories of social identity and decision making, is mostly absent in software security research.

Dual processing theory has been touched upon in its theoretical potential but has yet to be empirically applied to secure software engineering behaviours (Petre, 2022). Previous research implicitly refers to aspects of the theory, such as bias research, and priming interventions. However, it does not recognise the individual differences in the propensity or engagement of more reflective thinking styles. In measuring these predispositions to engage in more rational system 2 processing and applying them to secure software engineering domains, a better understanding of cognition should be achieved.

The influence of social identities within secure software development still needs to be better understood. Rauf et al. (2022) report that developers perceive security differently, but it is necessary to extend this to understand how these differences manifest. By identifying social identities around secure coding, we can better understand how secure behaviours occur because of social interactions.

As noted previously, soft skills are ephemeral and are loosely tied to more tangible psychological measures. In identifying the valuable soft skills and translating these to psychological theories, these can be applied in further research with more meaningful impact.

## **2.7 Methodological Motivations**

This thesis comprises a series of related empirical projects and collectively utilises a mixed-methods approach. Research chapters, oriented towards publication venues, are necessarily “light” on methodological background, as they focus more on the narrative of conclusions than justification of evidence, as is typically expected. Accordingly, the methods and analyses are reviewed here for a more complete narrative and reader comprehension. I will cover the research designs implemented and the analytical approaches taken, as this thesis used both quantitative and qualitative analyses. I will first highlight the methodological commitments and rigour, before I review the benefits, limitations, and relevance of the methods before covering the open science approaches.

### ***2.7.1 Overarching Methodological Commitments***

The methodological approach taken in this thesis is guided by an overarching commitment to socio-cognitive theory, supported by constructionist and experiential realist epistemologies within a pragmatic mixed-methods framework. The central research aim, to understand the psychological dimensions of computing and secure behaviours, necessitated approaches that could both explore subjective experience and model generalisable constructs across contexts.

***2.7.1.1 Social and Cognitive Foundations.*** The conceptual framing of the thesis is underpinned by socio-cognitive theory. From this perspective, individual cognition is viewed as fundamentally shaped by social contexts, particularly language and interactions. This perspective guided not only the choice of research questions but also the interpretation of both qualitative and quantitative data. Where quantitative work sought to model latent psychological constructs (e.g., cognitive styles), qualitative studies were designed to understand how these constructs are socially framed and experienced contextually by individuals. In line with cognitive and social approaches, it is assumed that cognitive



structures could be both measured quantitatively and contextualised qualitatively. These assumptions justified the use of both quantitative modelling and in-depth thematic exploration as complementary modes of inquiry.

**2.7.1.2 *Epistemological Commitments.*** This thesis adopts a pluralist epistemological stance, using constructionism for qualitative inquiry and experiential realism for quantitative analyses.

- Constructionist epistemology guided qualitative components, recognising that individuals interpret their experiences through socially and culturally shaped lenses. The thematic analyses therefore focus on how participants construct meaning.
- Experiential realist epistemology supported the quantitative studies, assuming underlying constructs (e.g., decision-making styles, or behaviours) can be measured with sufficient reliability and validity. This moderate realist stance was taken (rather than a stricter realism stance), which acknowledges that measurement is always partial and theory-laden.

This dual epistemological stance reflects a pragmatic approach to methodology, which views the research question as determining the method, rather than the other way around. This flexibility allowed the thesis to match research tools to the specific aims of each study, while maintaining coherence across the project.

**2.7.1.3 *Methodological Integration Across Studies.*** The thesis is structured in two broad phases, each containing both qualitative and quantitative components:

- Phase 1 (Chapters 4 and 5) explores stakeholder perceptions of soft skills, using surveys (Chapter 4) and semi-structured interviews (Chapter 5). These exploratory studies seek to map out how soft skills are socially constructed and understood within computing communities.
- Phase 2 (Chapters 6, 7, and 8) builds on the insights from Phase 1 to investigate

the psychological dimensions of software engineering through observational and experimental paradigms. These include more formal hypothesis testing using psychological tests and behavioural models.

Method choices in each study were directly informed by both the *theoretical framing* and the findings of prior studies. For example, thematic analysis was selected in Chapter 5 over other methods, such as discourse analysis, due to its flexibility (Braun & Clarke, 2006) and compatibility with a socio-cognitive framing that focuses on shared meanings rather than solely individual experience, enabling generalisations to be made. Similarly, the quantitative work models psychological traits as latent variables, assuming they have predictive utility but are still interpreted in relation to broader social patterns.

By integrating findings across both paradigms, the thesis reflects a coherent commitment to understanding how individual cognition and social interaction jointly shape soft skill development and deployment in software engineering contexts.

### ***2.7.2 Research Designs***

The research designs deployed are diverse, with surveys, interviews, and experimental paradigms being the primary methods used in this thesis.

***2.7.2.1 Surveys.*** Chapter 4 used a survey study on soft skill perceptions with CS and Psychology graduates. Survey studies are designed to extract information from large populations easily (Jones et al., 2013). Through careful design and development, once surveys are deployed, they require minimal researcher involvement. Due to their ability to canvas large samples, surveys explore human perceptions and behaviours (Singleton & Straits, 1988). The data can be aggregated for the identification of latent patterns and ideas. Survey studies have three core limitations: a lack of researcher control, response bias, and attrition.

Once distributed, researchers have little control over how participants impact data quality and validity (Theofanidis & Fountouki, 2018). By focusing efforts during the planning

stages, the survey design can be evaluated for its ability to collect the right information and how it answers research questions. Preregistering survey materials can motivate researchers to ensure they have carefully considered the materials before data collection.

Not all participants respond to survey invites, requiring a large sample to be canvassed to ensure enough data is collected. Those who respond are a self-selecting sample which influences the results (Bethlehem, 2010). Controlling for this is not simple, as it is difficult to encourage participation from those unwilling even with incentives. This limitation is typically addressed by accounting for it when considering the implications, and being aware that identified effects may be inflated as a result.

Attrition occurs when participants do not finish a survey for a variety of reasons, including survey flow issues or frustration. A balance lies in developing surveys to ensure all information is collected and minimising attrition rates. The smallest amount of information necessary should be collected, as too many questions can result in disengagement. Starting with the most essential sections can also improve the collection of usable information. Attrition is simply an expected part of survey studies, but the planning stages are essential to avoid participant frustration.

**2.7.2.2 Interviews.** In Chapter 5, a semi-structured interview design was deployed to understand staff perceptions regarding embedded soft skills. Interviews collect rich information on topics where personal experiences are valued (Taherdoost, 2022). Semi-structured interviews use pre-determined open-ended questions to guide the interview with the flexibility to explore other questions that arise from the interview itself. A good research interview is dominated by the interviewee but is kept relevant and within the bounds of the research question by the interviewer (Slembrouck, 2015). A poor interview environment may lead to interviewees not feeling in control of the interaction, resulting in lower-quality responses as they become less narratively inclined. Questions should be carefully considered, with the first question regarded as the most important. If the first question is open-ended and targets simple ideas that engage a narrative mindset,

participants may be more willing to offer more information, which has the benefit of placing the interviewee in charge of the interaction.

Interviews provide rich details that surveys cannot achieve. The time commitment from the researcher is much higher than in survey studies, but it gathers more in-depth information. Interviews complement surveys or experimental research as they offer valuable information otherwise unobtainable. The use of semi-structured interviews aligns with the thesis's constructionist epistemology and socio-cognitive framing. It allows participants to articulate socially-situated views, offering insight into how these concepts are shaped by culture and interpersonal dynamics.

**2.7.2.3 *Observational and Empirical Studies.*** Observational and empirical studies draw associations between two or more variables, where the intention is often to understand how functions of psychology may influence real-world behaviours. Using relevant psychological measures to capture aspects of cognition alongside software engineering tasks illuminates and offers valuable insights into whether effects exist that are worth pursuing in further research. The observational studies were informed by a socio-cognitive framing that views behaviour as the result of interaction between internal (cognitive) and external (social/environmental) influences. This framing supports the use of observational methods to capture naturally occurring behaviours in context.

Core limitations are confounds and the observational nature of the design. Unanticipated confounding variables can have unseen influences on the data, resulting in inappropriate implications being drawn. This issue is addressed during the planning stages, and possible confounds are identified by leveraging previous research. The second issue, which speaks to the design itself, is that the paradigm can only *observe* variable association and does not manipulate variables to determine causal patterns. Observational designs are best deployed in research contexts where an increased theoretical understanding is desired before experimental manipulation is used to determine cause and effect.

### 2.7.3 *Analytical Methods*

This thesis uses a mixed-methods approach to address the complexity of deploying psychology in software engineering contexts. Chapters utilise different analytical methods to seek answers to the research questions posed. Through this diverse application of methods, the thesis arrives at a set of implications based on the triangulation of methods.

**2.7.3.1 *Qualitative Approach.*** Qualitative data was implemented using thematic analyses, a method of analysing rich text to identify meaningful patterns that can be grouped into latent themes. These themes present as recurring ideas identified across documents, speaking to a common experience (Riger & Sigurvinsdottir, 2016). A core assumption is that the data contains people’s perceptions of their own reality, and through analysis, it is possible to make sense of their experiences (Banyard & Miller, 1998). The use of thematic analysis fits with the socio-cognitive framing of the thesis, as it enables the identification of recurring meaning patterns that emerge through shared language and social experience.

I approach qualitative research (specifically thematic analyses) from a constructionist perspective, aiming to understand a person’s personal reality instead of providing statistical support for a given hypothesis (Charmaz, 2008). A constructionist approach recognises that people experience reality differently, and it is these differences which are valuable. Qualitative research offers the ability to explore complex behaviours that do not translate easily to quantifiable scenarios.

Thematic analyses are carried out in five stages: familiarisation, code generation, theme generation, theme review, and theme naming (Braun & Clarke, 2006).

1. Familiarisation: it is important that researchers immerse themselves in the data through repeated readings. This process allows possible patterns to be identified.
2. Code generation: initial codes are generated across documents, and I use a data-driven approach allowing codes to be developed from the data (Gibbs, 2007). Codes

are the early organisation of data into units of meaning that by themselves are not themes, as multiple codes may speak to broader concepts (Riger & Sigurvinsdottir, 2016). At this stage, coding all information that potentially indicates a pattern allows for an unrestricted analysis in later stages (Braun & Clarke, 2006).

3. Theme generation: themes are generated through combining and updating codes which is why exhaustive code generation is essential. Code clustering and patterns help create early themes. A hierarchical approach is used to organise initial themes under umbrella terms to group related themes.
4. Theme review: the suitability of themes is considered, as not all themes are relevant to research questions. Theme distinction is important, and code coherence is critical. It is beneficial to review themes and code coherency multiple times.
5. Theme Naming: themes need names and definitions. The central idea attached to each theme is identified and named. Themes may be implicit or explicit and should capture meaningful information related to research questions (Riger & Sigurvinsdottir, 2016). Themes must be rationalised and presented beyond just a description (Braun & Clarke, 2006). If themes are not rationalised, they can be easily dismissed in favour of a reader's preferred rationale.

The major criticism of thematic analysis, or qualitative analysis in general, tends to emphasise its subjectivity and that the reliability or validity of the work cannot be guaranteed. This absence of guarantee speaks to the broader issue of reliability and validity. Reliability relates to whether the findings would be obtained in repeated samples, and validity describes the relationship between the findings and reality (Leung, 2015). Reliability and validity are issues typically presented from a quantitative perspective, where data should fit into an idea of reality that can be measured and quantified repeatedly. A constructionist approach emphasises that an individual's perceived experience is not representative of a global reality (Charmaz, 2008), so using principles from quantitative

approaches to assess qualitative research does not make sense. Instead of using the same metrics for quantitative work, qualitative work should be assessed against four criteria: credibility, transferability, dependability, and confirmability (Lincoln & Guba, 1985).

Credibility is whether the participant would find the results believable, as the aim is to describe *their* worldview. Confirmability shows that others can corroborate findings and are minimally influenced by researcher bias. To achieve credibility and confirmability, having those not involved in the analysis review results can be beneficial in gaining agreement towards the themes identified. Keeping an audit trail of the work carried out, how the initial codes became themes and how these were combined, split, or removed also helps. Transferability relates to how far the findings can be applied to other contexts. Dependability is a measure of the finding's consistency and repeatability. Transferability and dependability can be achieved by providing detailed descriptions of the data collection process, allowing others to replicate the approach elsewhere.

The researcher's personal experiences cannot be removed from the analysis. This is addressed through reflexivity, the critical reflection that considers the researcher's impact on reducing unseen influences (Riger & Sigurvinsdottir, 2016). Reflecting on methodological choices and research assumptions can reduce their impact through recognition. The aim of reflexivity is not to remove bias but to be transparent about how it may affect findings (Johnson & Waterfield, 2004). Research questions are also constructed to reduce potential biasing by keeping them open and without assumed direction.

**2.7.3.2 Quantitative Approach.** Quantitative analyses are presented from an experiential realist perspective, assuming a “real” world exists independently outside of our perceptions (Riger & Sigurvinsdottir, 2016). Quantitative analysis requires information to be numerically represented, which can be done by developing and validating scales and measures that seek to collect this information. Item response theory asserts that many scales and measures are intended to determine individual differences upon a latent construct (Thissen & Steinberg, 2009). The collection of items (questions) within a scale is

assumed to measure the same latent trait, and the scores derived from a scale refer to an individual's strength of the trait. Data can also be considered suitable for factor analyses, which share similarities with item response theory but focuses on the covariance between items to identify latent groupings of multiple items (MacCallum, 2009).

Correspondence analyses and multilevel models are detailed in later chapters and so are given no space here. Quantitative measures are used to measure latent traits numerically so that hypotheses can be tested through statistical means. In comparing the differences between groups or the relationships between a real-world action or behaviour (such as software security awareness), the ability to carry out these behaviours is quantified in terms of variance explained by the latent traits. These analyses give us an understanding of what influences these real-world behaviours.

Key issues with quantitative approaches primarily revolve around the reductionist approach of representing complex human behaviour as numerical values, as well as the assumptions of the realist approach. Realism assumes the researcher can detach themselves from their perceptions and worldview to identify the true reality correctly (Sukamolson, 2007). This issue is assuaged by taking a less absolute approach through experiential realism, which recognises that the perceptions we hold influence our views, but this subjectivity is limited. Reducing complex behaviour into quantities can result in basic snapshots of the measured behaviour, which is a superficial representation of the actual reality (Rahman, 2017).

**2.7.3.3 *Mixed-method Approach.*** On a surface inspection, combining quantitative and qualitative methods can seem incongruent, but it is possible to combine these effectively (Sukamolson, 2007). Constructionist approaches can be used where personal experiences are valuable for recognising motivations, which can then be translated into a realist approach regarding the shared experiences identified. This pragmatic approach recognises that a mixed method is needed where certain lines of scientific inquiry are best answered through different perspectives. The use of mixed methods in this thesis reflects



a socio-cognitive and pragmatic orientation. By combining methods that explore subjective, socially situated understandings with those that model individual cognitive traits, the research provides a layered account of soft skill development across contexts.

Mixed-method approaches enrich the research by achieving aims that cannot be suitably achieved by a single approach alone (Kelle, 2006). Each approach has limitations, but individual weaknesses are reduced when used together. Quantitative approaches reduce complex behaviours to numeric representations to gain statistical testing. However, it loses the data richness, and qualitative methods maintain data richness at the expense of statistical testing, complementing each other. This triangulation of different perspectives offers a more accurate representation of the event (Scott, 2007b).

In areas where previous research is limited, qualitative methods are effective for “casting wide” to understand how the landscape is understood broadly. In this thesis, this primarily refers to phase 1, where I explore the graduate and staff perceptions of soft skills. These ideas are then translated into their latent psychological dimensions that are investigated through further studies in phase 2. Both phases use both qualitative and quantitative methods.

## **2.8 Open Science**

This thesis enacts open science practices throughout, with preregistrations, material sharing, preprints, and open access. They represent a significant posture of the research programme that ultimately contributes to more robust and transparent research.

### ***2.8.1 Preregistrations***

Preregistrations are a method for formally registering the intended design, methodology, and analysis for a research project. They provide information about the intended questions and hypotheses that the project aims to address, as well as offering opportunities to justify research decisions prior to data collection (Nosek et al., 2018). Preregistrations improve research credibility and show foresight (Nosek et al., 2019) and provide clarity

about which analyses reported were selected a priori and what were post-hoc, reducing dubious research practices.

### ***2.8.2 Data Sharing***

Publicly sharing data allows others to validate published results and offer other researchers opportunities to explore and re-analyse data further (Towse et al., 2021b). Not only does it benefit others in this regard, but it offers support for the thesis itself, demonstrating that all the findings from this project can be checked, reproduced, and validated by interested parties. In sharing the associated data and analysis files, I follow the FAIR principles, ensuring data is Findable, Accessible, Complete, and Well-described (Towse et al., 2021a).

To ensure data is findable, Digital Object Identifiers provide permanent links, avoiding “link rot” – where URLs no longer exist. All data and materials are hosted on the Open Science Framework, which is searchable and links between manuscripts, preprints, and any significant research output.

Accessibility ensures that data is useable by the largest audience. Proprietary data formats restrict usage, so all data is stored in universally-readable formats, increasing access. It is also essential to ensure that data files are accurate and defect-free. In this thesis, the data files shared are the same files used during analyses and are not simply produced to satisfy basic data sharing.

To achieve completeness and to ensure sharing practices are meaningful, all essential data must be shared (Towse et al., 2021a). All datasets required to replicate the thesis findings are openly available and found through <https://osf.io/2fpbq/>. Providing analysis scripts also clarifies the data manipulation process, affording a more straightforward interpretation of the analysis.

Impactful data should also be well-described. Additional files that describe the analysis pipeline and the relevance of specific files can increase usability. Including commented

scripts for clarity and stating where steps were taken outside of computational environments (such as manual data tagging). Where this occurs, this should be made explicit so that the reader is aware of the whole process.

It is not just data files and analysis scripts that are shared. Survey designs, interview questions and other relevant, shareable artifacts are uploaded to open access repositories to facilitate future research. By providing these materials, the research is more easily replicated or improved upon by understanding the rationale and the research designs.

### ***2.8.3 Computing Environments***

Throughout the PhD journey, best efforts have been made to utilise open-source software. The quantitative analyses and data manipulation were all carried out predominantly in R, and qualitative approaches, such as the thematic analyses, were carried out using the open-source tool Taguette (Rampin & Rampin, 2021). To this end, the thesis document itself is fully reproducible.

### ***2.8.4 Preprints***

Preprints are openly available manuscripts published on dedicated servers (e.g. PsyArXiv, ArXiv) before the peer review process. They offer rapid means of dissemination, as well as greater exposure for early career researchers (Sarabipour et al., 2019). Indeed, Chapter 7's preprint received citations while the paper was undergoing peer review, providing an early contribution to further research. Throughout the PhD process, when papers were submitted for peer review, they have also been uploaded to preprint servers along with data sharing.

### ***2.8.5 Open access***

Finally, each research chapter has been published under open access (OA) agreements that are either Gold or Green publication models. Gold OA is where the final published version is openly available from the journal/publisher, such as in the case of Chapter 4, which is published in the APA Open journal Technology, Mind & Behavior, having to pass rigorous checks for publication in a Gold OA journal. Green OA is the case for all

other research chapters, as they are uploaded to preprint servers.

### ***2.8.6 Summary***

This thesis represents a programme of work that is open in as many ways as possible, from preregistering research designs and motivations, ensuring analyses were conducted in a FAIR way, publishing data and materials, to preprinting manuscripts and open publishing. Many of these aspects are often concealed, and the output does not necessarily reflect the increase in workload associated with a research project. It is more complex than just a journal submission; one must ensure that the data and materials are anonymised, shareable, and published simultaneously with manuscript submission. These practices reflect an invisible but significant aspect of the research process.

### **3 The Soft Skills of Software Learning Development: The Psychological Dimensions of Computing and Security Behaviours**

Ivory, M. (2022). The Soft Skills of Software Learning Development: The Psychological Dimensions of Computing and Security Behaviours. *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering 2022*, pp. 317–322. <https://doi.org/10.1145/3530019.3535344>



# The Soft Skills of Software Learning Development: the Psychological Dimensions of Computing and Security Behaviours

Matthew R Ivory

Supervised by Prof. J Towse, Prof. M Levine, Dr. M Sturdee, & Prof. B Nuseibeh

Lancaster University

Lancaster, UK

matthew.ivory@lancaster.ac.uk

## ABSTRACT

When writing software code, developers typically prioritise functionality over security, either consciously or unconsciously through biases and heuristics. This is often attributed to tangible pressures such as client requirements, but little is understood about the psychological dimensions affecting security behaviours. There is an increasing demand for understanding how psychological skills affect secure software development and to understand how these skills themselves are developed during the learning process.

This doctoral research explores this research space, with aims to identify important workplace-based skills for software developers; to identify and empirically investigate the soft skills behind these workplace skills in order to understand how soft skills can influence security behaviours; and, to identify ways to introduce and teach soft skills to computer science students to prepare the future generation of software developers.

The motivations behind this research are presented alongside the work plan. Three distinct phases are introduced, along with planned analyses. Phase one is currently in the data collection stage, with the second phase in planning. Prior relevant work is highlighted, and the paper concludes with a presentation of preliminary results and the planned next steps.

## CCS CONCEPTS

• Security and privacy → Human and societal aspects of security and privacy; • Software and its engineering; • Human-centered computing → Human computer interaction (HCI);

## KEYWORDS

Soft Skills, Cognitive Psychology, Security, Behavioural

### ACM Reference Format:

Matthew R Ivory. 2022. The Soft Skills of Software Learning Development: the Psychological Dimensions of Computing and Security Behaviours. In

*The International Conference on Evaluation and Assessment in Software Engineering 2022 (EASE 2022), June 13–15, 2022, Gothenburg, Sweden.* ACM, New York, NY, USA, 6 pages. <https://doi.org/10.1145/3530019.3535344>

## 1 INTRODUCTION

In this research, soft skills are defined as the psychological dimensions, or traits, that underpin behaviour [5]. Commonly, soft skills are synonymous with workplace-relevant transferable skills, including skills such as "teamwork" or "time management", and have been the focus of previous human factors research [10, 14, 15]. This current research seeks to go beyond these surface level traits, to identify the psychological dimensions that underpin transferable skills. In this body of research, transferable skills are referred to as "shallow skills", and soft skills are the underlying psychological dimensions. Shallow skills are referred to as such, because they provide little in the way of quantifiable skills, and their definitions often change depending on research context. Shallow skills can be considered as the manifestation of soft skills, particularly in workplace situations. Soft skills are the more immutable, psychological aspects of behaviour.

Software development is the direct product of human interaction, created through the combination of cognitive abilities, social interactions and the unique culture of software development [3, 23]. In recent years, the software industry has become aware of the significance of soft skills for successful software creation [5, 14, 15]. By 2030, there is an anticipated 22% increase in employment opportunities for software developers compared to an average 8% increase across all other industries<sup>1</sup>, but a rising concern that graduates entering the workforce are lacking the necessary cognitive and social skills required for successful integration into the workplace [13]. This issue has been evidenced in software security roles, with research indicating the most important skills required for security roles are not technical in nature, but are soft skills [8]. As a consequence, it is vital to identify the psychological traits required to successfully develop secure software.

Security in software is not a new concern, but the responsibility for security has changed over time. In 1999, Adams and Sasse [2] argued that software users were "not the enemy" and their fallible security behaviours were not their fault, but rather that of developers disregarding default user behaviour.

Similarly in 2008, Wurster and van Oorschot [24] posited that developers were "the enemy" and as they are the ones causing security issues, security should be removed from their responsibilities.

<sup>1</sup><https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

EASE 2022, June 13–15, 2022, Gothenburg, Sweden

© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-9613-4/22/06...\$15.00

<https://doi.org/10.1145/3530019.3535344>

They suggested the onus should be placed with API developers as they provide functionality (and security) to other developers. More recently, this sentiment about API developers was echoed by Green and Smith [9], who emphasised that developers typically focus on functionality and expect APIs to be secure by default. One issue with this argument is that it treats software developers as a homogeneous population with little security awareness, but expects API developers to be somehow more security conscious. API developers are as human as software developers, subsequently they are susceptible to the same cognitive and social biases [4, 16]. Rather than assigning responsibility to different groups, we should identify the psychological dimensions associated with good security behaviours and seek to promote these skills in software learning development.

### 1.1 Motivation and Rationale

The primary motivation is to understand how soft skills relate to software development, and how people’s skills develop and exhibit in software development environments. Of particular interest are the behavioural changes exhibited by relative novices during skill development, compared to more experienced developers. What potentially incorrect, but intuitive actions are ultimately suppressed through experience? What habits are built and how do these originate? What soft skills are required for secure software development and how do these skills manifest and evolve?

In recent years, increased attention has turned towards the psychology of software developers, particularly in relation to security [17]. Security vulnerabilities typically leverage psychological processes [21], via cognitive processes (such as exploiting expected use cases), or through exploiting heuristic use. If adversaries exploit developers’ behaviour, it is important to identify the soft skills involved and find ways in which these behaviours can be changed through psychological interventions, which can be taught to novice and experienced developers alike.

The project is motivated to provide practical impact through developing teaching materials for Computer Science courses. Incorporating psychological interventions into pedagogy will allow for the development of soft skills and security conscious behaviours in future generations of software developers.

### 1.2 Contribution

The main aim of this research is to understand how the software learning development process occurs and how behaviours change and evolve. By identifying these processes, we are better placed to encourage positive behavioural changes, resulting in more efficient code development. The project also seeks to investigate key soft skills that affect secure code production. As a result, psychological interventions can be developed for promoting better security practices. To practically encourage relevant behavioural changes in early-stage software developers (e.g. Computer Science students), pedagogical materials will be developed for education, with the aim for these to be incorporated into teaching practices.

## 2 RESEARCH QUESTIONS

RQ1: What soft skills are considered important for computing and security practices?

RQ2: How do soft skills evolve and develop in novice software developers with time and experience?

RQ3: How do software development behaviours evolve and change with experience?

RQ4: How can relevant soft skills be incorporated into pedagogical practice to promote security behaviours?

## 3 WORK PLAN

The research incorporates a breadth of analysis methods, including qualitative and quantitative approaches. This methodological pluralism allows for a broad range of information to be drawn from the data that would otherwise not be possible with a restricted methodology. The doctoral research will look at data collected through interviews, surveys, data scraping and behavioural studies. Analysis will be varied and include statistical modelling, natural language processing modelling, and qualitative approaches, such as thematic analysis. Not only will it provide broader interpretation to findings within this space, it allows for stronger links to other work in similar research spaces.

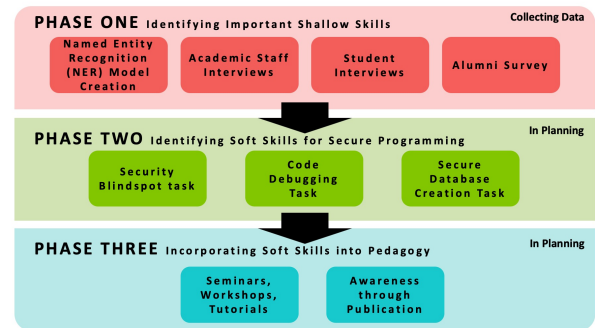


Figure 1: Outline of research phases within the planned work.

The research can be split into three phases, as illustrated by Figure 1. The first phase identifies shallow skills as taught in Computer Science undergraduate courses. It seeks to understand how shallow skills are perceived by current students, staff and alumni.

The second phase will build directly on the first phase. By identifying soft skills linked to shallow skills through previous research, phase two aims to build relationships between secure coding behaviours and soft skills. This will be achieved through empirical, lab-based research involving manipulation and measuring of soft skills and programming tasks.

Finally, the third phase will focus on incorporating findings on soft skills into pedagogical materials. This phase will measure the effectiveness of teaching these ideas, with the aim to raise awareness and increase the understanding of the psychological traits of software developers.

A stand-alone study is also being conducted into cognitive reflection and risk perception in software developers and computer scientists, see section 3.4. This will fit in with the phase two work.

Following open science practices, the research will include pre-registrations, data sharing and reproducible analysis scripts. This

will be managed through the use of the Open Science Framework<sup>2</sup> and provision of Docker containers with reproducible workflows.

### 3.1 Phase One: Identification of Shallow Skills

The first phase is currently in the data collection stage. This phase is comprised of four research projects: an examination of core modules in computer science programmes as taken from university websites; academic staff interviews on how they view shallow skills being taught; longitudinal interviews with current students on how they develop their shallow skills over an academic year; and an alumni survey of computer science graduates and psychology graduates, collecting data on their perceived importance of shallow skills.

**3.1.1 Curriculum Examination.** The online course information and core module descriptions for Computer Science and Psychology undergraduate courses were collected from eight UK universities belonging to the N8 research group (Durham, Lancaster, Leeds, Liverpool, Manchester, Newcastle, Sheffield and York).

To identify shallow skills in natural language texts, a named entity recognition (NER) model will be developed. Similar work has been created [6], but without the granularity attempted here. In efforts to further understand covariance of shallow skills and language used around them, the NER model weights can be analysed further, including factor analysis to find highly correlated skills. A preregistration, providing details on the data collection can be found at <https://osf.io/qcw3n>.

**3.1.2 Alumni Survey.** Lancaster University undergraduate alumni from Computer Science and Psychology were contacted to take part in a survey. Participants were asked to rank shallow skills for their importance in current employment. A Psychology sample were used as a comparative group, particularly when considering the less vocational nature of psychology undergraduate degrees (inferred from software development roles attained with a minimum education of a bachelor's degree<sup>3</sup>, compared to a minimum education of a postgraduate degree for most psychology roles<sup>4</sup>).

Data analysis will focus on loglinear models, correspondence analysis and exploratory factor analysis to identify the key shallow skills for computer science graduates compared to psychology graduates. The preregistration is found at <https://osf.io/5qb6a>.

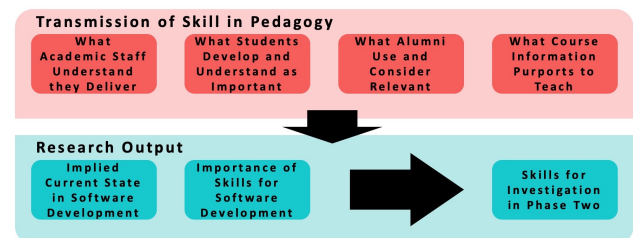
**3.1.3 Interviews with Staff and Students.** These two projects are planned, and interview schedules will be arranged for times when teaching volume is low for staff, and longitudinal student surveys will begin in line with the start of an academic year.

Staff interviews will look to identify soft skills considered important by teaching staff and how these are conveyed to students in teaching materials. Student interviews will be conducted over the course of the academic year, following the same students to identify the way in which they recognise and develop shallow skills. Analysis for both interview studies will use thematic and content analysis to extract relevant information.

**3.1.4 Data Analysis of Phase One.** The results from the individual projects in phase one can be cross-examined to identify areas of

shallow skills that are of most interest. The combined data can be used to understand the transmission of ideas from academic staff to students to what they take into the workplace, (see figure 2). Understanding the development of these skills and their importance can be used in the second phase. Analysis of data is in planning.

*Goal: to identify the shallow skills considered as important within the transmission of skills in the pedagogical process. Measured through various quantitative and qualitative methods.*



**Figure 2: Transmission of shallow skills can be expected to develop whereby teaching staffs' understanding of which skills are important feed into the skills students pick up on, which are reflected in alumni use of these skills.**

### 3.2 Phase Two: Behavioural Studies

The second phase is in early design stages, but will focus on empirical behavioural research, based on the findings from phase one. The exact soft skills to be included is dependent on phase one findings, as it is important to focus on the skills that are most likely to have the biggest effect on coding behaviours.

One study will investigate API blindspots, which can be defined as a misunderstanding or misrepresentation of API function security, resulting in vulnerabilities [16]. Using Python snippets from Brun et al (2021) [4], and measuring soft skills through cognitive tasks (e.g. the cognitive reflection test [7]), relationships can be drawn between soft skills and API blindspot awareness.

Similar studies, using different programming paradigms (such as code debugging, or secure password database creation) will also be used. It is important to understand the stability of soft skills across a range of development and security-related tasks.

**3.2.1 Data Analysis of Phase Two.** Data analysis for phase two experiments will be predominantly quantitative, using mixed effects models for group comparisons. These can be used to measure relationships between soft skills and security behaviours. Preregistrations will be published in due course.

*Goal: to identify and measure the effect of soft skills on security behaviours. Measured through mixed effect modelling and group comparisons.*

### 3.3 Phase Three: Inclusion in Pedagogy

For the final phase, the focus will be on the development of pedagogical materials for introducing students to the soft skills necessary for secure programming. This phase has not yet reached planning, as it relies on the work of phase two to be near completion. This will be achieved through seminars or workshops as methods to

<sup>2</sup>[www.osf.io](https://www.osf.io)

<sup>3</sup><https://nationalcareers.service.gov.uk/job-profiles/software-developer>

<sup>4</sup><https://nationalcareers.service.gov.uk/job-profiles/psychologist>



introduce the soft skills, to encourage students to engage with the psychology behind software development. Effectiveness of sessions will likely be measured through participant feedback.

To further disseminate research findings and promote inclusion of soft skills into current pedagogical materials, engagement through publication will be pursued. By raising awareness of research through publication, conferences and posters, along with the provision of basic materials for others to work with, phase three looks to create a meaningful impact in the domain of software learning development.

*Goal: to develop and deliver teaching materials in order to promote soft skills within computer science curricula. Measured through student engagement and feedback.*

### 3.4 Risk Perception and Cognitive Reflection

In this individual differences study aligned with phase two research, groups of professional software developers and computer science students were compared regarding risk perception in software. Participants completed a cognitive reflection test, a risk-oriented decision task, and answered qualitative questions about how they understand risk in software development.

Cognitive reflection is a person's ability to inhibit intuitive responses in favour of more reflective responses, indicating their skill in reflective thinking in search of a correct answer. Cognitive reflection was measured through the Cognitive Reflection Test (CRT) [7]. This is a three question test, including items such as, "A paperclip and an elastic band cost £1.10 in total. The elastic band costs £1 more than the paperclip. How much does the paperclip cost?" The intuitive answer is 10 pence, but upon reflection the correct answer is 5 pence. The risk-orientation task focussed on how participants view susceptibility of themselves and the "average developer" when considering security vulnerabilities as listed by OWASP (e.g. SQL injection). Data analysis is in progress. The preregistration can be found at: <https://osf.io/zbqeq>.

**3.4.1 Data Analysis of Risk Perception Study.** Data will be analysed through quantitative measures, such as linear modelling, along with more qualitative methods, including thematic and content analyses.

*Goal: to identify potential relationships between risk-related behaviours in software development and cognitive reflection. Measured through linear modelling and group comparisons.*

### 3.5 Validity Threats and Controls

Validity threats to the research are broadly discussed, relevant to the project overall. More granular considerations are included in preregistration documents.

One key threat is the consideration of software developers as a population. It is easy to treat developers as a homogeneous population who demonstrate similar characteristics, subsequently making approaches to promoting security behaviours intolerant to variance within the population. This can be controlled through mixed effect models, where population characteristics can be included in the analysis to identify the effect these have on behaviours.

This is a relatively new research field [17], so much of the planned research is exploratory. This can often lead to a series of analysis methods being used, increasing type I errors. To control for this, preregistration procedures are published prior to data collection. Open

data and reproducible analysis scripts will be uploaded following study completion, to allow replication and to confirm findings.

Using a range of methodologies, as highlighted in the work plan, may result in a trade-off between breadth of analysis and depth of analysis. To control for this, analysis plans and research choices will be well considered through the use of preregistration documents. By considering methodologies prior to execution, the connection between the studies within the wider research can be well justified.

Another threat to validity is the generalisation to different programming languages or work cultures. Not all languages have similar structures, and differences have been shown in security awareness between Java and Python APIs [4]. This can be controlled for by acknowledging that results may only apply to a single language. By focussing on Python, which is the most popular language<sup>5</sup>, findings will have relevance to many developers. The inclusion of preregistration documents, materials and analyses will also allow for replications, either directly or conceptually.

The tasks used in the second phase are designed to provide consistency across participants, reducing task variance and improving statistical power. This comes at a cost, which is that the tasks are less industry-specific, reducing the validity. This PhD research is specifically focussed on the software learning process, and work beyond the PhD may look into more industry specific tasks, or applying similar research to different programming languages.

## 4 RELEVANT PRIOR WORK

In this section the current literature relevant to the research is discussed. This is not an exhaustive literature review, but aims to identify key research influencing the doctoral research.

### 4.1 Phase One

In phase one, key research identified shallow skills in software development, such as Maturro et al. (2019) [14], who conducted a systematic analysis and identified 23 separate skills. Similarly, Stevens and Norman (2016) looked at job adverts to identify the most important shallow skills for developers [20]. These research papers provided context for the important shallow skills.

Groeneveld et al. [10] analysed computer science curricula for modules that taught shallow skills explicitly, but did not look into the implicitly taught skills in all modules. This motivated the investigation of the course curricula for text relevant to shallow skills.

Finding an absence of research that provided associations between shallow skills and soft skills is the motivation for phase one. The literature search for phase one has found little evidence of work associating security awareness and soft skills.

### 4.2 Phase Two

In the second phase, a series of work has been carried out concerning API blindspots and developers' use of heuristics when evaluating software code. Oliveira et al. (2018) [16] highlighted this issue with Java puzzles, finding that security blindspots in code snippets were difficult to identify, possibly due to developers' expectation of APIs being secure as default. Brun et al. (2021) [4] followed this work with a replication using Python code. They found that developers who exhibited better long term memory recall were

<sup>5</sup><https://www.tiobe.com/tiobe-index/python/>

more successful in solving puzzles with blindspots. They found that short term memory, memory span and episodic memory had no effect on solving the puzzles. Other works that touch on psychology in security include Hallett et al (2021) [12], where boosting security awareness through requiring planning promoted a small effect on security, and Shreeve et al (2020) [19] who identified decision making processes related to cybersecurity.

### 4.3 Phase Three

For the third phase, Taylor-Jackson et al. (2020) [21] advocated including psychology in security education, particularly when considering that vulnerabilities are often psychological in nature (e.g. phishing, API blindspots). They discuss the benefits of exposing computer scientists to the different ideas and styles of thinking found within psychology. There are also wider calls for inclusion of soft skills in university education [11]. It is important that the findings from the first two research phases are used for positive impact and one immediate way to achieve this, is to answer the calls for increasing soft skill teachings in cybersecurity courses to benefit future software developers.

### 4.4 Risk Perception

For this study, key items are papers on cognitive reflection by Frederick (2005) [7] and Thomson and Oppenheimer (2016) [22]. Combined with the understanding that developers are often not the most security conscious, as highlighted by Acar et al (2017) [1], it is clear that the understanding of risk by developers in a software context is poorly understood in relation to cognitive measures.

## 5 CURRENT STATUS

### 5.1 Early Results Analysis

Some of the preliminary results from the risk perception study (section 3.4) are mentioned here. The third hypothesis stated in the preregistration is examined here, "Mean scores closer to zero on the novel OWASP risk task will be found with higher scores of cognitive reflection". Data from 143 (70 students, 73 developers) participants is used.

The OWASP task is a measure devised for this study where participants were asked to respond to two sets of questions, the first asking about the percentage of web applications they believe to be created by others that suffer from one of the top five OWASP vulnerabilities (injection flaws, broken authentication, sensitive data exposure, XML External Entity and broken access control). Then following a separation task, participants were then asked to rate the percentage of web applications that they had developed that suffered from the same vulnerabilities. Scores closer to 100 indicate high optimism that they do not produce flawed products, scores near 0 indicate similar levels of flaws in both their own and other people's products, and scores approaching -100 indicate beliefs that their own work is highly susceptible to these vulnerabilities.

To test the hypothesis above, a linear regression was run to see whether CRT scores significantly predicted OWASP vulnerability scores and whether this differed between the two populations. The model formula was "Vulnerability ~ CRT". The overall regression was statistically significant ( $R^2 = .05$ ,  $F(3, 139) = 15.13$ ,  $p = .017$ ). Estimates, values and significance of model items can be seen in Table

**Table 1: Coefficients, t-values and p-values for the linear regression of CRT predicting OWASP vulnerability**

	Estimate	Std. Error	t value	$p^*$
Intercept	11.00	3.13	3.52	<.001***
CRTscore1	14.08	4.59	3.07	.003**
CRTscore2	5.61	4.86	1.15	.250
CRTscore3	10.67	4.91	2.17	.031*

\*Significant alpha values of <.001 indicated by \*\*\*

1. Despite significant terms in the model summary, the variance explained by the model is negligible (~5%) and further models will need to be developed to explain more variance in these scores.

Figure 3 shows the distribution of vulnerability scores for each level of CRT score as a box plot. Post-hoc Tukey tests identified no significant differences between any of the groups with all  $p > .05$ , except for those who scored zero and those who correctly scored one, adjusted  $p = .014$ . This indicates that there is little significant relationship between CRT scores and results on the novel OWASP risk task.

What is noted with the Vulnerability scores, and can be seen in Figure 3, is that most scores on the OWASP task, regardless of CRT scores, are around or above 0. One-way t-tests on the Vulnerability scores were run for both the developer and the student samples. In the Developer sample (mean score = 17.48), the scores were significantly higher than 0,  $t(69) = 7.16$ ,  $p < .001$ . Similarly for the student sample (mean score = 18.94), scores were significantly higher than 0,  $t(69) = 7.26$ ,  $p < .001$ .

This finding is indicative of optimism bias [18], suggesting that both professionals and student developers consider themselves to be better than average at preventing these OWASP-listed security issues. A score of zero would indicate respondents understand they were average, but higher scores suggest an over-optimistic outlook on their own abilities, which could lead to a more relaxed view on these security issues. These findings will be further developed and discussed in future publications.

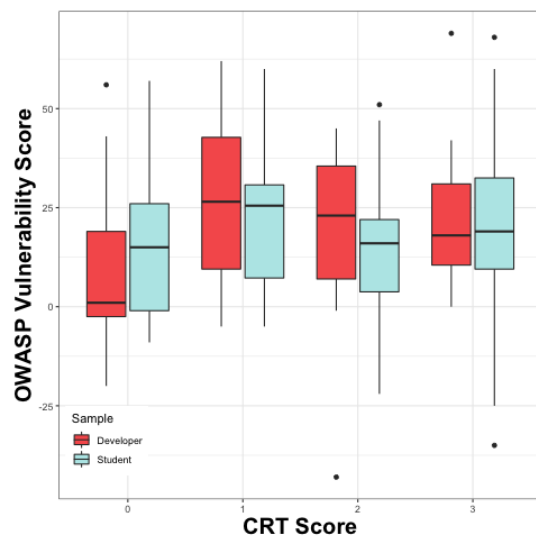
### 5.2 Next Steps

In the short term, the next steps are to continue with phase one data collection, and planning of phase two. It is intended that the research will progress according to the research phases outlined above. Following the completion of the doctoral work, future work would include the investigation of psychological interventions for developing soft skills and measuring their impact on security behaviours in longitudinal research.

The steps beyond the PhD research as outlined above is to focus on the skills that explain the largest variance in secure coding behaviours, and seek to identify the best ways to promote continued, stable use of these behaviours as opposed to short-term changes (such as those achieved through nudging, e.g. IDE pop-ups serving as reminders to look for blindspots).

## 6 CONCLUSION

This paper provides an overview of the planned work within the PhD research titled "The Soft Skills of Software Learning Development: the Psychological Dimensions of Computing and Security



**Figure 3: Box plot of mean OWASP vulnerability scores by CRT score split by population.**

Behaviours". The research is diverse in both aims and processes, ranging from thematic analysis of interviews, to modelling relationships between psychological dimensions and security issues, to incorporating the findings into pedagogy.

This project seeks to investigate the software learning development process; to better understand the behavioural changes and soft skill development of both computing and security behaviours. By identifying these changes, and when and how they develop, we can seek to promote these changes earlier in the learning cycle, allowing for more effective learning and encouraging positive behaviours for software development. In doing so, not only can we exhibit greater awareness of the psychology behind secure software development, we can develop interventions for encouraging these secure behaviours, reducing the likelihood of these security vulnerabilities. All public preregistrations, published data, analyses, and links to further research outputs will be accessible from <https://osf.io/v93zt>.

## REFERENCES

- [1] Yasemin Acar, Christian Stransky, Dominik Wermke, Charles Weir, Michelle L. Mazurek, and Sascha Fahl. 2017. Developers Need Support, Too: A Survey of Security Advice for Software Developers. In *2017 IEEE Cybersecurity Development (SecDev)*. IEEE, Cambridge, MA, USA, 22–26. <https://doi.org/10.1109/SecDev.2017.17>
- [2] Anne Adams and Martina Angela Sasse. 1999. Users Are Not the Enemy. *Commun. ACM* 42, 12 (Dec. 1999), 40–46. <https://doi.org/10.1145/322796.322806>
- [3] Faheem Ahmed, Luiz Fernando Capretz, Salah Bouktif, and Piers Campbell. 2015. Soft Skills and Software Development: A Reflection from the Software Industry. *International Journal of Information Processing and Management* 4, 3 (July 2015), 171–191. <https://doi.org/10.4156/ijipm.vol14.issue3.17> arXiv:1507.06873
- [4] Yuriy Brun, Tian Lin, Jessie Elise Somerville, Elisha Myers, and Natalie C. Ebner. 2021. Blindspots in Python and Java APIs Result in Vulnerable Code. *arXiv:2103.06091 [cs]* (March 2021). arXiv:2103.06091
- [5] Luiz Fernando Capretz and Faheem Ahmed. 2018. A Call to Promote Soft Skills in Software Engineering. *Psychology and Cognitive Sciences - Open Journal* 4, 1 (Aug. 2018), e1–e3. <https://doi.org/10.17140/PCSOJ-4-e011> arXiv:1901.01819
- [6] Silvia Fareri, Nicola Melluso, Filippo Chiarello, and Gualtiero Fantoni. 2021. SkillNER: Mining and Mapping Soft Skills from Any Text. *Expert Systems with Applications* 184 (Dec. 2021), 115544. <https://doi.org/10.1016/j.eswa.2021.115544>
- [7] Shane Frederick. 2005. Cognitive Reflection and Decision Making. *Journal of Economic perspectives* 19, 4 (2005), 25–42.
- [8] Steven Furnell and Matt Bishop. 2020. Addressing Cyber Security Skills: The Spectrum, Not the Silo. *Computer Fraud & Security* 2020, 2 (Feb. 2020), 6–11. [https://doi.org/10.1016/S1361-3723\(20\)30017-8](https://doi.org/10.1016/S1361-3723(20)30017-8)
- [9] Matthew Green and Matthew Smith. 2016. Developers Are Not the Enemy!: The Need for Usable Security APIs. *IEEE Security Privacy* 14, 5 (Sept. 2016), 40–46. <https://doi.org/10.1109/MSP.2016.111>
- [10] Wouter Groeneveld, Brett A. Becker, and Joost Vennekens. 2020. Soft Skills: What Do Computing Program Syllabi Reveal About Non-Technical Expectations of Undergraduate Students?. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education (ITICSE '20)*. Association for Computing Machinery, New York, NY, USA, 287–293. <https://doi.org/10.1145/3341525.3387396>
- [11] Sandra Patricia Guerra-Báez. 2019. A Panoramic Review of Soft Skills Training in University Students. *Psicología Escolar e Educativa* 23 (2019), 1–10. <https://doi.org/10.1590/2175-35392019016464>
- [12] Joseph Hallett, Nikhil Patnaik, Benjamin Shreeve, and Awais Rashid. 2021. "Do This! Do That!, And Nothing Will Happen" Do Specifications Lead to Securely Stored Passwords?. In *Proceedings of the 43rd International Conference on Software Engineering (ICSE '21)*. IEEE, Madrid, Spain, 486–498. <https://doi.org/10.1109/ICSE43902.2021.00053>
- [13] Janet Liebenberg, Magda Huisman, and Elsa Mentz. 2014. Knowledge and Skills Requirements for Software Developer Students. *International Journal of Social, Behavioral, Educational, Economic, Business and Industrial Engineering* 8, 8 (2014), 6.
- [14] Gerardo Matturro, Florencia Raschetti, and Carina Fontán. 2019. A Systematic Mapping Study on Soft Skills in Software Engineering. *Journal of Universal Computer Science* 25, 1 (2019), 26.
- [15] João Eduardo Montandon, Cristiano Politowski, Luciana Lourdes Silva, Marco Tulio Valente, Fabio Petrillo, and Yann-Gaël Guéhéneuc. 2021. What Skills Do IT Companies Look for in New Developers? A Study with Stack Overflow Jobs. *Information and Software Technology* 129 (Jan. 2021), 106429. <https://doi.org/10.1016/j.infsof.2020.106429>
- [16] Daniela Seabra Oliveira, Tian Lin, Muhammad Sajidur Rahman, Rad Akefirad, Donovan Ellis, Eliany Perez, Rahul Bobhate, Lois A DeLong, Justin Capps, and Yuriy Brun. 2018. {API} Blindspots: Why Experienced Developers Write Vulnerable Code. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*. USENIX Association, Baltimore, MD, USA, 315–328.
- [17] Irum Rauf, Marian Petre, Thein Tun, Tamara Lopez, Paul Lunn, Dirk Van Der Linden, John Towse, Helen Sharp, Mark Levine, Awais Rashid, and Bashar Nuseibeh. 2021. The Case for Adaptive Security Interventions. *ACM Transactions on Software Engineering and Methodology* 31, 1 (Sept. 2021), 9:1–9:52. <https://doi.org/10.1145/3471930>
- [18] Tali Sharot. 2011. The Optimism Bias. *Current Biology* 21, 23 (Dec. 2011), R941–R945. <https://doi.org/10.1016/j.cub.2011.10.030>
- [19] Benjamin Shreeve, Joseph Hallett, Matthew Edwards, Pauline Anthonysamy, Sylvain Frey, and Awais Rashid. 2020. "So If Mr Blue Head Here Clicks the Link." Risk Thinking in Cyber Security Decision Making. *ACM Transactions on Privacy and Security* 24, 1 (Nov. 2020), 5:1–5:29. <https://doi.org/10.1145/3419101>
- [20] Matt Stevens and Richard Norman. 2016. Industry Expectations of Soft Skills in IT Graduates: A Regional Survey. In *Proceedings of the Australasian Computer Science Week Multiconference (ACSW '16)*. Association for Computing Machinery, New York, NY, USA, 1–9. <https://doi.org/10.1145/2843043.2843068>
- [21] Jacqui Taylor-Jackson, John McAlaney, Jeffrey L. Foster, Abubakar Bello, Alana Maurushat, and John Dale. 2020. Incorporating Psychology into Cyber Security Education: A Pedagogical Approach. In *Financial Cryptography and Data Security (Lecture Notes in Computer Science)*, Matthew Bernhard, Andrea Bracciali, L. Jean Camp, Shin'ichiro Matsuo, Alana Maurushat, Peter B. Rønne, and Massimiliano Sala (Eds.). Springer International Publishing, Cham, 207–217. [https://doi.org/10.1007/978-3-030-54455-3\\_15](https://doi.org/10.1007/978-3-030-54455-3_15)
- [22] Keela S Thomson and Daniel M Oppenheimer. 2016. Investigating an Alternate Form of the Cognitive Reflection Test. *Judgment and Decision making* 11, 1 (2016), 99.
- [23] John Towse, Mark Levine, Marian Petre, Arosha Bandara, Tamara Lopez, Awais Rashid, Irum Rauf, Helen Sharp, Thein Tun, Dirk van der Linden, and Bashar Nuseibeh. 2020 (in press). The Case for Understanding Secure Coding as a Psychological Enterprise. *Cyberpsychology, Behavior, and Social Networking* (2020 (in press)).
- [24] Glenn Wurster and Paul van Oorschot. 2008. The Developer Is the Enemy. In *Proceedings of the 2008 New Security Paradigms Workshop (NSPW '08)*. Association for Computing Machinery, New York, NY, USA, 89–97. <https://doi.org/10.1145/1595676.1595691>

### 3.1 Statement of Continuous Thesis Summary

*“Everyone has a plan ’till they get punched in the face” - Mike Tyson, 1987*

In this chapter, I present the intended plan for the PhD project. I detail the project, its motivations, and describe the intended research phases as published in 2022. This chapter serves as a primer for the remainder of the thesis, with the phased work plan being the main contribution, outlining and justifying the components of each phase. Scholarly work is not always straight like an arrow, and the chronology of the research phases did not take place in the precise linear sequence of thesis chapters. Instead, components were pursued in overlapping waves. Nonetheless, preliminary findings from phase 1 still informed phase 2 decisions and offered constructive information.

Elements from all three phases were completed, but some components could not be addressed (discussed further in Chapter 9), and ultimately, resource constraints meant that only some of the intended research could be completed. The work plan was conceptualised during the early stages of the programme and formalised for publication in January 2022, 15 months after PhD commencement, and with 21 months remaining, it naturally evolved over this time. Some gaps between plans and activities can be attributed to naïvety, but my personal development replaced this with realism over the PhD development. The absence of these components does not detract from the substance of the thesis and the meaningful contributions that the work can make to our understanding of the psychological dimensions of computing and security behaviours in software engineers.

The following five chapters cover the research conducted, with phase 1 comprising Chapters 4 and 5, and phase 2 comprising Chapters 6, 7, and 8. The next chapter begins phase 1 by exploring the perceptions of CS graduates regarding the value of soft skills. This is important to understand as it offers the opportunity to link their skill development to education and to identify the most important non-technical aspects of their careers.

#### 4 What's in an undergraduate Computer Science Degree; Alumni perceptions about soft skills in careers

Ivory, M., Towse, J., Sturdee, M., Levine, M., & Nuseibeh, B. (*in review*). What's in an undergraduate Computer Science Degree; Alumni perceptions about soft skills in careers. *Transactions on Computing Education*. <https://osf.io/8me4w>

## What's in an undergraduate Computer Science degree? Alumni perceptions about soft skills in careers

MATTHEW IVORY, Lancaster University, Great Britain

JOHN TOWSE, Lancaster University, Great Britain

MIRIAM STURDEE, University of St Andrews, Scotland

MARK LEVINE, Lancaster University, Great Britain

BASHAR NUSEIBEH, LERO, Republic of Ireland and Open University, United Kingdom

Software engineering skills are broad and varied, encompassing not only technical abilities, but cognitive and social dimensions as well. Previous research establishes soft skills as being central for software engineering, e.g., teamwork, communication, and problem solving, but the relationship between these skills and how higher education prepares future software engineers for the workplace is unclear. These programmes should teach students the soft skills required for their careers, but a better understanding of which skills are valued and taught is needed. The perceptions of undergraduate alumni about the value of soft skills and where they were developed are reported on in this preregistered study of computer science and psychology alumni (using the latter as a contrast group). Replicating previous conclusions, computer science alumni rated problem solving, communication, and teamwork as amongst the most important skills, but when examining where these skills originate from, problem solving was strongly associated with education, and teamwork was strongly linked with employment. Communication is weakly associated with education, with influence from employment sources too. Alumni working in security-related careers highlighted that organisational skills were important more so than for the alumni population more broadly. We make recommendations for prioritisation of skill development based on the skills reported by the alumni. We suggest potential ways to encourage learning opportunities that promote scenarios students will find familiar during their careers, better preparing students for the needs of the workplace.

CCS Concepts: • **Security and privacy** → **Social aspects of security and privacy**; • **Human-centered computing** → *Empirical studies in HCI*;

Additional Key Words and Phrases: graduate employability, faculty perceptions, soft skills, computer science, higher education

---

Authors' addresses: Matthew Ivory, Lancaster University, Department of Psychology, Lancaster, Lancashire, LA1 4YW, Great Britain, matthew.ivory@lancaster.ac.uk; John Towse, Lancaster University, Department of Psychology, Lancaster, Lancashire, LA1 4YW, Great Britain, j.towse@lancaster.ac.uk; Miriam Sturdee, University of St Andrews, School of Computer Science, North Haugh, St Andrews, LA1 4YW, Scotland, m.sturdee@lancaster.ac.uk; Mark Levine, Lancaster University, Department of Psychology, Lancaster, Lancashire, LA1 4YW, Great Britain, mark.levine@lancaster.ac.uk; Bashar Nuseibeh, LERO, Ireland, Lancaster, Lancashire, LA1 4YW, Republic of Ireland, Open University, United Kingdom, bashar@lancaster.ac.uk.

---

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

© 2023 Association for Computing Machinery.

Manuscript submitted to ACM

**ACM Reference Format:**

Matthew Ivory, John Towse, Miriam Sturdee, Mark Levine, and Bashar Nuseibeh. 2025. What’s in an undergraduate Computer Science degree? Alumni perceptions about soft skills in careers. 1, 1 (July 2025), 29 pages. [https://doi.org/10.475/123\\_4](https://doi.org/10.475/123_4)

**1 INTRODUCTION**

The importance of software in modern society cannot be understated, and subsequently the skills required to create functionally useful and secure software should be incorporated into Computer Science (CS) higher education programmes. Developing software is a complex cognitive-socio-technical enterprise, requiring more than just skills of programming languages and systems [24]. Software engineers’ cognition and social skills play a critical role in project success [41, 63], but where we have a reduced understanding of the origin of soft skills, the ability to prepare students for employment is limited. While educators may design curricula to incorporate specific skills, if students do not perceive these skills to be present, this limits their employability.

Some skills cannot be adequately developed prior to employment, and so employers will also need to commit to training new employees, and prospective employees are constrained in what they can offer. For technical skills, such as proprietary tools, training is simply a necessity. For non-technical skills, these are often domain-agnostic and can be developed prior to employment. A small amount of training may be required to align the existing skills with employer’s cultures or processes, but skills of teamwork or problem solving are typically applicable in a great variety of roles. Ensuring that students are developing these skills during their education improves their graduate employability.

Human behaviours are often referred to as *soft skills* or non-technical skills [30, 48], and engineers should be able to recognise the value of soft skills and use them in practice [9]. Soft skills are domain-agnostic, covering a range of psychologically-rooted abilities, including interpersonal communication, habits, and cognition [2], and soft skills often determine the success of technical skill application [26, 49]. In the context of software development, some of the more valued soft skills include problem solving, communication, and teamwork [3]. For security-related software development, various forms of communication, teamwork, and social skills are required for working in cybersecurity roles [39].

Many software engineers or those working in adjacent fields complete an undergraduate CS education. These undergraduate programmes have a significant role in preparing students for future employment, offering students opportunities to practice and develop their soft skills in controlled environments. Due to increasingly levels of security awareness around software, these programmes often incorporate cybersecurity and security training, and so it is important to understand what skills are considered important for these industries too.

It is imperative we understand the relationship between the skills that are valued within software engineering and the skills that are developed during an undergraduate education. In doing so, our findings can inform CS undergraduate programmes to address all relevant skills. CS educators have a responsibility to expose their students to opportunities to develop these skills, and increase student awareness as to their importance, and the present research maps this responsibility to the skills used in employment through exploring graduate perceptions.

To address the limited understanding of how soft skills bridge between education and employment, we surveyed CS undergraduate alumni about the skills valued in the workplace, where they developed these

skills, and explored the skills required in security roles. In previous research in this domain, the population is considered in a vacuum, lacking any comparative groups for reference. To address this shortfall, we collected data from psychology alumni to form the reference group. Specifically, CS undergraduate programmes are often highly driven to prepare students for careers in software engineering. This makes sense as over 79% of graduates find employment in software engineering [6]. In contrast, many psychology graduates enter non-psychology careers, such as sales or teaching [58], and psychology professions often require graduate-level programmes. This combination of participants provides a unique perspective to our research that is absent in other related work.

*1.0.1 Hypotheses and Research Questions.* Our research was driven to better understand the mapping between undergraduate CS education and employment activities, with a complementary analysis of security-related work. Previous studies often focus on employers or current employees in a particular domain, but by placing the focus on individuals as CS alumni, we can make stronger associations between their skills used and how these were formed or developed during their education, which can offer implications for undergraduate pedagogy, as it highlights the skills that are considered important, but also whether they are taught effectively in the programme or not. We were motivated to make comparisons between two undergraduate programmes, as through a comparison of CS and psychology alumni gives a contrast in the skills necessary and highlights how the skills are emphasised differently in their education, as well as how these then correspond to employment.

We also explore the soft skills and alumni perceptions within security-related employment as cybersecurity increasingly becomes more and more valued within software engineering. As more universities offer cybersecurity courses [52], it is key to ensure the skills required with security sectors are reflected in the education offered within CS courses. If different skills are required when working in security-sensitive domains, then these requirements can be used to help inform conversations around the skills highlighted and emphasized in present education programmes.

To focus the research, two hypotheses were preregistered:

H<sub>1</sub>. Computer Science undergraduate alumni will prioritise different groupings of soft skills to Psychology alumni, representing two different populations of perceived soft skill importance.

H<sub>2</sub>. Computer Science undergraduate alumni involved in security-related work will prioritise different soft skills than those who are less involved in security.

To help structure the analysis plan, and complement the hypotheses, five research questions were preregistered at (<https://osf.io/5qb6a>). These questions help in breaking down the complex dataset.

RQ<sub>1</sub>: How do undergraduate alumni understand the importance of soft skills in their employment (current or most recent)?

RQ<sub>2</sub>: How do Computer Science and Psychology undergraduate alumni differ in the soft skills deemed important for their workplace?

RQ<sub>3</sub>: How do undergraduate alumni recall being introduced to soft skills, via education or through employment?

RQ<sub>4</sub>: How do the technical skills possessed by alumni relate to the perceived importance of soft skills in employment?



RQ<sub>5</sub>: How do Computer Science undergraduate alumni differ in their use of security in their work, and how do the required soft skills differ between those who use security and those who do not?

### 1.1 Contribution

Our research provides three unique contributions:

1. We identify the skills considered to be the most important by CS alumni, these being problem solving, communication, responsibility, teamwork, and analytical thinking.
2. We identify links between skills and their sources of development (through education, employment, or personal means), where education is strongly associated with problem solving and analytical thinking, teamwork with employment, and communication is not strongly associated with any one source.
3. We highlight the soft skills that are considered as the most important for those who work in a security context, with those working in higher security domains reporting important skills of communication, problem solving, and organisation.

### 1.2 Structure

The rest of the paper is structured as follows. Section two explores the existing literature surrounding soft skills in software engineering and CS. Section three describes the methodology used, including the participants, measures, and analysis processes. Section four provides the results. Section five discusses the results and implications. Section six concludes the paper.

## 2 RELATED WORK

### 2.1 Psychology within Software Engineering

Psychology's important role in software engineering has become increasingly visible [14], with an understanding technical skill alone does not define an exceptional software engineer [17, 30]. Soft skills are psychological in nature, encompassing personality, habits, social aptitude, and cognition [2]. They can be considered as the surface features underpinned by psychologically quantifiable measures and theories.

In recent years research has been carried out in software engineering with psychology as the focus, examining behaviours involved in secure software development. Security provides an additional complexity to software development and reflects the requirement to protect personal and sensitive information. Technical knowledge and experience have little effect for secure software development [10, 53], suggesting social or cognitive factors must be involved, reaffirming a need to better understand the soft skills and psychology linked to secure development.

Software engineers often assume tools are secure by default, increasing the chances of security vulnerabilities [50]. Long term memory recall was linked to an increased detection of insecure code [10], perhaps indicating a greater ability to recall relevant technical information. Individual differences play a role in security, and cognitive biases impact perceptions of risk [37], as well as cognitive cues prompting greater security behaviours [33, 51]. Social identity plays a role in secure development, and developers that perceive more group differences in a project subsequently experience reduced responsibility and commitment to testing software security [36]. Others report increased interaction with security teams encourages security behaviours in others [64], likely due to an increased sharing of social identities.

**2.1.1 Soft skills in Industry.** The understanding of which soft skills are relevant for software engineering often comes from industry data, such as job adverts or interviews with current employees or managers. Two of the commonly reported soft skills are teamwork and communication [3, 30, 38, 46, 48]. These findings indicate that the industry is group-oriented, requiring communication, both between teams and stakeholders. Another key skill was problem solving [3, 22, 45]. Problem solving requires a mindset allowing for real-world problems to be interpreted within a software setting. Organisation was frequently mentioned as necessary to guarantee projects can be maintained even if employees leave by ensuring they adhere to workflows and institutional decisions [3].

Soft skills are seen to improve employee integration and enhance team productivity [59], highlighting the importance of soft skills for using technical skills effectively. Other employers note that technical skills are easier to teach compared to soft skills [23, 43].

**2.1.2 Soft Skills in Cybersecurity.** Within cybersecurity, communication is seen as crucial [5], along with critical thinking, adaptability, innovation, and interpersonal skills [34]. Communication is needed for conveying the need for security or security information, along with being able to communicate to a wide audience. Interpersonal skills included displaying positivity and building rapport and trust between clients and peers. Decision making and problem solving were highlighted [61], and organisational skills, as a requirement for creating knowledge bases [27].

In exploring where these soft skills originated from, there is no clear consensus of a single origin, and a conflict between perception of many soft skills being immutable and innate, and a split between education and employment [39]. Others rated communication, motivation, and a willingness to learn/curiosity as a priority for being taught in cybersecurity education [5].

Communication, problem solving, organisational, and interpersonal skills have been mentioned already in general software engineering, indicating that cybersecurity skill requirements are similar to those in general software engineering too.

**2.1.3 Soft skills in Education.** The higher education sector is partly responsible for preparing students for employment. Groeneveld et al. [31] identified self-reflection, conflict resolution, communication, and teamwork as the most taught skills, which were reflected in a further examination of modules focusing on soft skills, identifying teamwork, ethics, communication, and presentation skills [29]. Others identified ethics and analytical thinking styles as important [18]. Research conducted with alumni by Calitz et al. [13] and Watson and Blincoe [62] found skills of problem solving, teamwork and communication prioritised above other skills.

It is seen that student perceptions of important soft skills are different to the skills valued by employers [60], with graduates not possessing the relevant skills for employment [54]. Students entering CS programmes come with preconceptions that technical skills, such as programming, are the more valuable skills and prioritise these over soft skills [35]. Students often confuse skill importance with enjoyment, associating lower value with less-enjoyable activities [42]. It is wrong to assume students will strengthen the appropriate skills, or that soft skills develop organically without opportunities, and so it is crucial CS degrees facilitate students' development of these skills. It is imperative students are provided with ways to upskill [8], meeting prospective employer's wishes for pre-existing soft skills [4].

We offer a unique contribution through the report on the present study, contributing to our existing knowledge around the soft skills needed within software engineering. Previous research has identified the skills desired by software engineering employers, and security-specific domains. Others have noted that student perceptions do not necessarily align with industry expectations, and that they undervalue soft skills compared to technical competencies. We offer a greater understanding of how CS alumni perceive soft skills as being valued in their employment, and where these skills were developed. This work contributes to closing the gap between student skills and employer expectations, by identifying the sources of important skills.

### 3 METHODS

#### 3.1 Participants

Alumni from Lancaster University were contacted if they had completed a degree in CS ( $n = 750$ ) or psychology ( $n = 1,576$ ). An 11.62% response rate was achieved (276 responses), and 128 were retained for analysis. The 128 responses contained enough data to answer most research questions (1, 2, 4, and 5) and 100 of these responses were able to be used for answering question 3. All participants provided informed consent.

Of the 128 participants, 69 were psychology alumni, and 59 were CS alumni. The spread of graduation years is reported in Figure 1. The ages of respondents were similar between departments with a mean age of 41.32 ( $SD = 13.97$ , median = 38.00) in psychology, and 48.03 ( $SD = 15.92$ , median = 50.00) for CS. Reported gender for CS alumni was 13.56% female and 86.44% male, and psychology alumni reported 59.42% female, 37.68% male and 2.90% other. Figure 2 shows participant gender and age.

For CS alumni, 54 (91.53%) had worked in an industry or role related to software engineering post-graduation, compared to 9 (13.04%) psychology alumni. For psychology employment, 25 psychology alumni (36.23%) and four CS alumni (6.78%) reported related roles. This reflects the high versus low industry-preparation status of these degree pathways.

In this research, our demographic of interest were alumni who completed an undergraduate degree at Lancaster University, and in the rest of the paper, references to “university” explicitly refer to the institution where the alumni completed their undergraduate education. It is acknowledged that “university” can be considered a Westernised term, but we use it in the context of the institution where the sampled students studied.

#### 3.2 Measures

The survey was presented through Qualtrics, an online surveying tool, and completed remotely. The survey is found on OSF (<https://osf.io/s52r7/>). Figure 3 shows survey flow, consisting of an information sheet, informed consent, and demographic information. The demographic questions included age, gender, education, employment status, job history and department of study. Participants were asked about their technical skills, with CS participants asked about security usage. No restrictions were placed on what participants considered to be technical skills. Following this, both groups ranked 23 skills (detailed in Skill definition validation) on a five-point Likert scale ranging from “Not Important” to “Very Important” in relation to current employment. Participants were asked where these skills were developed, undergraduate education, employment, or personal development. Finally, participants were debriefed.

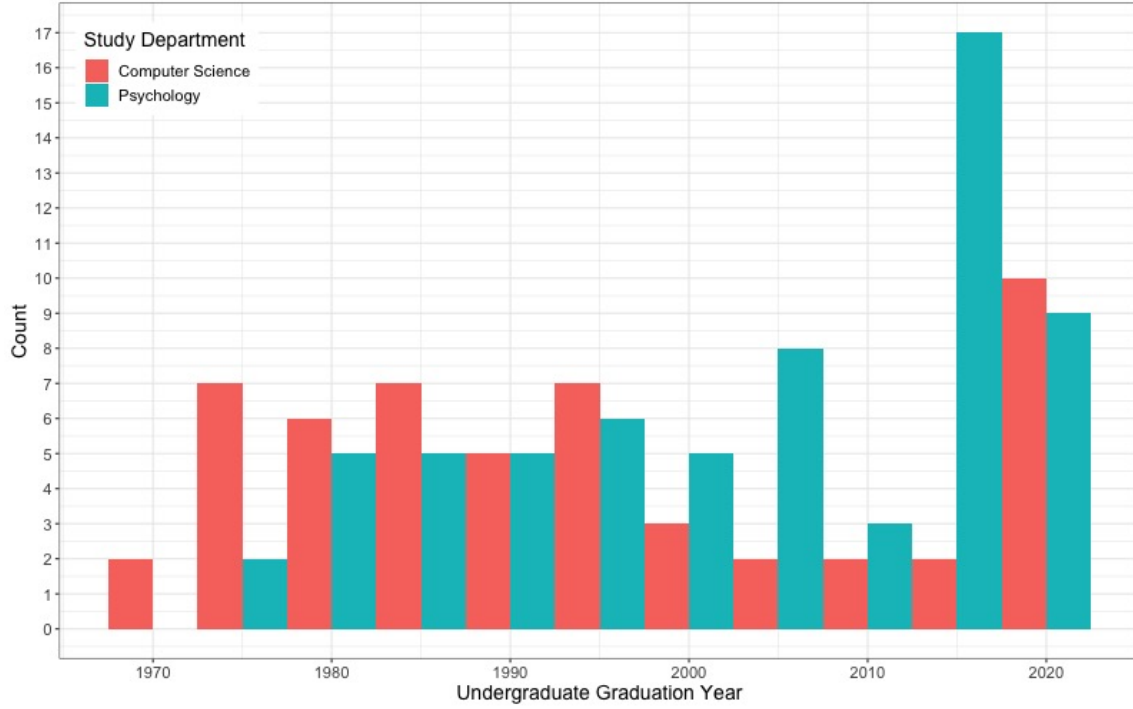


Fig. 1. Histogram of undergraduate graduation year split by department. Bars are grouped into ten-year intervals.

### 3.3 Procedure

**3.3.1 Skill definition validation.** A recent study found over 400 different soft skill types [19], highlighting the sheer magnitude of skills. Presenting participants with over 400 options was not appropriate, and so we developed a select list of commonly identified skills. Items were taken from Stevens and Norman [59], Matturro et al. [46], and Ahmed et al. [3], resulting in 23 distinct soft skills. The definitions of these skills were created using the above works and their cited sources.

To assess definition distinctiveness, they were validated using an independent participant pool recruited through Prolific, an online recruitment platform. Over three rounds of validation were conducted with 20 participants in each round. The task had participants presented with a list of soft skills and definitions and were then asked to match each definition to the most appropriate term. Multiple terms could be selected and were ordered.

Following each round, definitions were updated to aim for term distinctiveness. To identify these definitions, mean rankings, standard deviations and frequency counts were calculated. These were combined to provide a metric:  $\frac{\text{mean} \times SD}{\text{count}}$ . Correct definition-term items with a score less than half of the next score were deemed appropriate, and terms with a count less than the square root of total participant count were not considered to be strong associations. After three rounds, definitions were considered appropriate and with high distinction. See (<https://osf.io/s52r7/>) for the skills and definitions.

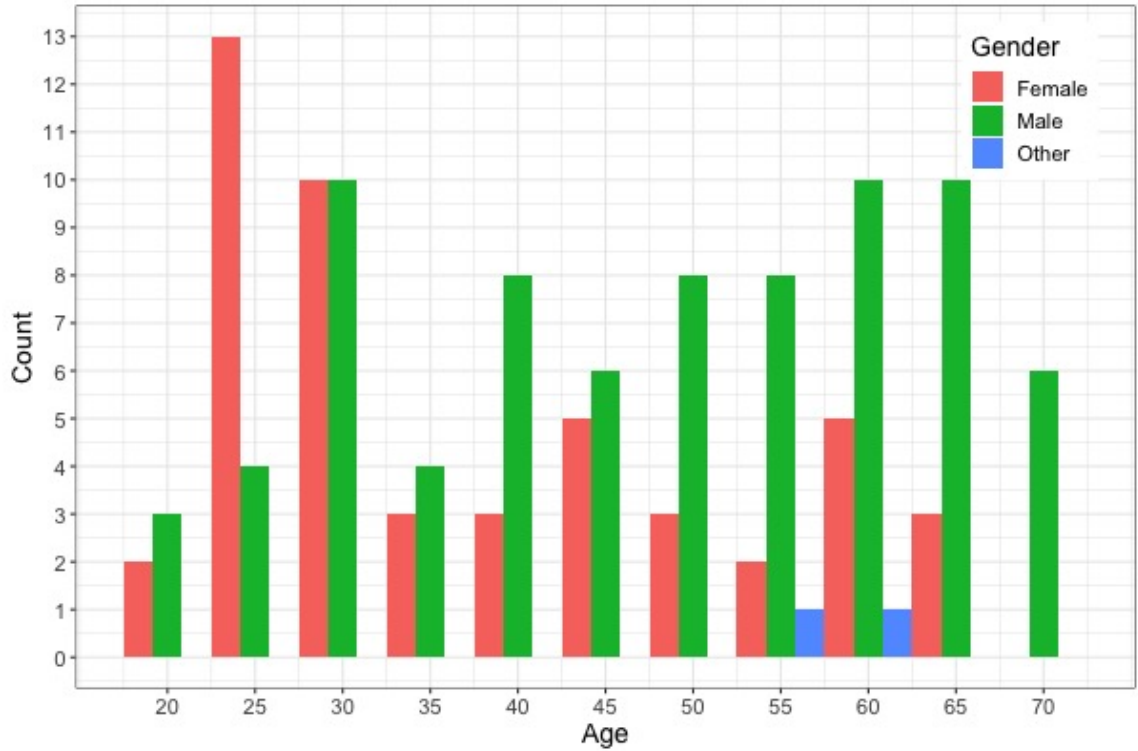


Fig. 2. Histogram of age and gender splits. Bars are grouped into five-year intervals.

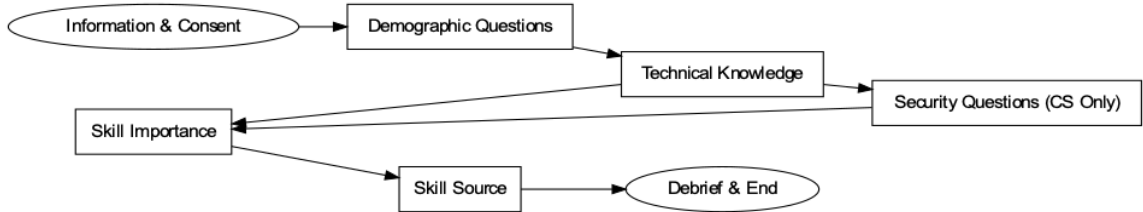


Fig. 3. Survey flow with CS alumni filling out an additional section regarding their experience working with security. This focussed on whether their employment was linked to security work, and how much of their work is concerned with security.

### 3.4 Data analysis

Two analysis methods were used, correspondence analysis with ordinal regression and factor analysis. Correspondence analysis offers the identification of relationships between individual skills against other categorical data (such as skill importance), and exploratory factor analysis (EFA) can be used to identify latent skillsets.

The analysis methods were preregistered (<https://osf.io/5qb6a>). The conducted analysis deviated from the preregistration as it used ordinal regression rather than loglinear regression. Both achieve a similar

Manuscript submitted to ACM

goal, but the interpretation of large loglinear models is difficult and requires a reference skill, meaning the interpretation is always a comparison between skills.

**3.4.1 Correspondence Analysis.** Correspondence analysis represents relationships between categorical and ordinal variables in reduced dimensional space. It shows skill importance relative to other skills, not necessarily which skills are the most categorically important. Two interpretation methods are used; the first being the absolute distance of a point from the origin (0,0) indicating skill strength, and the angle between the vector of importance rating drawn from the origin and the vector of skill. Smaller angles indicate higher association, and right-angles indicate no association.

When working with ordinal data, correspondence analysis requires “doubling” [28], anchoring responses against upper and lower bounds to differentiate between constant responses, otherwise a response of “Strongly Agree” on a Likert scale (5,5,5,5) is interpreted as having the same profile as all “Strongly Disagree” (1,1,1,1). Joint correspondence analyses allow for comparisons between two or more groups.

**3.4.2 Ordinal Logistic Regression.** Ordinal logistic regression, or proportional odds logistic regression are used to build upon the correspondence analyses by offering a statistical analysis of the findings. Likert scales are limited in that the distance between responses is not constant (the difference between “Very Important” and “Quite Important” may be different than “Quite Important” and “Important”). Ordinal regressions accommodate for this variable distance by examining odds-ratios [11]. They are the most appropriate way for modelling ordinal Likert data.

**3.4.3 Factor Analysis.** EFA groups large numbers of items into latent factors, based upon the response profiles. EFA is used when there are no pre-existing factors, allowing for free association. Skills associated with the same factor are assumed to share common variance, indicating a latent grouping structure.

### 3.5 Data Validity

The requirement for correspondence analysis is there is no missing data, data independence, and the scales used are identical for all responses. This is ensured through the survey design. People could not respond more than once and could only sort skills into a single category, missing data was filtered from analysis, and all scales were identical for all participants.

For ordinal regression, data must be ordered with more than two possible outcomes (for example, a five-point Likert scale), and the proportional odds assumption must be satisfied. To test this assumption the Brant-Wald test was used. Non-significant values suggest the data satisfies this assumption, which was met in all reported instances.

For EFA, Kaiser-Meyer-Olkin (KMO) test and Bartlett’s test of Sphericity can be used. KMO is used as an indicator for data factorability, our KMO score was .85, passing the threshold of .80 [15]. Bartlett’s test assesses association between variables, with a significant result indicating correlations in the data. Bartlett’s test was 1234.50,  $p = <.001$ , indicating suitability.

## 4 RESULTS

This section is structured using the research questions. We provide a description of the analysis, followed by an extraction of key findings and their relevance to the research question and hypothesis.

Table 1. Values of the factor loadings of the final EFA, ordered by factor eigenvalue and item loading strength.

	Computational Thinking	Social	Workplace	Independence	Visionary
Analytical Thinking	0.74				
Problem Solving	0.70				
Methodical Thinking	0.63				
Critical Thinking	0.63				
Organisational Initiative	0.53				
Communication	0.50				
Interpersonal		0.75			
Conflict Management		0.67			
Ethical Thinking		0.64			
Decision Making		0.51			
Handling Pressure		0.43			
Responsibility			0.57		
Flexibility			0.53		
Willingness to Learn			0.53		
Teamwork			0.53		
Motivation			0.46		
Autonomy			0.44	0.77	
Adaptability				0.54	
Creativity					0.75
Innovative					0.67

#### 4.1 Testing Hypothesis One: Differences in Valued Skills for CS and Psychology Alumni

Hypothesis 1 stated, “Computer Science undergraduate alumni will prioritise different groupings of soft skills to Psychology alumni, representing two different populations of perceived soft skill importance.”

First, an exploratory factor analysis was conducted to reduce the 23 skills into smaller sets of similarly rated skillsets. EFA factors with larger eigenvalues indicate greater importance in response to the survey question “how important are the following skills to your job role”. Parallel analysis was used to identify the optimal factor count, with eigenvalues above one used as a threshold. The optimal number of factors was five, using weighted least squares and varimax rotation.

Leadership and time management cross-loaded against multiple factors, indicating minimal unique contribution and were removed. The factor analysis was rerun resulting in no cross-loading. Final factor scores are presented in Table 1, with loadings under .4 suppressed. Fit indices of RMSEA = .05 [CI = .02, .07], TLI = .93 were acceptable [21]. Correlations between factors ranged from -.09 to .12 confirming factor distinctiveness.

It can be useful to name factors for easier interpretation and the identified factors were named as: Computational Thinking (eigenvalue,  $\lambda = 6.67$ ), Social ( $\lambda = 2.31$ ), Workplace ( $\lambda = 1.47$ ), Independence ( $\lambda = 1.33$ ) and Visionary ( $\lambda = 1.18$ ). It is noted these names recognise core items, but as the skills are factored according to their importance, they could easily be called “Most Important”, “Important”, “Less than Important”, “Unimportant”, and “Not Necessary”, but as this is reflected in the eigenvalues, more

What's in an undergraduate Computer Science degree? Alumni perceptions about soft skills in careers 11

descriptive names were chosen. The factorisation provides an answer to the first research question of what skills are considered important to the undergraduate alumni group as a whole.

The second research question asked how the two alumni groups differ in their reported importance of skills, and to answer this using the reduced set of skillsets, a linear regression model can be used to examine differences between the two alumni groups, examining the interaction between department and skillsets, with a random effect of participant to account for individual differences in participant responses. Pairwise comparisons were carried out, which returned no significant relationships. An  $R^2_{conditional}$  of .09 provided minimal variance, and model diagnostics suggested a poor model fit. Subsequently, the factor analysis provides little contribution to the question of how the two groups differ in their ratings.

Looking more broadly at the individual skills, a joint correspondence analysis of importance ratings for each skill between groups was used. Ratings of “Not important” and “Less than Important” were combined to remove zeroes in the data improving coherence. Correspondence analysis plots ordinal data in reduced dimensional space and as shown in Figure 4, the first dimension explains 84.99% variance and can be interpreted as importance (and dimension two as department with 10.19% variance). To note are the barycentres, which are the cumulative average of the two groups (denoted by the larger points on the figure), from this we can see that overall psychology alumni were more intense in their ratings suggesting they rated the skills higher on average.

As the first dimension explains such high variance, the analysis can be reduced to a single dimension, shown in Figure 5. Psychology alumni are centred on zero allowing the CS responses to vary, reflecting perceived importance. To test for statistically significant differences between the two alumni populations, ordinal logistic regressions were used. For brevity, only skills with significant differences are reported.

**4.1.1 Ethics.** For differences in ethics, CS alumni are associated with odds .72 times lower of rating ethics highly compared to psychology alumni,  $\beta = -1.27$ ,  $t = -3.81$ ,  $p < .001$ . Figure 6 shows the score probabilities for each level of the rating, with all skills.

**4.1.2 Time Management.** CS alumni had .61 lower odds of ranking time management skills as highly as psychologist alumni,  $\beta = -.95$ ,  $t = -2.73$ ,  $p = .006$ . Differences are seen in Figure 6, with psychology alumni ranking time management highly compared to CS alumni.

**4.1.3 Team Skills.** CS alumni had 1.90 times higher odds of rating team skills more importantly than psychology alumni with a large difference in the “Very Important” ratings by CS alumni as shown by Figure 6,  $\beta = .64$ ,  $t = 1.95$ ,  $p = .051$ . CS alumni were more likely to report team skills as “Very Important” compared to psychology alumni.

**4.1.4 Interpersonal.** For interpersonal skills, CS alumni had .58 times lower odds of ranking highly compared to psychology alumni,  $\beta = -.88$ ,  $t = -2.51$ ,  $p = .012$ . Both demonstrated low probabilities of reporting low importance as shown in Figure 6.

**4.1.5 Communication.** Like interpersonal skills, CS alumni were associated with .60 times lower odds of ranking communication as high as psychology alumni,  $\beta = -.92$ ,  $t = -2.43$ ,  $p = .015$ . Figure 6 shows the difference in the rating scale with greater probability of CS alumni ranking communication lower than “Very Important”.



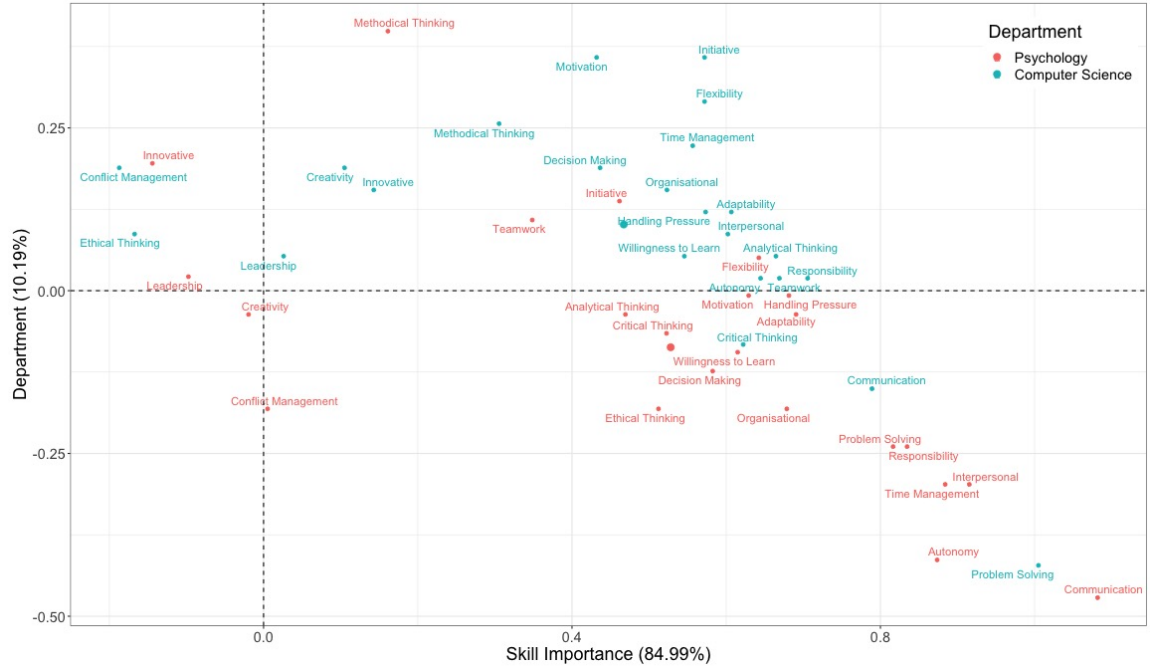


Fig. 4. Correspondence Analysis plotting skill importance between alumni groups. Dimension one can be interpreted as importance and explains 84.99% of variance, with the second dimension interpreted as differed in department explaining only 10.19% of variance.

RQ<sub>2</sub> contributes to answering H<sub>1</sub>, that the two alumni populations differ in their prioritised skills. Using EFA, no significant differences were seen, but on a more granular level, CS alumni valued teamwork higher than psychology alumni, who valued communication, interpersonal skills, time management, and ethics more than CS alumni.

RQ<sub>3</sub> explored how undergraduate alumni recalled being introduced to soft skills. Participants were asked to sort the soft skills they possessed into being developed through education, employment, personal interests, or that they did not possess the skill. The latter were removed from the analysis, as they provided little descriptive power, and removal had minimal effect on the analysis.

A joint correspondence analysis identified the sources that alumni associated skills originating from. Figure 7 reports items with an associative strength greater than .25 for easier representation. The first dimension explains 68.28% of the variance and can be interpreted as a continuum of Education to Occupation. The y-axis provided 29.49% variance representing a spectrum of Personal Development, with higher values indicating more self-led development. Figure 8 identifies angles of less than 30° and greater than 150° and the strengths associated with each skill. Only skills with a strength greater than .25 are included to aid clarity. Skills with a small positive angle (less than 30) are positively aligned with the source, and items that are linked through obtuse angles of 150-180° are negatively associated with a source.

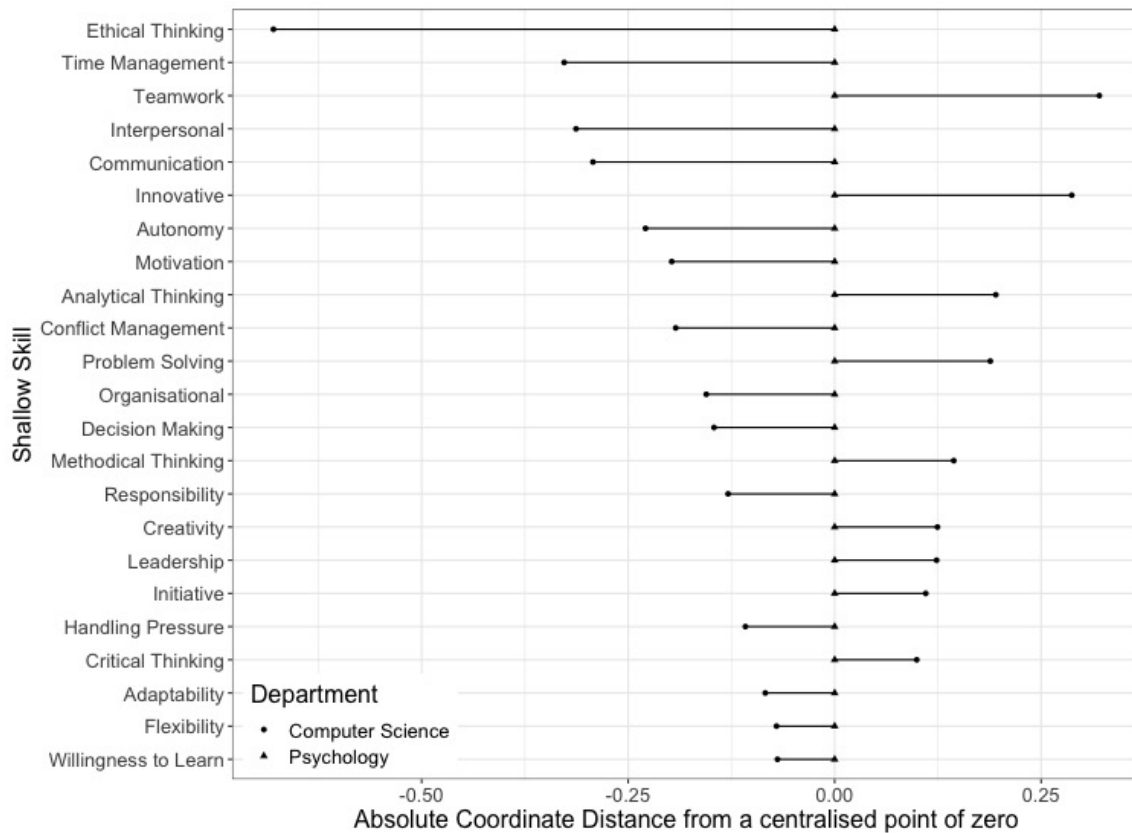


Fig. 5. A representation of the absolute difference in distance and direction between alumni for each skill. As dimension one explains so much variance, omitting dimension two still provides a strong representation of the skills reduced to a single dimension.

Using the associative angles and strengths of the correspondence analysis, four key interpretations can be made:

1. Both groups viewed analytical and methodical thinking as originating from university. Differences were seen with CS alumni associated problem solving with university, and psychology alumni associated critical thinking and ethics with education. These are the primary skills that graduates develop in university prior to employment, representing the strengths of these degrees.
2. For CS alumni, a willingness to learn, ethics, and motivation were associated with personal development. A willingness to learn was weakly associated with personal development for psychology alumni, and creativity being more strongly associated.
3. Skills such as leadership, conflict management, and decision making were associated with employment by both alumni groups. CS alumni felt team working was linked with employment more so than other sources. Psychology alumni associated flexibility and adaptability with employment more strongly than CS alumni.

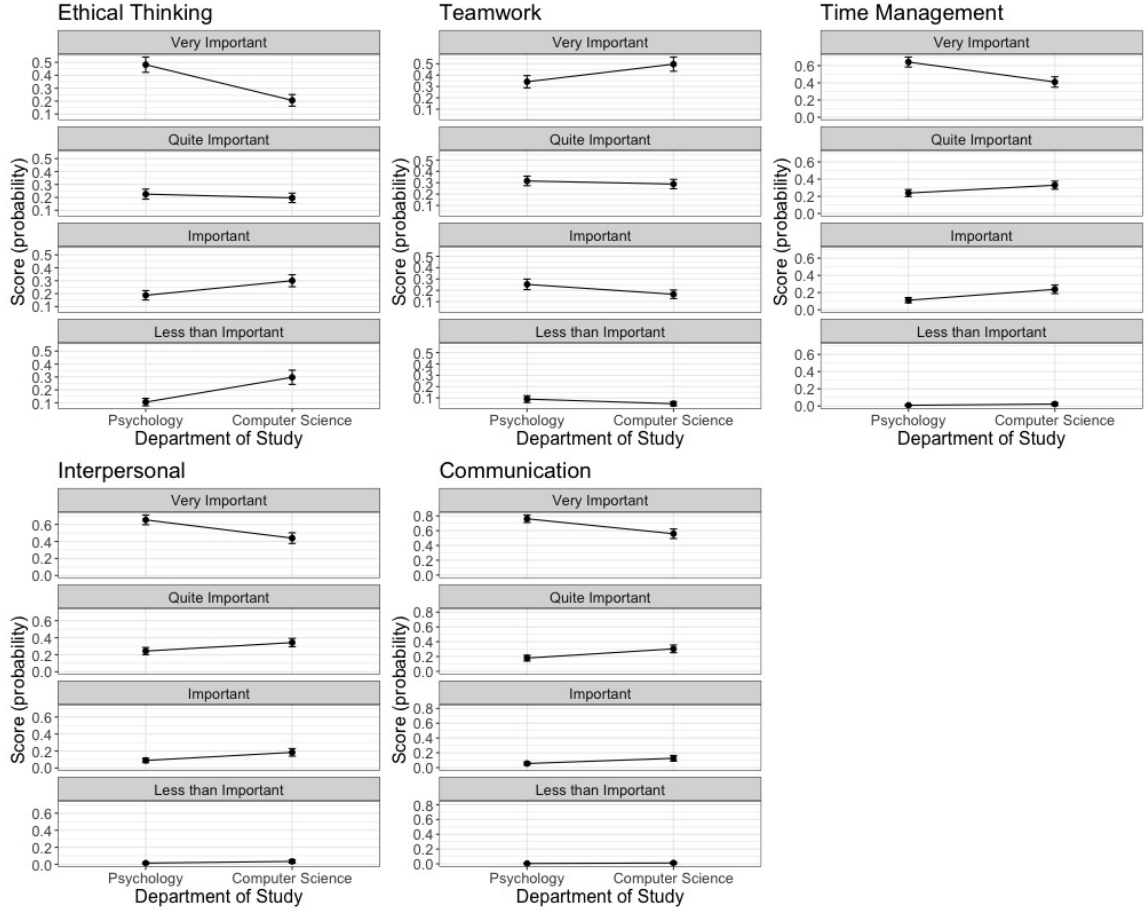


Fig. 6. Plot of the fitted probabilities for all skills with significant differences between the two departments. Each individual plot indicates the differences between alumni responses for each rating level

- the barycentre (the average of all skills combined by alumni population) for CS alumni is positioned closer to employment as an overall source of skill development, whereas psychology alumni associated more skills with their education.

The fourth research question explored how the technical skills possessed by alumni related to the perceived importance of soft skills in employment. When asked for the technical skills taught and used, participants responded with free text answers that were manually categorised. Twenty technical skill types were derived, seen in Figure 9.

Frequently mentioned skills by CS alumni were programming, followed by declarative knowledge (a broad skill category encompassing concepts like computational theory, history, or skills such as UML). and then data management. In comparison, psychology alumni referenced statistics and data analysis (which included tool use such as SPSS, R, NVivo), along with technical writing which included policy documents. Psychology alumni also used research skills, like designing studies, surveys, and interviews. The skills mentioned by CS

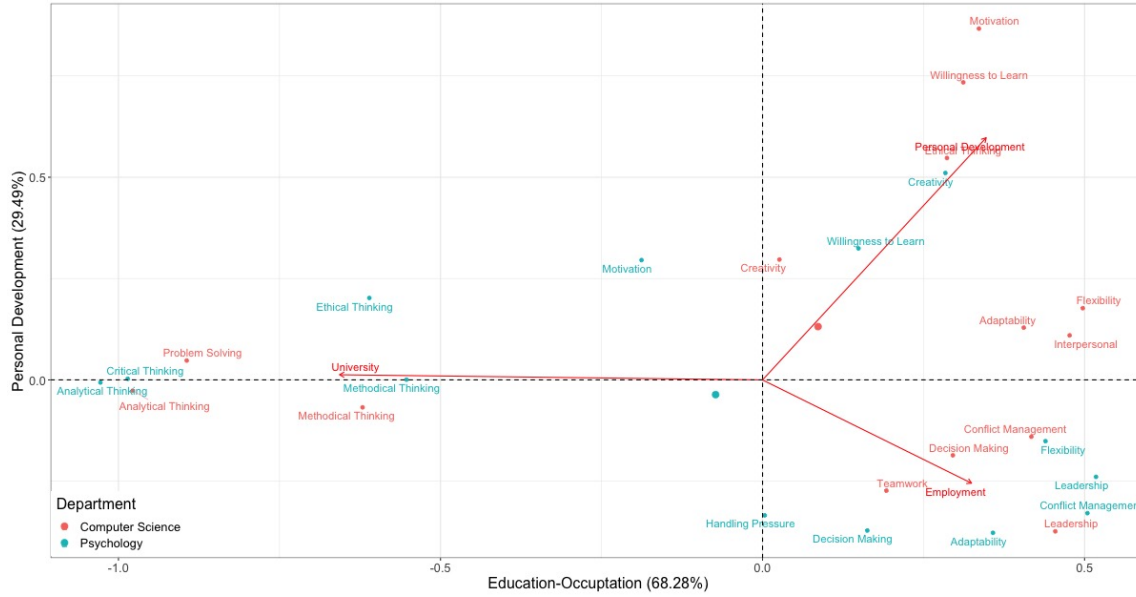


Fig. 7. Joint correspondence analysis of skills and their perceived originating source as either university, through hobbies or through work, split by department. Only items with an associative strength greater than .25 are included in the plot for clarity. Arrows indicate the angles of association.

alumni align with skills needed in software engineering roles, which is unsurprising as over 91% reported software engineering relevant roles.

Alumni also listed soft skills in the free-text responses here, which are shown in Figure 10. The most common skills mentioned were problem solving, communication, and teamwork. In the survey flow, participants were presented with this question prior to completing the soft skill ranking task, indicating that some participants considered these as technical skills, perhaps highlighting their considered importance.

#### 4.2 Addressing Hypothesis Two: Skills Valued for Security-Related Work

Hypothesis two, “Computer Science undergraduate alumni involved in security-related work will prioritise different soft skills than those who are less involved in security”, was answered through asking how CS undergraduate alumni differ in their use of security, as well as differences between levels of security use.

A joint correspondence analysis was carried out with the question, “Do you use security in your role?” which had a five-point Likert response of “None at all”, “A little”, “A moderate amount”, “A lot”, and “A great deal”. “None at all” and “A little” were combined into one factor called “Little to none” due to minimal responses to “None at all”. Responses were doubled and Figure 11 provides the correspondence analysis plot split into skill-isolated plots.

The first dimension explains 70.54% variance and can be interpreted as skill importance, with the second dimension explaining 17.37% variance and interpreted as rating intensity, corresponding to either high or low scores (“Very Important” or “Less than Important”). Higher scores on dimension one were more positively

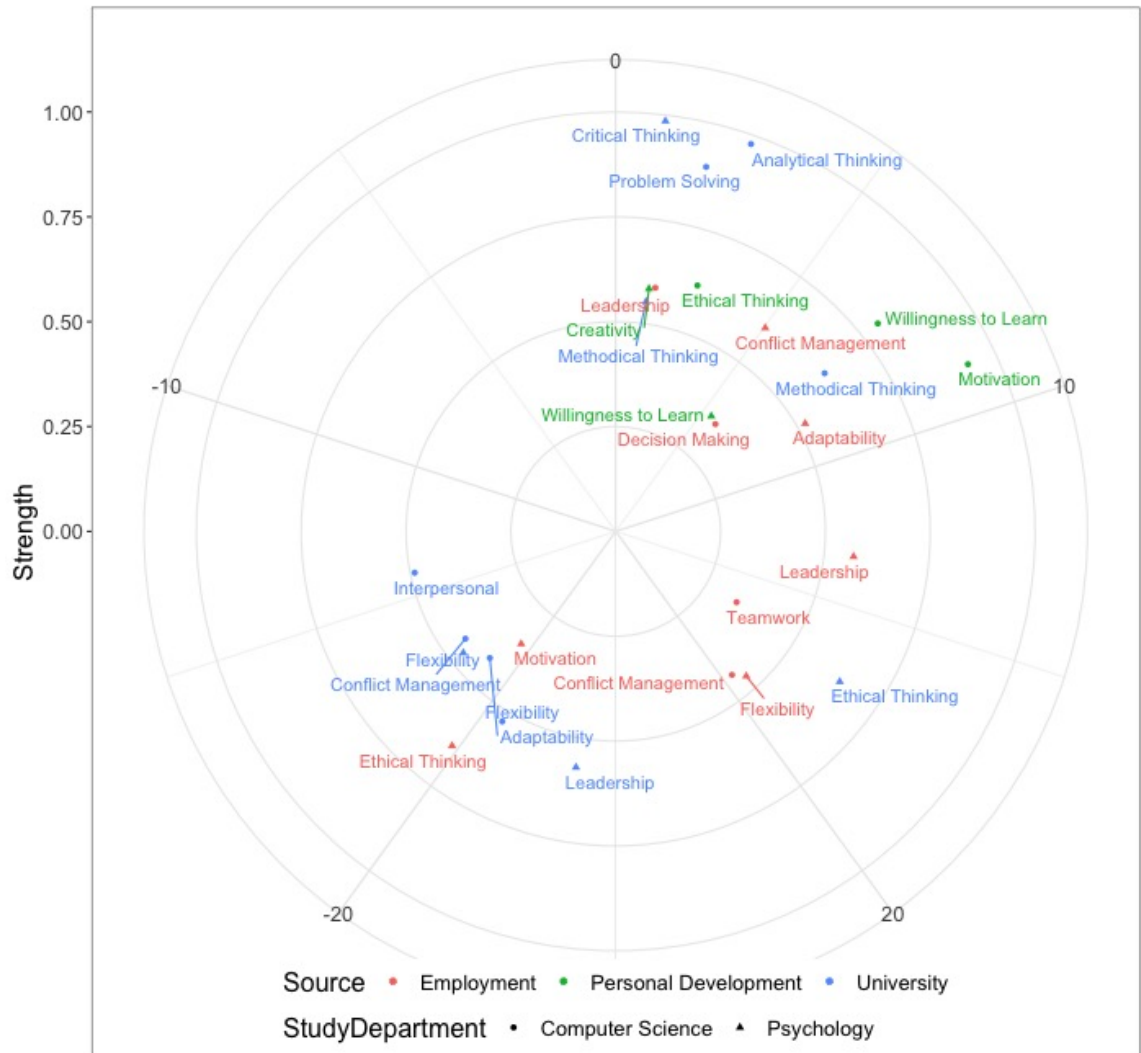


Fig. 8. Associations of skills for each source of development. Items plotted closer towards the zero axis (the top) have a smaller associative angle, indicating a stronger association, and items further from the centre are more strongly associated. A strong cluster of positively connected skills are seen in the arc between 0 and 10, indicating their strong association to their respective sources. Skills seen to be closely associated with a source are within the righthand side of the positive angles, and skills that are not associated with a particular source have negative associations and are seen towards the lefthand side.

associated with security usage. As self-reported security usage increased, skills of communication, critical thinking, decision-making, flexibility, initiative, innovation, organisation, and problem solving were rated more highly.

Manuscript submitted to ACM

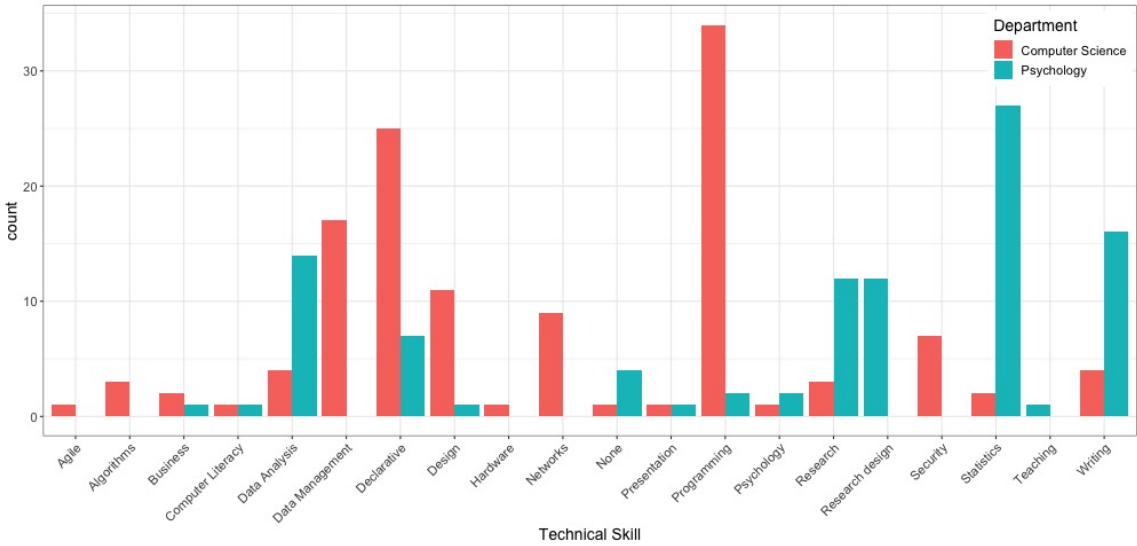


Fig. 9. Frequencies of each reported skill type per participant.

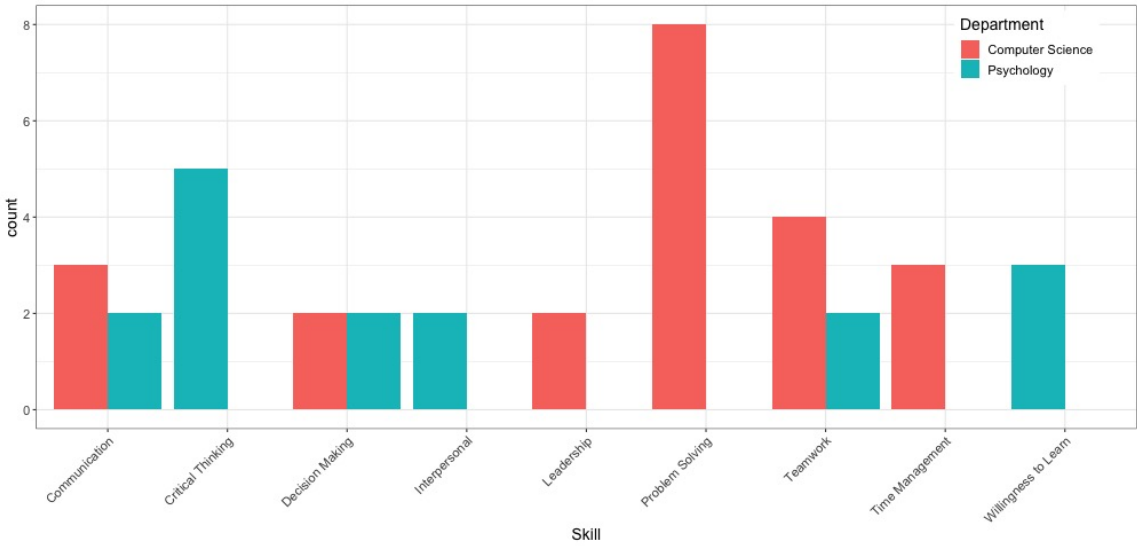


Fig. 10. Soft skills mentioned within the answers of which technical skills were taught and used in employment. Items mentioned once only are omitted.

Participants were asked for further security usage detail providing context. Those who reported using a great deal of security included cybersecurity industries, security team leads, handling financial data, security consulting, or working for national defence contractors. Those who used a lot of security reported data confidentiality, fixing insecure code, adhering to security recommendations, and secure sectors such as aerospace. Those reporting a moderate amount of security mentioned data confidentiality, physical building

security, preventing common insecure code issues (e.g., OWASP top 10), and secure system design. Those who used security a little mentioned GDPR adherence, personnel vetting, integrating software with automated security scans, and basic input validations. This reflects a general trend in the self-reported security levels with higher levels of security aligning with tasks or domains that can be reasonably considered as more security-conscious (e.g., national defence versus GDPR adherence).

Due to the relatively small dataset of those asked about security ( $n = 59$ ), ordinal models of specific skills contained high multicollinearity levels and so further analysis was not conducted.

The responses answer H<sub>2</sub> in that security work will require specific skills in comparison to lower security-related work, with an indication that communication, problem solving, critical thinking, decision-making, flexibility, initiative, innovation, and organisation were required more than others.

### 4.3 Results Summary

For RQ<sub>1</sub>, and how undergraduate alumni understand the importance of soft skills for employment, factor analysis was used. Five skillsets were found, grouped by importance. Skillsets included Computational Thinking (analytical thinking, problem solving, methodical skills, critical thinking, organisational skills, and initiative), Social (communication, interpersonal, conflict management, ethical thinking, and decision making), Workplace (handling pressure, responsibility, flexibility, willingness to learn, teamwork, and motivation), Independence (autonomy and adaptability), and Visionary (innovation and creativity) skills.

For RQ<sub>2</sub>, how do the two alumni groups differ in the soft skills deemed important for work, this was answered using a joint correspondence analysis. The responses were reduced to a single dimension explaining 84.99% variance. An ordinal regression was carried out on skills demonstrating the greatest differences. CS alumni had 72% lower odds of rating ethics highly compared to psychology alumni, 61% lower odds of rating time management skills highly, 58% lower odds for interpersonal skills, and 60% lower odds for communication. Inversely, CS alumni had 90% higher odds of reporting team skills as important compared to psychology alumni.

For RQ<sub>3</sub>, how do undergraduate alumni recall being introduced to soft skills? A joint correspondence analysis of where skills were developed was used. CS alumni associated analytical and methodical thinking, and problem solving with university; a willingness to learn, motivation and ethical thinking through personal development, and leadership, conflict management, teamwork, and decision making through employment. Teamwork and interpersonal skills were developed under more occupational settings than university.

For RQ<sub>4</sub>, how do the technical skills possessed by alumni relate to the perceived importance of soft skills in employment? An answer was provided through tagging of technical skills identified as being taught in education and used in employment. Key skills identified were various programming languages, declarative knowledge, and data management.

For RQ<sub>5</sub>, how CS alumni differ in their security usage and how the required soft skills differ, a correspondence analysis indicates that significant skills for security include communication, flexibility, initiative, innovation, organisation, and problem solving. The data was not appropriate for further statistical analysis due to insufficient data quantities.

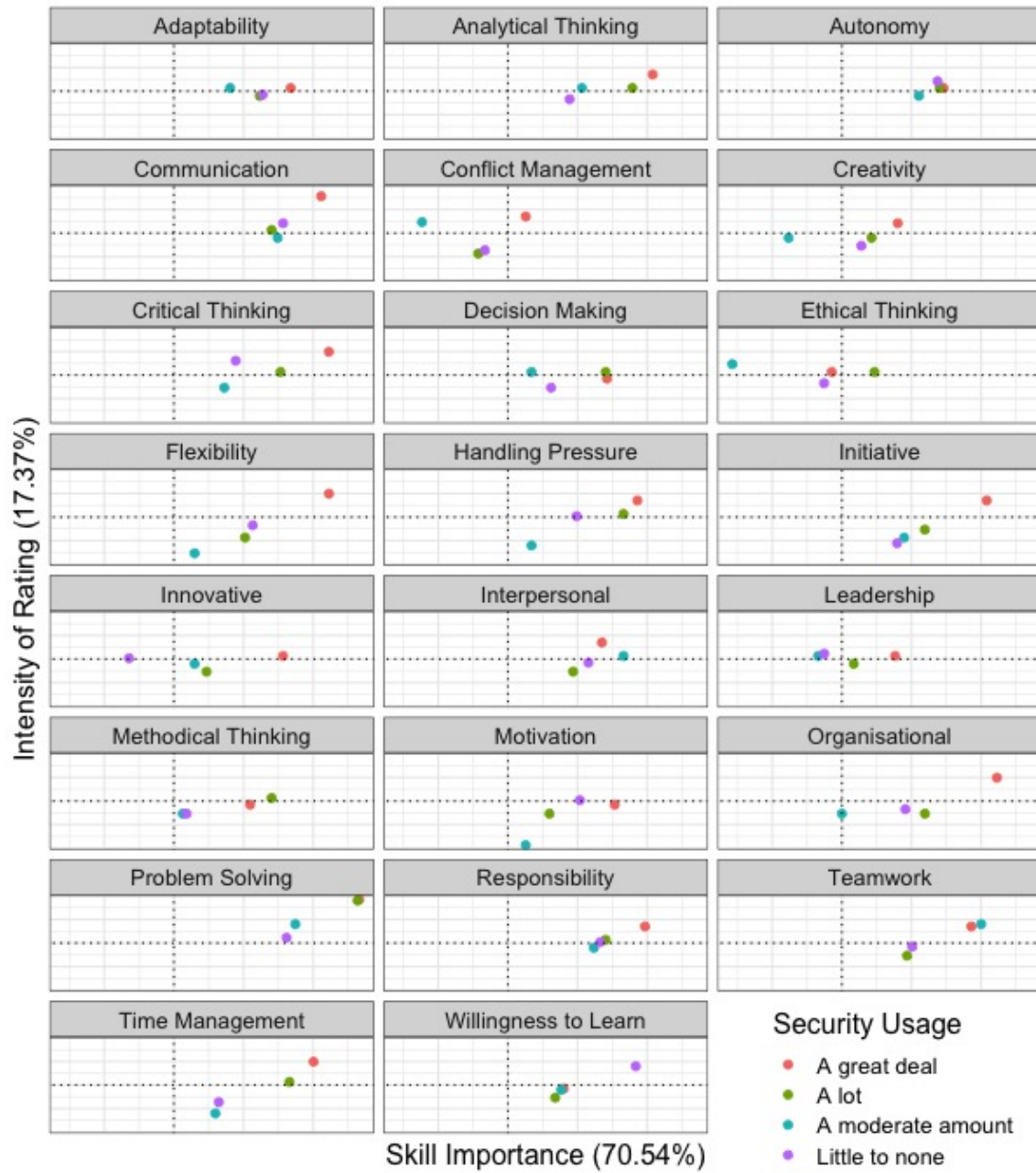


Fig. 11. A split-by-skill representation of each skill rated for CS alumni's security usage in employment. Items closer to the centre or each other indicate minimal differences in the way skills are used around security. Dimension one provides over 70% of variance and so items located further along the x-axis indicated greater perceived importance.



## 5 DISCUSSION

In this preregistered study, we report on the perceptions of soft skills for CS alumni, the sources of skill development, and compared their valued skills against a sample of psychology alumni. We include an exploration of the soft skills valued in security-related work. A multi-pronged analysis found computational thinking, communication, and social skills were valued by both CS and psychology alumni, indicating the universal nature of these skills. For CS alumni, the skills of problem solving, analytical, and methodical thinking were developed during education, whereas more workplace-specific skills, like leadership and teamwork, were associated with employment, and curiosity and motivation were linked with more personal interests and hobbies. In contrast to psychology alumni CS alumni valued team skills more, indicating the importance of team working in the industry. For skills around security work, key skills included communication, problem solving, critical thinking, decision-making, flexibility, initiative, innovation, and organisation.

We explored alumni perceptions regarding the skills they value in employment and where these skills originate from. Importantly, the intentional collection of their perceptions emphasises the role of their individual experiences, and how this may contrast with reality. The experiential nature of soft skills in education is arguably more valuable to understand than what a curriculum offers as it relates to the beliefs alumni hold about their own abilities. Student perceptions are updated over the course of their education, and soft skills increase in their value [12]. Despite this change, students' skills are different to industry expectations [54, 60].

CS alumni valued skills such as problem solving and communication, with less distinction between other skills. Problem solving, communication, and teamwork are key skills that have been previously identified as important for software engineering [3, 13, 45, 59, 62], indicating an alignment between industry and CS alumni. This is unsurprising, as most of the sample (91%) reported having had at least one software engineering-related job position since graduation. This is indicative that the skills desired by software engineering employers are the skills that are being used by their employees.

### 5.1 Sources of Valued Soft Skills

If undergraduate programmes are to prepare students for the world beyond education, it is important they introduce and develop the skills valued in the workplace. Figure 12 shows the top five important skills for CS alumni and their origins as shown. It is the same as Figure 7 but only shows CS alumni top five reported skills for enhanced clarity. It is clear these skills are associated with different sources. Problem solving and analytical thinking are developed during their university education. These skills represent the strengths of the CS undergraduate course, as it gives students opportunities to develop the skills considered important for the careers chosen by the alumni.

Figure 12 shows that problem solving is closely associated with their education, likely due to the influence of programming which is seen to develop problem solving skills [1]. This suggests that students are well prepared for the technical aspects of software engineering. CS courses have previously been seen to teach problem solving well [57], and this is a soft skill that can be considered as being integral to a CS undergraduate programme. The skill of communication had a small positive association with education over the other two

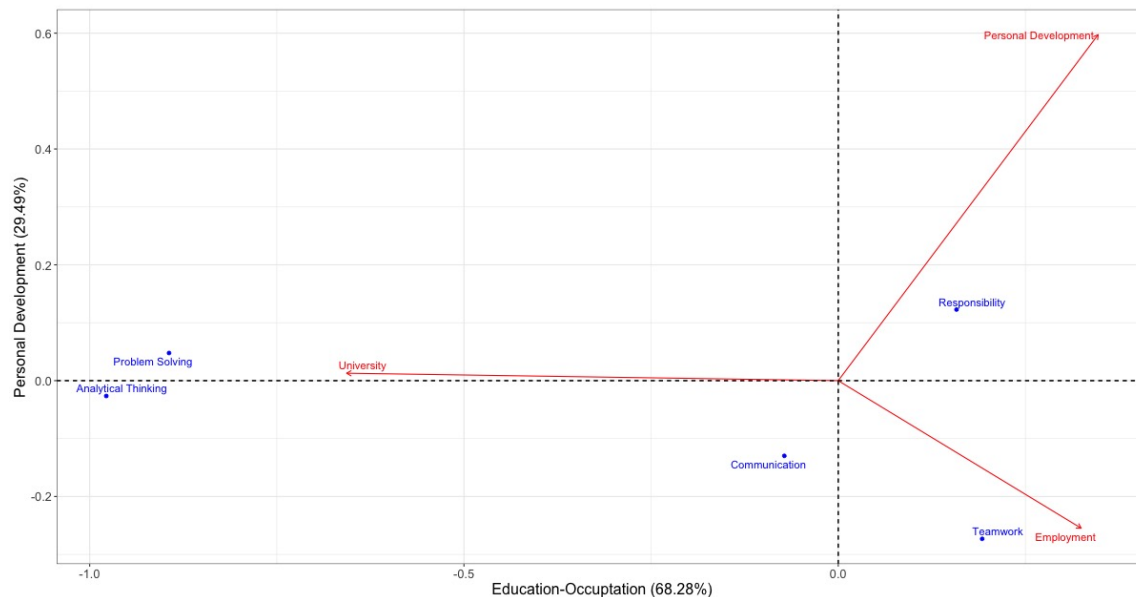


Fig. 12. The source of development for the top five skills reported by CS alumni.

sources. Responsibility was poorly associated with any specific source. It is noted these weak associations do not imply these skills are not developed anywhere, but rather that there is no consensus on its source.

Teamwork was most associated with employment, which possibly contrasts the expectation from educators who emphasise it in their teachings. Teamwork has long been recognised as a core skill for CS students [44], and subsequently included in most curricula. Despite this, alumni perceive teamwork being developed during employment. This may be a result of participants perceiving employment to have a stronger role over education in understanding how team work is carried out, and so was chosen as only one source could be selected. More concerningly, it may reflect participants not considering education to have provided them with effective team working skills. It is important that the CS undergraduate education continues to foster the development of team working skills.

One method of encouraging team-working skills is through project-based learning [55], allowing students to develop their team working skills in a controlled environment with opportunities to explore without serious ramifications (such as impacts on business success). This benefits not only the students who enter the workplace with the necessary skills, but employers also as new employees require less training.

Ethical thinking was ranked low in the valued skills of CS alumni, indicating either a general agreement over its influence in CS alumni careers, or that ethics in education is different to the way ethics is handled in industry. In recent years, ethics has become more of a component in education [20, 56], and so our findings may not reflect the perception of recent graduates as we surveyed a broad range of graduation years. One way to improve the value of ethics is to ensure that teaching materials possess ecological validity, such as incorporating realistic ethical scenarios or information into teaching materials. Producing these more concrete examples can improve how ethics is perceived by students [32]. If students are entering careers

with minimal understanding of thinking ethically, they risk causing harm to others. We emphasise that our findings are not reflective of an idea that CS alumni do not value ethics, but that in relation to other skills, it is not as valued.

Our findings show the CS undergraduate course provides students with opportunities to develop their problem solving and analytical thinking skills, which are reported as being some of the most valuable soft skills. Problem solving and analytical thinking are some of the programme's strengths with problem solving being highly valued. Teamwork is an important skill in the workplace but fails to be considered adequately taught during education. It is important that students are given experience in working in teams or projects that translate into their future work. We note that ethics, despite its place on the curriculum in many courses, failed to be valued to the same degree as other skills. Further work is needed to understand why, and to ensure students are being made aware of the value of ethics.

## 5.2 Comparisons With Psychology Alumni

To ensure our findings were not considered in a vacuum, we provide a contrast between CS and psychology alumni, which offers context to the perceptions of CS alumni. Taking the stance that psychology alumni have greater career variance compared to CS alumni, it highlights how a CS education prepares graduates for their likely employment in software engineering. We see a potential universal nature for some of the more important skills, such as communication and problem solving, which are noted in other technical industries including science [25], and business [16, 40].

When exploring differences between the two groups, we found CS alumni valued ethics, time management, interpersonal skills, and communication lower than psychology alumni. In contrast, they valued teamwork more than psychology alumni. These skills reflect the differences in perceptions between CS and psychology alumni. It is necessary to interpret these findings alongside the importance ratings, and to this end, a simplified representation of these skills is shown in Figure 13. The distance from the barycentre and plot coordinates represents the differences between the two groups. For time management, interpersonal skills, and communication they show similar distances from their respective barycentres in the same direction, indicating differences primarily in the intensity of the ratings rather than one group valuing the skills more.

The findings suggest whilst ethics is of no major importance to psychology alumni, it is undervalued by CS alumni and represented the biggest difference between the two groups. The comparison against psychology alumni suggests CS alumni may be entering employment with a reduced sense of importance around ethics. Incorporating ethics into software engineering is challenging, with many organisations lacking ethical guidelines, placing responsibility on individuals [47], and on entering the industry, recent graduates may feel that what ethics they did learn is not applicable to their work.

Teamwork was seen as being valued more by CS alumni, considered more important than the average skill rating. As over 91% of CS alumni reported a software engineering related role, this indicates these roles require team working, which aligns with many software teams using team-based project management such as agile working [7]. The value of teamwork for CS alumni, with the finding that this was associated with having been developed in employment rather than education, suggests that improved methods of including teamwork in undergraduate programmes should be considered.

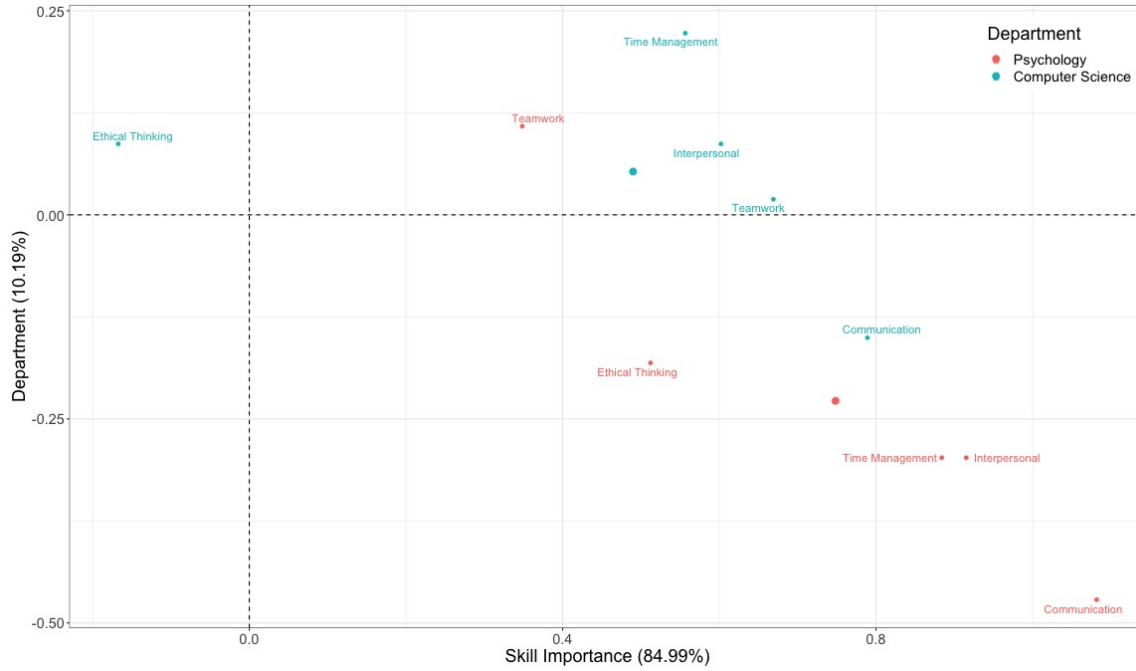


Fig. 13. A plot highlighting the skills that had significant differences between the two alumni populations.

### 5.3 Security-related Skills

With increased security use, skills of communication, critical thinking, decision making, flexibility, initiative, innovation, organisation, and problem solving were considered important. If there were no link between certain skills and security usage, the correspondence analysis would have shown very tightly clustered points for each skill. This was not the case, and the skills that show deviation from the main cluster as security usage increased are required more often when working in security.

Cybersecurity professionals have previously reported communication and teamwork as being valuable soft skills for their profession [39], as well as being curious [5]. Along with our own findings that problem solving is associated with undergraduate education, skills of organisation and flexibility were more associated with employment than education, and communication, initiative, and innovation were not seen to load strongly against any one source, suggest that the skills needed for cybersecurity are variable in their development source. Curricula that incorporate aspects of security should consider how to incorporate these skills into their content.

We suggest that modules and courses focused on cybersecurity should ensure they provide opportunities to develop these key skills in a security context. Whilst many of the skills are not linked with a definitive source, educators should ensure they are introducing and providing opportunities to develop these skills, boosting the prominence of their presence in cybersecurity teaching materials.

#### 5.4 Threats to Validity

This research targeted CS alumni from the same institution, limiting generalisability to other programmes. However, the spread of graduation years encompasses multiple iterations of the CS programme, spanning significant developments in CS as a discipline. As such, the findings speak to a more generalised experience of skills across time. This indicates the skills reported are consistently desired, and that their importance is not a new phenomenon.

The use of ordinal scales limits the analysis options due to subjectivity of individuals' interpretation of distance between points on the scale. Despite limitations of ordinal scales, the research was designed to better understand which skills are of interest to CS alumni, reflecting the use of soft skills that CS alumni possess and the role of university in their development. The use of ordinal logistic regression and correspondence analysis were selected as the most appropriate methods for analysing these data types.

The ordinal regression models, used for making comparisons between the valued importance of the two populations, offered minimal variance explanation, which was anticipated due to the fluid definitions of soft skills. Soft skills as a defined concept are weak, but we controlled for this by giving participants a defined list of what each skill referred to. This list was validated prior to data collection, as described in Skill definition validation. A second issue of soft skill predictive power is more complex, and our findings show whilst associations exist between soft skills and the variables examined, they are not particularly strong. This was expected and the motivation behind this research was to identify the key skills to allow further psychological research to examine these areas more closely.

Finally, the findings are recognised as being of a Western-centric position, potentially limiting generalisability to the global population. One of the aims of this research was to link the skills used in employment to a specific population of alumni from an undergraduate programme. In doing so, we provide strong links between a particular educational programme and the skills that are delivered and then used. Our materials, processes, and analyses are all available for use at (<https://osf.io/s52r7/>), allowing future work to use the same paradigm with different populations.

#### 5.5 Further Work

This research uncovered relevant soft skills providing a base to build future research upon. With an understanding of the skills considered relevant for CS alumni, and security use, further work can explore these skills by examining the links between more measurable psychological concepts that contribute to soft skills. As previously acknowledged, soft skills and their definitions are ephemeral and broad, whereas psychological ideas of cognition and social interaction are more easily measured and supported by theory, such as social identity being responsible for interpersonal skills, or cognitive heuristics research supporting problem solving and analytical thinking approaches.

### 6 CONCLUSION

We carried out a preregistered survey study with alumni of both CS and psychology departments to understand the skills considered important for employment, and where these skills originate from. By focusing on the alumni association, as opposed to industry or job title, our findings speak to the way the CS undergraduate programme influences skill perceptions.

The most prioritised skills could be grouped into computational thinking, social skills and communication through a factor analysis based off data from both CS and psychology alumni, suggesting these skills are of a universal requirement in employment. University education was identified by CS alumni as responsible for developing computational thinking skills, such as analytical and methodical thinking, and problem solving. Employment developed skills of leadership and teamwork, and curiosity and motivation were associated with personal development instead. For CS alumni working in security, key skills were communication, organisation, and problem solving.

The research implications extend towards improving undergraduate programmes. We highlight the skills valued by CS alumni in their employment and emphasise where these skills were developed. Key skills are those of communication and problem solving, which are valued in employment, and for those working in security. Educators who wish to improve CS graduate employability should ensure their modules cover the key skills in a multitude of ways, allowing students to develop and experience these skills prior to employment.

## **ACKNOWLEDGEMENTS**

We thank Dharini Balasubramaniam for her invaluable comments in improving our manuscript.

## REFERENCES

- [1] Friday Joseph Agbo, Samuel T. Yigzaw, Ismaila Temitayo Sanusi, Solomon Sunday Oyelere, and Alem Habte Mare. 2021. Examining Theoretical and Pedagogical Foundations of Computational Thinking in the Context of Higher Education. In *2021 IEEE Frontiers in Education Conference (FIE)*. 1–8. <https://doi.org/10.1109/FIE49875.2021.9637405>
- [2] Faheem Ahmed, Luiz Fernando Capretz, Salah Bouktif, and Piers Campbell. 2015. Soft Skills and Software Development: A Reflection from Software Industry. *International Journal of Information Processing and Management* 4, 3 (July 2015), 171–191. <https://doi.org/10.48550/arXiv.1507.06873> arXiv:1507.06873
- [3] Faheem Ahmed, Luiz Fernando Capretz, and P. Campbell. 2012. Evaluating the Demand for Soft Skills in Software Development. *IT Professional* 14, 1 (Jan. 2012), 44–49. <https://doi.org/10.1109/MITP.2012.7>
- [4] Christina Andersson and Doina Logofatu. 2018. Using Cultural Heterogeneity to Improve Soft Skills in Engineering and Computer Science Education. In *2018 IEEE Global Engineering Education Conference (EDUCON)*. 191–195. <https://doi.org/10.1109/EDUCON.2018.8363227>
- [5] Miriam E. Armstrong, Keith S. Jones, Akbar Siami Namin, and David C. Newton. 2020. Knowledge, Skills, and Abilities for Specialized Curricula in Cyber Defense: Results from Interviews with Cyber Professionals. *ACM Transactions on Computing Education* 20, 4 (Nov. 2020), 29:1–29:25. <https://doi.org/10.1145/3421254>
- [6] Association of Graduate Careers Advisory. 2022. What Can I Do with a Computer Science Degree? — Prospects.Ac.Uk. <https://www.prospects.ac.uk/careers-advice/what-can-i-do-with-my-degree/computer-science>.
- [7] Corey Baham and Rudy Hirschheim. 2021. Issues, Challenges, and a Proposed Theoretical Core of Agile Software Development Research. *Information Systems Journal* 32, 1 (2021), 103–129. <https://doi.org/10.1111/isj.12336>
- [8] K. V. A. Balaji and P. Somashekar. 2009. A Comparative Study of Soft Skills Among Engineers. *IUP Journal of Soft Skills* 3, 3/4 (Sept. 2009), 50–57.
- [9] Randy Bancino and Claire Zevalkink. 2007. Soft Skills: The New Curriculum for Hard-Core Technical Professionals. *Techniques: Connecting Education and Careers (J1)* 82, 5 (May 2007), 20–22.
- [10] Yuriy Brun, Tian Lin, Jessie Elise Somerville, Elisha M. Myers, and Natalie C. Ebner. 2023. Blindspots in Python and Java APIs Result in Vulnerable Code. *ACM Transactions on Software Engineering and Methodology* (April 2023). <https://doi.org/10.1145/3571850>
- [11] Paul-Christian Bürkner and Matti Vuorre. 2019. Ordinal Regression Models in Psychology: A Tutorial. *Advances in Methods and Practices in Psychological Science* 2, 1 (March 2019), 77–101. <https://doi.org/10.1177/2515245918823199>
- [12] Manuel Caeiro-Rodríguez, Mario Manso-Vázquez, Fernando A. Mikic-Fonte, Martín Llamas-Nistal, Manuel J. Fernández-Iglesias, Hariklia Tsalapatas, Olivier Heidmann, Carlos Vaz De Carvalho, Triinu Jesmin, Jaanus Terasmaa, and Lene Tolstrup Sørensen. 2021. Teaching Soft Skills in Engineering Education: An European Perspective. *IEEE Access* 9 (2021), 29222–29242. <https://doi.org/10.1109/ACCESS.2021.3059516>
- [13] André Calitz, Margaret Cullen, and Jean Greyling. 2015. *South African Alumni Perceptions of the Industry ICT Skills Requirements*.
- [14] Luiz Fernando Capretz and Faheem Ahmed. 2018. A Call to Promote Soft Skills in Software Engineering. *Psychology and Cognitive Sciences - Open Journal* 4, 1 (Aug. 2018), e1–e3. <https://doi.org/10.17140/PCSOJ-4-e011> arXiv:1901.01819
- [15] Barbara A. Cerny and Henry F. Kaiser. 1977. A Study Of A Measure Of Sampling Adequacy For Factor-Analytic Correlation Matrices. *Multivariate Behavioral Research* 12, 1 (Jan. 1977), 43–47. [https://doi.org/10.1207/s15327906mbr1201\\_3](https://doi.org/10.1207/s15327906mbr1201_3)
- [16] A. K. Chattoraj and Saleha Shabnam. 2015. Importance of Soft Skill in Business. *Anusandhanika* 7, 2 (July 2015), 105–110.
- [17] Giuseppe Destefanis, Marco Ortu, Steve Counsell, Stephen Swift, Roberto Tonelli, and Michele Marchesi. 2017. On the Randomness and Seasonality of Affective Metrics for Software Development. In *Proceedings of the Symposium on Applied Computing (SAC '17)*. Association for Computing Machinery, New York, NY, USA, 1266–1271. <https://doi.org/10.1145/3019612.3019786>
- [18] Rosanne English and Alan Hayes. 2022. Towards Integrated Graduate Skills for UK Computing Science Students. In *Proceedings of the 2022 Conference on United Kingdom & Ireland Computing Education Research (UKICER '22)*. Association for Computing Machinery, New York, NY, USA, 1–7. <https://doi.org/10.1145/3555009.3555018>
- [19] Silvia Fareri, Nicola Melluso, Filippo Chiarello, and Gualtiero Fantoni. 2021. SkillNER: Mining and Mapping Soft Skills from Any Text. *Expert Systems with Applications* 184 (Dec. 2021), 115544. <https://doi.org/10.1016/j.eswa.2021.115544> arXiv:2101.11431
- [20] Casey Fiesler, Natalie Garrett, and Nathan Beard. 2020. What Do We Teach When We Teach Tech Ethics? A Syllabi Analysis. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 289–295. <https://doi.org/10.1145/3328778.3366825>
- [21] W. Holmes Finch. 2020. Using Fit Statistic Differences to Determine the Optimal Number of Factors to Retain in an Exploratory Factor Analysis. *Educational and Psychological Measurement* 80, 2 (April 2020), 217–241. <https://doi.org/10.1177/0013164420938888>

## What's in an undergraduate Computer Science degree? Alumni perceptions about soft skills in careers 27

- //doi.org/10.1177/0013164419865769
- [22] Raluca Florea and Viktoria Stray. 2018. Software Tester, We Want to Hire You! An Analysis of the Demand for Soft Skills. In *Agile Processes in Software Engineering and Extreme Programming (Lecture Notes in Business Information Processing)*, Juan Garbajosa, Xiaofeng Wang, and Ademar Aguiar (Eds.). Springer International Publishing, Cham, 54–67. [https://doi.org/10.1007/978-3-319-91602-6\\_4](https://doi.org/10.1007/978-3-319-91602-6_4)
  - [23] Kevin P. Gallagher, Kate M. Kaiser, Judith C. Simon, Cynthia M. Beath, and Tim Goles. 2010. The Requisite Variety of Skills for IT Professionals. *Commun. ACM* 53, 6 (June 2010), 144–148. <https://doi.org/10.1145/1743546.1743584>
  - [24] Vahid Garousi, Gorkem Giray, Eray Tuzun, Catayatal Catal, and Michael Felderer. 2020. Closing the Gap Between Software Engineering Education and Industrial Needs. *IEEE Software* 37, 2 (March 2020), 68–77. <https://doi.org/10.1109/MS.2018.2880823>
  - [25] Anaïs Gibert, Wade C. Tozer, and Mark Westoby. 2017. Teamwork, Soft Skills, and Research Training. *Trends in Ecology & Evolution* 32, 2 (Feb. 2017), 81–84. <https://doi.org/10.1016/j.tree.2016.11.004>
  - [26] Sharlett Gillard. 2009. Soft Skills and Technical Expertise of Effective Project Managers. *Issue in Informing Science and Information Technology* 6, 1 (2009), 723–729.
  - [27] C. Matt Graham and Yonggang Lu. 2022. Skills Expectations in Cybersecurity: Semantic Network Analysis of Job Advertisements. *Journal of Computer Information Systems* 0, 0 (Sept. 2022), 1–13. <https://doi.org/10.1080/08874417.2022.2115954>
  - [28] Michael Greenacre. 2017. *Correspondence Analysis in Practice*. CRC Press.
  - [29] Wouter Groeneveld, Brett A. Becker, and Joost Vennekens. 2020. Soft Skills: What Do Computing Program Syllabi Reveal About Non-Technical Expectations of Undergraduate Students?. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '20)*. Association for Computing Machinery, New York, NY, USA, 287–293. <https://doi.org/10.1145/3341525.3387396>
  - [30] Wouter Groeneveld, Hans Jacobs, Joost Vennekens, and Kris Aerts. 2020. Non-Cognitive Abilities of Exceptional Software Engineers: A Delphi Study. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 1096–1102. <https://doi.org/10.1145/3328778.3366811>
  - [31] Wouter Groeneveld, Joost Vennekens, and Kris Aerts. 2019. Software Engineering Education Beyond the Technical: A Systematic Literature Review. <https://doi.org/10.48550/arXiv.1910.09865> arXiv:cs/1910.09865
  - [32] Barbara J. Grosz, David Gray Grant, Kate Vredenburg, Jeff Behrends, Lily Hu, Alison Simmons, and Jim Waldo. 2019. Embedded EthiCS: Integrating Ethics across CS Education. *Commun. ACM* 62, 8 (July 2019), 54–61. <https://doi.org/10.1145/3330794>
  - [33] Joseph Hallett, Nikhil Patnaik, Benjamin Shreeve, and Awais Rashid. 2021. “Do This! Do That!, And Nothing Will Happen” Do Specifications Lead to Securely Stored Passwords?. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 486–498. <https://doi.org/10.1109/ICSE43902.2021.00053>
  - [34] Julie M Haney and Wayne G Lutters. 2017. Skills and Characteristics of Successful Cybersecurity Advocates. In *Thirteenth Symposium on Usable Privacy and Security*. USENIX Association, Santa Clara, CA, USA.
  - [35] Jim Ivins, Brian R Von Konsky, David Cooper, and Michael Robey. 2006. Software Engineers and Engineering: A Survey of Undergraduate Preconceptions. In *Proceedings. Frontiers in Education. 36th Annual Conference*. 6–11. <https://doi.org/10.1109/FIE.2006.322364>
  - [36] Matthew Ivory, Miriam Sturdee, John Towse, Mark Levine, and Bashar Nuseibeh. 2023 (in review). Can You Hear the ROAR of Software Security? How Responsibility, Optimism And Risk Shape Developers' Security Perceptions. *Empirical Software Engineering* (2023 (in review)). <https://doi.org/10.31234/osf.io/pexvz>
  - [37] Matthew Ivory, John Towse, Miriam Sturdee, Mark Levine, and Bashar Nuseibeh. 2023. Recognizing the Known Unknowns; the Interaction Between Reflective Thinking and Optimism for Uncertainty Among Software Developer's Security Perceptions. *Technology, Mind, and Behavior* 4, 3: Winter 2023 (Dec. 2023). <https://doi.org/10.1037/tmb0000122>
  - [38] Jingdong Jia, Zupeng Chen, and Xiaoping Du. 2017. Understanding Soft Skills Requirements for Mobile Applications Developers. In *2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, Vol. 1. 108–115. <https://doi.org/10.1109/CSE-EUC.2017.29>
  - [39] Keith S. Jones, Akbar Siami Namin, and Miriam E. Armstrong. 2018. The Core Cyber-Defense Knowledge, Skills, and Abilities That Cybersecurity Students Should Learn in School: Results from Interviews with Cybersecurity Professionals. *ACM Transactions on Computing Education* 18, 3 (Aug. 2018), 11:1–11:12. <https://doi.org/10.1145/3152893>
  - [40] Michael Jones, Cindi Baldi, Carl Phillips, and Avinash Waikar. 2016. The Hard Truth about Soft Skills: What Recruiters Look for in Business Graduates. *College Student Journal* 50, 3 (Sept. 2016), 422–429.
  - [41] Damien Joseph, Soon Ang, Roger H. L. Chang, and Sandra A. Slaughter. 2010. Practical Intelligence in IT: Assessing Soft Skills of IT Professionals. *Commun. ACM* 53, 2 (Feb. 2010), 149–154. <https://doi.org/10.1145/1646353.1646391>



- [42] Antti Knutas, Timo Hynninen, and Maija Hujala. 2021. To Get Good Student Ratings Should You Only Teach Programming Courses? Investigation and Implications of Student Evaluations of Teaching in a Software Engineering Context. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. 253–260. <https://doi.org/10.1109/ICSE-SEET52601.2021.00035>
- [43] Janet Liebenberg, Magda Huisman, and Elsa Mentz. 2014. Knowledge and Skills Requirements for Software Developer Students. *International Journal of Social, Behavioral, Educational, Economic, Business and Industrial Engineering* 8, 8 (2014), 6.
- [44] Robert Lingard and Shan Barkataki. 2011. Teaching Teamwork in Engineering and Computer Science. In *2011 Frontiers in Education Conference (FIE)*. F1C–1–F1C–5. <https://doi.org/10.1109/FIE.2011.6143000>
- [45] Gerardo Maturro. 2013. Soft Skills in Software Engineering: A Study of Its Demand by Software Companies in Uruguay. In *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. 133–136. <https://doi.org/10.1109/CHASE.2013.6614749>
- [46] Gerardo Maturro, Florencia Raschetti, and Carina Fontán. 2019. A Systematic Mapping Study on Soft Skills in Software Engineering. *Journal of Universal Computer Science* 25, 1 (2019), 26.
- [47] Anna Mitchell, Dharini Balasubramaniam, and Jade Fletcher. 2022. Incorporating Ethics in Software Engineering: Challenges and Opportunities. In *2022 29th Asia-Pacific Software Engineering Conference (APSEC)*. 90–98. <https://doi.org/10.1109/APSEC57359.2022.00021>
- [48] João Eduardo Montandon, Cristiano Politowski, Luciana Lourdes Silva, Marco Tulio Valente, Fabio Petrillo, and Yann-Gaël Guéhéneuc. 2021. What Skills Do IT Companies Look for in New Developers? A Study with Stack Overflow Jobs. *Information and Software Technology* 129 (Jan. 2021), 106429. <https://doi.org/10.1016/j.infsof.2020.106429>
- [49] E. Mtsweni, Tertia Hörne, and J. Poll. 2016. Soft Skills for Software Project Team Members. (2016). <https://doi.org/10.7763/IJCTE.2016.V8.1035>
- [50] Sarah Nadi, Stefan Krüger, Mira Mezini, and Eric Bodden. 2016. Jumping through Hoops: Why Do Java Developers Struggle with Cryptography APIs?. In *Proceedings of the 38th International Conference on Software Engineering (ICSE '16)*. Association for Computing Machinery, New York, NY, USA, 935–946. <https://doi.org/10.1145/2884781.2884790>
- [51] Alena Naiakshina, Anastasia Danilova, Eva Gerlitz, Emanuel von Zezschwitz, and Matthew Smith. 2019. "If You Want, I Can Store the Encrypted Password": A Password-Storage Field Study with Freelance Developers. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–12. <https://doi.org/10.1145/3290605.3300370>
- [52] NCSC. 2021. More Master's Degrees at UK Universities Recognised by Cyber Security Experts. <https://www.ncsc.gov.uk/news/more-university-degrees-certified>.
- [53] Daniela Seabra Oliveira, Tian Lin, Muhammad Sajidur Rahman, Rad Akefirad, Donovan Ellis, Eliany Perez, Rahul Bobhate, Lois A. DeLong, Justin Cappos, and Yuriy Brun. 2018. {API} Blindspots: Why Experienced Developers Write Vulnerable Code. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*. 315–328.
- [54] Mohamad Osmani, Vishanth Weerakkody, Nitham M. Hindi, Rajab Al-Esmail, Tillal Eldabi, Kawaljeet Kapoor, and Zahir Irani. 2015. Identifying the Trends and Impact of Graduate Attributes on Employability: A Literature Review. *Tertiary Education and Management* 21, 4 (Dec. 2015), 367–379. <https://doi.org/10.1080/13583883.2015.1114139>
- [55] Stuart Palmer and Wayne Hall. 2011. An Evaluation of a Project-Based Learning Initiative in Engineering Education. *European Journal of Engineering Education* 36, 4 (Aug. 2011), 357–365. <https://doi.org/10.1080/03043797.2011.593095>
- [56] Rob Reich, Mehran Sahami, Jeremy M. Weinstein, and Hilary Cohen. 2020. Teaching Computer Ethics: A Deeply Multidisciplinary Approach. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 296–302. <https://doi.org/10.1145/3328778.3366951>
- [57] Shima Salehi, Karen D. Wang, Ruqayya Toorawa, and Carl Wieman. 2020. Can Majoring in Computer Science Improve General Problem-solving Skills?. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 156–161. <https://doi.org/10.1145/3328778.3366808>
- [58] Karen Stamms, Luona Lin, and Peggy Christidis. 2016. Datapoint: What Do People Do with Their Psychology Degrees? *Monitor on Psychology* 47, 6 (June 2016), 12.
- [59] Matt Stevens and Richard Norman. 2016. Industry Expectations of Soft Skills in IT Graduates: A Regional Survey. In *Proceedings of the Australasian Computer Science Week Multiconference (ACSW '16)*. Association for Computing Machinery, New York, NY, USA, 1–9. <https://doi.org/10.1145/2843043.2843068>
- [60] Chiara Succi and Magali Canovi. 2020. Soft Skills to Enhance Graduate Employability: Comparing Students and Employers' Perceptions. *Studies in Higher Education* 45, 9 (Sept. 2020), 1834–1847. <https://doi.org/10.1080/03075079.2019.1585420>

- [61] Lori L. Sussman. 2021. Exploring the Value of Non-Technical Knowledge, Skills, and Abilities (KSAs) to Cybersecurity Hiring Managers. *Journal of Higher Education Theory and Practice* 21, 6 (2021), 99–117.
- [62] Catherine Watson and Kelly Blincoe. 2017. Attitudes Towards Software Engineering Education in the New Zealand Industry. (2017).
- [63] Jen-Her Wu, Yi-Cheng Chen, and Jack Chang. 2007. Critical IS Professional Activities and Skills/Knowledge: A Perspective of IS Managers. *Computers in Human Behavior* 23, 6 (Nov. 2007), 2945–2965. <https://doi.org/10.1016/j.chb.2006.08.008>
- [64] Shundan Xiao, Jim Witschey, and Emerson Murphy-Hill. 2014. Social Influences on Secure Development Tool Adoption: Why Security Tools Spread. In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing*. Association for Computing Machinery, Baltimore, MD, USA, 1095–1106. <https://doi.org/10.1145/2531602.2531722>

#### 4.1 Statement of Continuous Thesis Summary

*“Organisations would be well served by diversifying the soft skills of their software developers. This would provide richer talent and viewpoints to help them tackle the inherent complexity of software construction.” - Ahmed et al., 2012*

This chapter seeks to describe and understand soft skill perceptions among graduates from the CS and Psychology departments. It shows that CS graduates value problem solving, communication, and teamwork skills. These skills are not homogeneous in origin; problem solving is positively associated with their university education, whereas teamwork is associated more strongly with the workplace, and communication has no strong associations with any specific source. Soft skills valued for security work cannot be statistically tested due to data quantity, but responsibility and organisational skills are reported as valuable over the overall graduate population’s perceptions. Using the psychology graduates as a contrasting sample, CS graduates rate ethics as less valuable than the psychologists, along with skills of time management, interpersonal skills, and communication. Conversely, CS graduates value teamwork significantly more than psychology graduates.

Graduate perceptions indicate that social and cognitive skills are valuable for software engineering roles, as many CS graduates reported working within software engineering-related domains. This supports previous findings where similar skills are seen to be valued for software engineering (Ahmed et al., 2012b; Calitz et al., 2015; Matturro, 2013; Stevens & Norman, 2016; Watson & Blincoe, 2017). In investigating skill development sources, it is identified that education teaches computational skills well, such as problem solving and analytical thinking, but that more socially-aligned skills are associated more with employment, indicating that despite efforts from educators, students do not recognise the opportunities to develop these skills.

As an answer to the first research question, “What non-technical skills (or soft skills) are valued within computer science and software engineering?”, the three primary skills of

problem solving, communication, and teamwork are offered - this is fully answered in Chapter 9. How they are developed is more nuanced, with the evidence suggesting that undergraduate programmes must reflect on fully integrating communication and teamwork into their modules and have students recognise the opportunities to improve these skills before seeking employment.

This chapter also plays a foundational role in shaping the conceptual and empirical framing of soft skills that underpins subsequent chapters. While later chapters draw from different participant samples (e.g., freelance developers), this initial study offers a generalisable baseline by illustrating key soft skill constructs and their associated domains of development. This chapter set out the initial framework from which the rest of the thesis develops.

In the next chapter, I present a two-study paper exploring the embedded soft skills in curricula leveraging staff perceptions and curricula content. Through multi-site interviews with core module leaders, staff perceptions about the most valued soft skills and how they are incorporated into modules are exposed, along with the staff perspective regarding student engagement with soft skills. The second study applies a content analysis approach to extract soft skills from publicly available course material.

#### ***4.1.1 Contribution to Thesis Argument and Forward Trajectory***

This chapter contributes to the overarching thesis, which explores whether theories of decision-making and social identity can explain developers' computing and security behaviours within software engineering. Specifically, it addresses the first research question by empirically identifying the non-technical (or "soft") skills recent computer science graduates perceive as essential in software development roles. The findings offer insight into how these skills are shaped by different developmental contexts, namely education versus employment, and highlight a potential misalignment between academic preparation and industry expectations.

More broadly, this chapter helps establish one of the contributions of the thesis: that the

psychological and interpersonal dimensions of software engineering work are often unevenly supported across educational and professional contexts. By foregrounding this misalignment, the chapter positions the well-known idea of soft skills as central to the wider interdisciplinary argument of the thesis, that software engineering practice cannot be fully understood without serious engagement with underlying psychological constructs.

The next chapter builds directly upon this chapter by shifting focus from graduate perceptions to educator perspectives and curricular content, providing a deeper institutional and pedagogical context for understanding how soft skills may be embedded in university CS programmes.

## **5 Everything but Programming; Investigating Academics' Perceptions of Embedded Soft Skills in Computer Science Undergraduate Education**

Ivory, M., Towse, J., Sturdee, M., Levine, M., & Nuseibeh, B. (*in review*). Everything but Programming; Investigating Academics' Perceptions of Embedded Soft Skills in Computer Science Undergraduate Education. *Transactions on Computing Education*. <https://doi.org/10.31234/osf.io/y7guj>

# Everything but Programming; Investigating Academics' Perceptions of Embedded Soft Skills in Computer Science Undergraduate Education

MATTHEW IVORY, Lancaster University, Great Britain

JOHN TOWSE, Lancaster University, Great Britain

MIRIAM STURDEE, University of St Andrews, Scotland

MARK LEVINE, Lancaster University, Great Britain

BASHAR NUSEIBEH, Lero, Republic of Ireland and Open University, United Kingdom

Student employability is a key goal of a computer science undergraduate education. A soft skills gap has previously been reported between employer requirements and the skills graduates offer, suggesting that educators are inadequately preparing students for their future careers. It is important to identify the links between educators and the materials they claim to teach as it offers insight into how non-technical aspects of software engineering are promoted. We report on two studies where we first explore the staff perceptions of embedded soft skills in five computer science undergraduate courses, before identifying soft skill presence in curricula across eight UK universities. A multi-site interview with educators identified core skills of *critical thinking*, *communication*, and *teamwork* being included in curricula for student employability. Staff believe students experience a temporal delay between being introduced to skills and *actually* valuing them. In the second study, we mined publicly-available course and module information, and then analysed non-technical skill references. Soft skills are commonly found in proximity to other soft skills, suggesting they are taught or assessed together. *Software engineering* was seen to be closely linked to *teamwork* and *communication*, emphasising it is taught as a social enterprise. Taking these two studies together, educators show a close alignment to curricula, and the skills valued by higher education institutions reflect the skills valued in software engineering industries, suggesting the skill gap is the result of student misconceptions.

CCS Concepts: • **Security and privacy** → **Social aspects of security and privacy**; • **Human-centered computing** → *Empirical studies in HCI*;

Additional Key Words and Phrases: soft skills, graduate employability, student misconceptions, curriculum analysis

---

Authors' addresses: Matthew Ivory, Lancaster University, Department of Psychology, Lancaster, Lancashire, LA1 4YW, Great Britain, matthew.ivory@lancaster.ac.uk; John Towse, Lancaster University, Department of Psychology, Lancaster, Lancashire, LA1 4YW, Great Britain, j.towse@lancaster.ac.uk; Miriam Sturdee, University of St Andrews, School of Computer Science, North Haugh, St Andrews, LA1 4YW, Scotland, ms535@st-andrews.ac.uk; Mark Levine, Lancaster University, Department of Psychology, Lancaster, Lancashire, LA1 4YW, Great Britain, mark.levine@lancaster.ac.uk; Bashar Nuseibeh, Lero, Ireland, Castletroy, Co. Limerick, LA1 4YW, Republic of Ireland, Open University, United Kingdom, Bashar.Nuseibeh@open.ac.uk.

---

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

© 2024 Association for Computing Machinery.

Manuscript submitted to ACM

**ACM Reference Format:**

Matthew Ivory, John Towse, Miriam Sturdee, Mark Levine, and Bashar Nuseibeh. 2025. Everything but Programming; Investigating Academics' Perceptions of Embedded Soft Skills in Computer Science Undergraduate Education. 1, 1 (July 2025), 31 pages. <https://doi.org/10.475/123.4>

**1 INTRODUCTION**

Graduate employability is a key outcome of an undergraduate education. Recent Computer Science (CS) graduates face intense competition when seeking employment, and in 2023, CS undergraduate applicants in the UK increased by nearly 10% [7], suggesting an ever-growing body of prospective employees vying for the same job positions. Consequently, students want their education to provide them with the key skills required to stand out from other job applicants. Students are reliant on staff to teach them the necessary skills [30], but student misconceptions can restrict their ability to fully engage with the teachings, leading to incorrect value being assigned to technical and non-technical skills. Following graduation, students are then perceived to not possess the soft skill competencies required by employers, resulting in a soft skills gap between education and employment [4]. From both the student and employer perspective, it is the educator who is responsible for developing these soft skills, and so an exploration of their perceptions is beneficial to further illuminate the gap, and surface important issues around soft skill development.

Soft skills are the domain-agnostic, non-technical skills that determine the success of technical skill application [34], and they are emphasised as being important to the health of software engineering [12]. Despite this, students' expectations do not typically accommodate soft skills, which leads them to undervalue educator efforts to nurture these skills. This results in the soft skills gap between employer expectations and what graduates offer [36]. The situation raises important questions that deserve answers. How do staff design and implement educational materials to reduce this soft skills gap? How do staff handle the tensions between what students want, and what they need (as determined by prospective employers)? In this empirical project, we report on two complementary studies, where we evidence how staff address and resolve this tension, as well as extract soft skills from curricula material to understand how skills are embedded across the educational pathway.

Employers, students, and educators comprise a stakeholder triad, each group invested in the soft skill development process but with different motivations. Employers are motivated to recruit graduates who offer the necessary skills, because they directly experience the consequences of employees not possessing the relevant skills [44]. Students are motivated to gain the necessary experience and expertise required to secure employment [54], and educators are motivated to teach the necessary soft skills because graduate employability is an important metric, and not least, because they want to ensure future generations of software engineers possess the skills required to make an impact. Out of the three stakeholder groups, only the student population is expected to actively change and develop expertise in the soft skills being taught. Employers expect this change to occur during education [47] because educational environments are suitably-placed to allow students to test and develop their own ideas and understandings of soft skills. Despite the different motivations, they are insufficient to close the skill gap. While students are those who actively need to develop these skills, educators are directly responsible for cultivating the necessary environment for soft skill development, and it is this that we are interested in: the educator perspective.



Educators are not unsupported when it comes to deciding which soft skills to include in their curriculum. The SE2014 (Software Engineering 2014 [6]) provides a framework for CS curricula design that includes key soft skills to be taught. SE2014 is built upon the Software Engineering Body of Knowledge (SWEBOK [8]), which represents the skills needed for software engineering. The fact that SE2014 is derived from SWEBOK suggests that a strong alignment between what employers want and what educators teach, but how staff interpret the guidelines and then implement them is not necessarily homogeneous, and it is this that we seek to understand.

Recognising that the closing of the soft skills gap is not a unilaterally-sided task, educators must work to present the necessary skills, but employers need to recognise the tension between student wants and student needs. To this end, we need to evidence the importance of soft skills, recognise the need for learning and training opportunities, and understand how to direct the triad of stakeholders towards a common position. Identifying the embedded skills is essential as it reflects the human-intensive nature of software engineering [12]. Our research was motivated to address the soft skills gap, by exploring the educator stakeholder perspective. We identified three preregistered research questions to guide the study:

1. How do academic staff understand how soft skills are taught in their core modules?
2. What importance do staff ascribe to certain soft skills, in comparison to other skills?
3. How do important soft skills align with the information available in online course and module information?

We answer these questions in this two-study project. The first study reports on a multi-site interview study regarding educators' perceptions on soft skills within their module and CS course. The second study deploys a content analysis to extract references to soft skills from publicly-available curriculum content. Taken together, the findings offer a unique perspective into how soft skills are embedded across CS teaching, and how students are perceived to receive these skills.

## 2 RELATED WORK

The employer-student-educator triad helps to contextualise the issues around the topic of soft skills in CS education. In this section, I address relevant literature on the perspectives held by prospective employers, students, and academic staff.

### 2.1 Future Employer Perceptions

Employers typically prioritise soft skills [50], in part because technical skills are perceived as easier to teach to new employees [40]. By reducing the on-boarding time for new employees, they can meaningfully contribute to the company quicker, which benefits the company as it frees up resources otherwise required for training. Despite this, graduates are reportedly approaching companies without the required soft skills making them a less attractive hire, and particular deficiencies are seen for soft skills such as communication and group working [31].

Employers explicitly request soft skills in job adverts [22], emphasising that these are expected from prospective employees. Employers typically display a homogeneous view on the relevant soft skills, just with small differences in the order of importance [4]. Two primary soft skills are teamwork and communication [2, 26, 35, 42, 45]. Collaborative skills are also seen to be tied into contemporary methods of working, such

as Agile approaches [49]. Problem solving is also reported as being essential for various roles [2, 21, 41]. These skills are repeatedly identified as being essential for software engineering.

Employers are seen to blame educators for the soft skills gap, suggesting that they are failing to make students aware of the importance of soft skills [36], and others emphasise the need for educators to ensure curricula include adequate soft skills [47]. This occurs despite the SE2014 being in place which suggests that the support for curriculum design is derived from software engineering practitioners. Despite this, Akdur [4] reported that graduates are perceived not to possess the required skills when entering employment.

## 2.2 Student and Graduate Perceptions

To maximise impact and effectiveness, students must buy-in to the relevance of soft skill development, as they are the only stakeholders who are required to develop these skills. Broadly speaking, students are seen to hold misconceptions about soft skills, and typically prefer to prioritise technical knowledge acquisition over soft skill development [33], which is particularly evident for students with technical career aspirations [32]. This misalignment of priority skills occurs despite evidence for collaborative skills improving software quality in student work [39].

Students also conflate enjoyment with value during tasks [37] and so if students have a negative experience with an activity or project that they associate with soft skills, they will attach a lower value to these soft skills, in favour of tasks and activities that they see as more enjoyable. This may feed into their conception that technical skills are more important [33], for example programming tasks can be typically autonomous alleviating the need for collaboration with others, and it can be satisfying to solve problems through code. Students are seen to value distinctly different skills than those reported as desirable by employers. Students focus more on networking and other skills that they perceive as key to securing a job [51], rather than developing the skills asked for by employers.

Gathering data from graduates can be beneficial as they represent a population of ex-students who have experienced the educational process. CS graduates recognise the value of soft skills in their careers, emphasising the value of social and cognitive skills [11, 56]. Valued skills are not all developed during education, with group-working skills reported as being developed in employment [34], highlighting that a CS education may not be as effective as intended. Graduates have expressed concerns over their preparedness for collaborating in employment [16], indicating they do not perceive educators to have successfully carried out their role. From a contrasting perspective, when asked about the development of professional skills (which includes critical thinking, ethics, and group working), graduates report that they possess skills needed for employment [15].

## 2.3 The Educator's Role

Students trust educators to teach them the skills needed for employment [30], and subsequently educators need to ensure that their courses provided opportunities for students to develop soft skills, ideally over an extended period to allow for a gradual development process [28]. Amidst suggestions from industry that educators need to be teaching these skills [48], it benefits everyone to understand what soft skills are being taught and how they are embedded within curricula.

Examinations of curricula and modules can expose the soft skills that are prioritised within computer science and software engineering, with a systematic review identifying self-reflection, conflict resolution,

communication, and teamwork as the most taught skills [27], which were reflected in a further examination of soft skill-specific modules, identifying teamwork, ethics, communication, and presentation skills [25]. Others identified ethics and analytical thinking styles as being valuable within curricula [18].

Educator perceptions influence student perceptions, and students converge towards staff over time, resulting in increased soft skill valuation [24], with late-stage students demonstrating closer alignment to their educators than early-stage students do [10]. In general, those who are further through their education align more closely with the views of educators [38]. These findings highlight that students do gradually develop a better representation of the role of soft skills, but it is still limited, and total agreement is not achieved.

### 3 STUDY 1

Study 1 is a multi-site interview study with 12 educators across five institutions. This study offers insight into staff perceptions of embedded soft skills within core undergraduate modules.

#### 3.1 Methodology

The study was preregistered on the Open Science Framework, lodging details on the study design, data collection strategies, and analytic plans. The preregistration can be accessed at [https://osf.io/9yt35/?view\\_only=d8a5ff2663714a11a2ea250a59bfef81](https://osf.io/9yt35/?view_only=d8a5ff2663714a11a2ea250a59bfef81). Data, analysis scripts, and supplementary materials are found at [https://osf.io/25ecz/?view\\_only=917e6de233484d59abaac243647c89a2](https://osf.io/25ecz/?view_only=917e6de233484d59abaac243647c89a2).

**3.1.1 Participants.** Interviews were conducted with 12 staff members from five universities in the UK N8 research group (Durham, Lancaster, Leeds, Liverpool, Manchester, Newcastle, Sheffield, and York). This university group was chosen as they are all research-intensive institutions in the North of England, highlighting a shared commitment to research excellence that feeds into teaching. Educators teaching core modules for the Computer Science (UCAS code G400) undergraduate course were invited to participate via email. Participants from five institutions responded. Four from institution A (11 contacted), two from B (13 contacted), two from C (11 contacted), three from D (17 contacted), and one from E (12 contacted). See Table 1 for further details. For anonymity, participants were assigned random identifiers denoting institution using letters A through E, and a number (e.g., A1, B3) to distinguish individuals. No interviewees were personally known to the interviewer.

The invite explained the research and purpose of the contact. Those who responded were sent further information and a link where they could schedule an interview. Three educators expressed interest but did not respond further. Four declined, stating their module did not include soft skills. All participants provided informed consent. The Faculty Ethics Committee approved the research.

**3.1.2 Methodology.** A semi-structured interview was deployed, and the interview guide is in Appendix A. Beforehand, the interviewer made notes regarding participants' modules, department information, and any information offered during email exchanges. Each interview started with the interviewer describing the project, allowing participants to ask questions, and confirming they consented to the interview being recorded for transcription.

The first question, "tell me about the undergraduate module(s) that you teach", was designed to engage participants in a narrative mindset and provided information to direct the interview further. Questions were

Table 1. Relevant participant demographics, including their position within the department and the delivery year of core modules.

Interviewee	Position	Module(s) Year
A1	Lecturer	1
A2	Lecturer	1
A3	Reader	2
A4	Professor	1
B1	Senior Lecturer	2
B2	Lecturer	2
C1	Senior Lecturer	1
C2	Senior Lecturer	1, 3
D1	Lecturer	1
D2	Senior Teaching Fellow	1
D3	Teaching Fellow	1
E1	Senior University Teacher	1

based on the interview guide and interviewer notes. The first author conducted all interviews via Microsoft Teams, which provided automatic transcriptions and were kept to one hour. Transcriptions were validated against the original recordings and were anonymised. Transcriptions were sent to interviewees for content approval and consent to transcripts being shared under open science practices. Approval was sought to ensure staff were happy with the anonymisation carried out. Minor changes were made following feedback, which only improved interviewee anonymisation.

**3.1.3 Analysis.** Data were subject to a thematic analysis following the five stages: code familiarisation, code generation, theme generation, theme review, and theme naming [9]. The first author familiarised themselves via the transcription validation process and multiple readings. Initial notes were made during readings, and preliminary codes were developed. This process was iterated until no new quotes were added and no additional codes were generated. This process ended when further reading generated no substantial changes. Codes were then reviewed, which included merging or splitting codes where necessary. Themes were then formed by grouping the codes. Themes were reviewed through authors' discussion and assessed for independence, coherency, and distinctiveness. Finally, themes were given relevant names. Transcripts were coded in Taguette, an open-source qualitative analysis tool [46]. The codebook and the tagged dataset are available in the OSF repository.

## 3.2 Results

Two key themes that contribute to answering the research questions were identified. The first theme of *Valued Soft Skills* relates to soft skills identified as components of teaching materials. The second theme of *Student Perceptions* relates to how educators understand the student experience. The quotes have been edited for clarity and coherence, but the original intent is maintained. The use of quotes is selective rather than exhaustive and represents staff perceptions.

**3.2.1 Valued Soft Skills.** The soft skills mentioned can be split into three skillsets: Cognitive, Social, and Professional. Cognitive skills relate to internal mental processes for reasoning and applying technical

knowledge. Social skills are those involved in interpersonal communication and are required to facilitate successful interactions with others. Professional skills are required for high performance and functioning in a work environment, in the context of this research, they are distinct from social and cognitive skills.

*Cognitive.* Educators emphasised the cognitive development opportunities that students are exposed to throughout the course. Many of these opportunities include tasks involving programming and other related activities, such as code comprehension or code writing. One exception to this general rule was educators deploying cognitive puzzles, such as sudoku, to teach logic-based reasoning. Removing the coding element ensures students develop the cognitive foundations needed for programming tasks. These puzzle-solving sessions preceded programming tasks and offered students opportunities to deploy the logical processes and programmatic approach required to program successfully.

*“The aim over a series of seven or eight [sessions] is to teach all the skills of programming without ever writing a line of code. That’s our overall aim of these workshops, and they’re all generally built around logic puzzles.” – A2*

Other opportunities for students to develop mental models needed for more complex tasks involved using low-level languages such as Assembly, which was used for its machine-readable syntax; by exposing students to programming languages with minimal abstraction, they can develop their internal representation of how programming works. Almost opposite to using non-programming tasks to teach logic, low-level languages can be used to teach about how programming works without requiring knowledge of complex functions. Doing so gives students the mental models to use when working with high-level languages that abstracts many low-level language behaviours.

*“They’re looking at code that they’ll never use again, but it’s giving them the mental foundation so that when they’re writing in a higher level language, they can refer back” – C2*

Students want to be taught fashionable languages, but educators stress the need to teach languages that offer pedagogical advantages (such as Assembly) because they help students develop essential mental representations and the necessary technical knowledge. Educators emphasise that understanding the concepts of programming are more valuable and language-agnostic than the language itself. Learning these less-popular languages ensures that students develop the mental representations necessary for programming.

*“A degree in computer science isn’t a vocational course on programming. Learning a programming language is not the object for the exercise. Understanding what programming is and how to solve the problem computationally, such that when you’re thrown into a new language and a new situation, you can get your bearings with reference to some broader concepts.” – E1*

By giving students the cognitive foundations required, students are expected to reach a point where they can translate a real-world problem into computational space, allowing for it to be solved through software code. This is a critical milestone as it ensures students can successfully engage in more complex software engineering tasks.

*“The purpose is to try and get them all to a level where they can think about a problem as a computer programme.” – D1*

Staff need to ensure students experience the opportunity to solve problems independently and own the responsibility for developing their understanding. Part of this means staff should be careful not to obstruct

students' learning opportunities by resolving issues that are or just above the level of where students should be.

*"One of the things I found really hard is to tell my teaching assistants to never solve the student's problems, Definitely not on their keyboard, unless it's something weird. But in general, just to say, What, the problem here? And to engage [students] in this process" – A4*

A problem solving mindset can be facilitated using projects that necessitate critical approaches. Projects often require a mix of both technical and non-technical competencies, which can be used to demonstrate to students the value of soft skills. Project-based learning offers opportunities for students to apply the more abstract understandings developed during puzzles or low-level language-based tasks with more relevance to software engineering.

*"[The] programming project is not an easy module, and it is made deliberately so that students learn how real world applications work in terms of testing and debugging. At an advanced level, C is not easy to work when you have to do a lot of memory allocation and deallocation and most of the students struggle with segmentation faults. They don't know where the code is going wrong. So it's basically approaching the problem solving skills and following good programming practice." – D3*

The progression of students' cognitive models of software engineering often begins with removing software nuances (using logic puzzles or low-level languages) before exposing students to increasingly complex situations. Once they possess the necessary mental models, they can perform tasks related to software engineering careers.

*"When you ask students to find the one line that has a bug on it in a 6,000 class Java programme, they have to be analytical, and we teach them the code reading skills to help them track down the source of the bug." – A3*

The two previous comments succinctly highlight what the end goal of a student's cognitive development is driving towards – the ability to effectively problem solve using a critical mindset. Critical thinking can be defined as "being able to critically review the information and determine its importance for informing decisions and being able to use evaluative and inferential reasoning to increase the likelihood of a desired outcome" [34], and this is what educators are encouraging students to develop. Through an incremental development process, students are directed towards these essential cognitive capabilities required for technical tasks such as programming.

*Social.* Students are encouraged to develop their communication and group working skills during their education. Communication includes aspects of speaking, reading, and writing. Group working skills focus on how communication is used in intragroup and intergroup contexts to complete shared goals. The pedagogical aim of social development is to help students communicate in professional and articulate ways. Peer discussions are valuable for developing communication skills and boosting students' understanding. By allowing multiple perspectives to be presented, students can challenge their preconceptions by reflecting on other perspectives. Discussions can be incorporated through software engineering relevant ways using asynchronous tools. The tools mentioned include institutional education systems (e.g., Moodle and Blackboard) and industry-specific tools like GitLab. Asynchronous tools allow educators to help facilitate discussions, and they provide features that promote more efficient working (such as notifications) and practical applications to future employment.

Everything but Programming; Investigating Academics' Perceptions of Embedded Soft Skills in Computer Science Undergraduate Education 9

*"We adopt Gitlab, but we use the bug trackers as generalised issue trackers. We tell them to use that issue tracker as a general task planning thing. So put deadlines and assign them to people. If you've got issues, put comments in there so notifications get shot round to people. Because that's the way it's going to be in a company."* – B2

Two skills mentioned together were reading and writing. They form a valuable part of a student's skillset, spanning technical and academic materials. These skills are typically not taught explicitly but are indirectly required to excel academically.

*"I think reading comprehension is a soft skill. I would say we don't teach it. We assume they have it and that's an interesting thing."* – C2

The ability to actively comprehend written materials is also an important aspect of student development, allowing them to explore information critically to further their technical skills. Reading skills can be facilitated by engaging students in various materials, including academic and technical information, and teaching them about relevant or trustworthy sources.

*"I show them there's quite a lot of reference material and if they want to know how a function works, they look up the definition of it. If they want to use a library function in C, it's really well defined exactly what the inputs are, what the output is, exactly what it does. If you give it the wrong inputs and if you can read that and understand it, then you can use that function and be happy."* – D1

Students must understand how to utilise communication in both intragroup and intergroup contexts. Successful groups consist of individuals who share common values or goals, which ties them into a group structure; where these values are not shared, it can result in group breakdown that impedes goal-directed behaviour. Students must learn about how groupwork influences different tasks and situations, and one skill that comes with groupwork is the flexibility of approach. A2 facilitates this by giving students various tasks, and by experiencing group successes and difficulties, they provide opportunities for students to develop an understanding of group functionality.

*"Just because working in pairs as part of your team worked for task A. It doesn't mean it's necessarily going to be the perfect one for task B. For that first session we let them just go for it. And we'd rather that they, through trial and error, figure things out and naturally learn."* – A2

Groupwork does not necessitate in-person working. Asynchronous working styles were emphasised by educators for communication, and it is also necessary for students to develop remote group working skills, as it reflects how tasks are typically handled in employment. Communication between group members (intragroup) can be facilitated on a course cohort level, creating a shared identity for all students on the academic course. Through systems like Blackboard, students can request and provide assistance, allowing them to explore how communication is used within intragroup settings to achieve shared goals.

*"I think the soft skill they learn is how to collaborate with different members of the team more efficiently. We have a space on our Blackboard system where if they have any questions they can ask them. I encourage other students to chime in and try to answer them. I monitor these discussions and if I see that an answer is not correct, or it needs adjustment then I can jump in. They create a self-organised community where they're trying to help each other in answering questions"* – A1

Longer-term projects offer students team working experiences to prepare them for future careers. Many educators use group-based projects as they combine multiple soft skills and give students experience in more

complex, long-term tasks. Projects typically require the distribution of specific tasks because the project is too time-intensive for one person to complete.

*“The goal of it is to give students experience of doing a medium scale project from: ‘here’s an idea’, coming up with requirements, a design, implementation, testing, it being done for an actual end user as opposed to themselves. As well as doing it in the context of a group because most software development in the future will happen in groups.” – B2*

Conflict Management exists as a by-product of teamwork because students will invariably experience group breakdown. A3 highlights this issue of working collaboratively via GitLab and where differences in working style can lead to frustration. Staff consider it a learning experience in understanding effective collaboration and different working styles.

*“When they have a team member who just does nothing and is invisible to them. They say, we’re all pushing our commits and updating our issues. But this person, we have no idea what they’re doing. And sometimes those people will come in the day before the deadline and push a lot of stuff. So from that they start to understand the important role of visibility of working and managing a team.” – A3*

Group breakdown can occur when students distance themselves from others and fail to contribute. In these situations, emotions can be high, and it can be difficult to articulate or justify complaints about peers. D2 offers students a solution through practical methods, such as taking meeting minutes. These provide a way for students to articulate issues, with the benefit of educators being able to review minutes for fair grading.

*“I think at the time they find [taking minutes] very helpful because almost all of them have to deal with one group member who’s not really performing. And so, learning how to deal with them through the minutes is a useful skill. It takes the characters and the personality out a bit.” – D2*

Students are exposed to intragroup skills such as communication, conflict management, and handling group breakdown, as well as intergroup skills, including communicating with individuals outside their project groups. Some courses introduced components where students were expected to communicate with various stakeholders. Intergroup communication (between groups) requires a different set of skills than intragroup communication because individuals may not share the same goals or values, requiring different means of communication. Students experienced intergroup settings where they were required to engage with “clients” (from within the university) as part of requirements gathering. This intergroup working was not a common approach to education, partly due to the complexity of arranging it.

*“It’s about talking to the client and getting the requirements out of it and learning the important skill that the client isn’t technical and has no idea what they want. And how do you handle that relationship?” – E1*

Educators aim to inculcate students about the relevance of intergroup communication by offering workplace context to specific skill development. This highlights that more than technical skills are needed, particularly when others possess the same technical skills; students must understand how to engage with management or stakeholders to communicate a point effectively.

*“You can be very good at coding but if someone else is better than you in communicating what they have done, what are your unique skills? What is the unique aspect of your work then if the other person can do the job. So I think this kind of argument for them, this is very important, and it catches their attention.” – B1*

Intergroup communication was much more limited compared to intragroup exposure. However, intergroup communication can also be framed through the sustained interaction between educator and student, which



manifests through presentation opportunities or assessments where students are required to deliver high-level information to an individual outside of their group. These development opportunities are much more implicit and speak to the nature of the education system.

*Professional Skills.* Professional skills, which includes ethical thinking and time management, are valuable for successful employment. In SE2014, professional skills include ethical thinking, whereas time management is not referenced, indicating the implicit expectation to manage and complete activities across the curriculum.

Ethical thinking refers to the way people consider the implications of their actions. It is commonly developed in association with communication, allowing students to engage with each other and be exposed to a greater range of perspectives, which educators can mediate.

*"We asked students in groups to read 10 positive news stories and 10 negative ones, and to make some short notes about why they think this technology is positive, why they think this technology is negative. And at the end of the session we asked them to reflect on the ethical implication, social implication, economics of industry, implication of what they have read. I think this is kind of connected with the 'what does it mean to you', to reflect on the impact that your work can have in real life" – B1*

Others mentioned opportunities to employ educators who specialise in topics such as ethics. In doing so, they gain the expertise, passion, and interest for incorporating ethics. One of the interviewers noted that not all modules were necessarily taught by domain experts (and whilst they were referring to technical experts, the notion stands for soft skills) who sometimes lacked the passion required. This may transfer to the development of students' valuation of these skills, as they typically converge towards staff values [38].

*"Ethics spans right across the university and they set up a teaching centre where they could collect appropriate staff and those. Staff get seconded across various courses and there's also a faculty ethics team as well that looks at issues across our faculty. If there's an ethical component to what we're teaching, we would generally second someone from that unit to teach it." – D1*

Time management is recognised to be a complex skill to teach, but that it is essential for students to learn. Looking at the course holistically, A1 notes that the department offers so many activities that it is unrealistic for a student to attend everything. From this, students must manage their time to attend to what they consider the most important things for their personal goals.

*"If they attend every single class, lab, lecture, and all the external activities that we organise, there is really no time to go home and do exercises. So they have to make decisions of what to attend and what not to attend." – A1*

Students must manage their time successfully and know when to finish an open-ended task. The diminishing returns on programming tasks can be meaningful learning opportunities. Programming assignments can be made intentionally challenging and time-consuming. However, these attributes offer pedagogical advantages, requiring students to effectively manage their time and decide when they have sufficiently completed an assignment.

*"A lot of interviews have coding interviews, and they use similar style principles. So they have five days to do it. It's hard. It takes time. We had a few students, who said 'I just didn't even bother with the final task', but that's fine. That's the point in a way, as well as to know when to manage their expectations, we have some students who will stay up till who knows what hour in the morning trying to hammer this bit of coursework to gain an extra 1%." – A2*

**3.2.2 Perceptions of student engagement.** Student engagement was primarily negatively perceived by educators. Staff felt that students misrepresent the purpose of a CS degree, limiting their ability to develop soft skills. These misconceptions lead students to attend to their technical development and reduce their attention towards soft skills, resulting in students avoiding or rejecting opportunities to develop or use soft skills, as they perceive them as irrelevant to their careers. Subsequently, students experience a temporal delay between the opportunities to develop soft skills and requiring them, highlighting the event-driven nature of soft skill development. This mismatch may contribute to the skill gap observed between education and employment, where students are not expecting to be exposed to soft skills content.

Preconceptions about what a CS course encapsulates are tied to how modern university education is perceived. As university fees increase, students demand greater value for their money, representing the “student consumer” [53]. This consumer identity may legitimise student expectations that educators must teach the technical content students want to be taught, rather than soft skills. This transactional relationship means students evaluate the cost-benefit of their education, and where they perceive no benefit, they challenge why they are being taught these skills. As students are seen to devalue soft skills, this can lead to tensions between what students want and what they would benefit from being taught in the long term.

*“Getting people to sit still and listen when they think they’ve signed up for something, and that’s the issue. A lot of the current mood is this notion of we are the customers, now give us what we want.” – E1*

Student preconceptions and previous experiences can influence how they internalise the soft skills material they are presented. Naïve perceptions of what is required to secure a career may lead to an overvaluation of technical skills, either because of prior experiences (such as programming hobbies) or a misunderstanding of software engineering.

*“They’re coming back to Uni after maybe some gap year, and they don’t see the purpose of [soft skills]. The students who just arrived from school, they feel [soft skills] don’t have anything to contribute from the career perspective” – C1*

Students may misrepresent what a software engineering career involves, expecting a highly technical endeavour, reducing their receptiveness towards soft skills. Students may avoid engaging in soft skills because of these perceptions, contributing to the soft skills gap. This finding points towards more general perceptions of students, in that underdeveloped skills may remain this way because students refuse to engage in difficult tasks.

*“I have guest speakers come and talk about what it’s really like working in industry. One of the students asked, ‘what if you don’t like working with other people?’ And the industry speaker said, ‘please don’t work in the software industry.’” – D2*

*“Sometimes it is hard to convince our students that we are not making their life difficult.” – D3*

Students may refuse soft skills because they devalue any skills associated with tasks they find challenging and avoid both the task and similar tasks that involve the same skills. Their reduced value results in students attending less focus on their development in favour of activities that they find more enjoyable or rewarding [37].

*“I think there is a mismatch between what we do and what the students expect, or even what the students are good at.” – D1*

Building off D1’s comment about a mismatch between the CS course material and what students are good at is reflected in the following quote by B1, who notes that students do not enjoy writing outside of

programming scenarios. Steady constant exposure can be essential to ensure students recognise where these skills hold value to counter disengagement with skill development. It is a gradual process, but it is necessary to emphasise the persistent presence of soft skills in software development.

*“Students hate to write. They’re not used to it. And the first reaction [is] we don’t want to do that. But then this is a little bit of work that [I] do across the course to create a kind of consistency [to] make them understand why this is important.” – B1*

Soft skills and their domain-agnostic nature can result in students disengaging as they do not see a direct link between the skills and software engineering, emphasising a student’s misperception of the CS degree being a programming course. The students’ categorisation of soft skills as being more relevant to other professions evidences their view of computer science and reflects a misunderstanding of what is required in the workplace, which may involve managing or leading a team as they progress through their careers.

*“Various bits of peer marking go on which has to be moderated. There’s a tension that we often say it’s for you to try and bring on the rest of the group. [And students say], I didn’t sign up to do a management or a leadership course. I’m not a teacher. Why am I having to supervise this guy? And that’s working in a team.” – E1*

A temporal delay was observed between the point where students are first introduced to soft skills and the point when they realise the importance or relevance of these skills. Staff perceive students to reject soft skills during their educational journey, and it is only when they enter the workplace (through placements or employment) that they recognise their value.

*“What they don’t realise is what they’re buying is me knowing better than them about what they’re going to need in ten years’ time. And that’s where we come to those project weeks. They pathologically hate them until they’ve graduated, and they come back and say, ‘no, that’s the thing I talked about in an interview. That’s the thing that’s helped me for most of my career’.” – E1*

The temporal delay reported reflects the event-driven nature of soft skill development. The following two quotes represent two possible event types that can bring on the valuation of soft skills. In A4’s example, they refer to students recognising the value of soft skills in situations where they were originally taught these skills against their desires, and it is post-graduation when they are required to use these skillsets that their impact is recognised. In contrast, D2 describes a different experience students undergo, where an absence of soft skills drives the recognition. Students must have been exposed to these soft skills at some point in their education, where they were required to be implemented to recognise this absence. The event should offer students opportunities to judge whether to use these skills. In choosing not to use these skills voluntarily, their absence may be recognised, driving the recognition of the need for the skill.

*“so we have got a lot of anecdotal evidence where people who hated their second year software engineering Course unit two years after graduation, come back and said this was the most relevant thing I’ve ever learned.” – A4*

*“When I supervise my third years, I say, ‘Guys, I’m leaving it up to you to work out how you want to work in a group. You’re welcome to do all the minutes and stuff I taught you in the first year, but I’m not going to check’. Invariably, they don’t bother, but usually at the feedback at the end they say they wish they had done so.” – D2*

Staff perceive the student experience of soft skills as largely negative, with students rejecting or avoiding opportunities to develop these skills. This rejection is based on student misconceptions about soft skills and

their value. Soft skill development is an event-driven process in which students must experience situations where soft skills are either used successfully and tied into goal attainment or where goals were not met due to an absence of a specific skill.

### 3.3 Discussion

In this first discussion, we cover the main findings and limitations from the interviews before a more comprehensive general discussion later. We reported on the staff perceptions regarding the embedded soft skills. Interviews were conducted across five institutions, offering an enhanced understanding of the soft skills gap between graduates and employment. Using a thematic analysis, we identified core soft skills, which can be separated into cognitive, social, and professional skills. We also identified a pattern in how staff perceive students, with a general refusal of soft skills during education, and only once students enter the industry do they understand the value of soft skills.

**3.3.1 Staff Perceptions of how Soft Skills are Taught.** We answer the first research question, “How do academic staff understand how soft skills are taught in their core modules?” by identifying the skills taught directly and indirectly. Direct methods relate to the intentional incorporation of skills in the course, and indirect methods relate to where skills are a by-product of other activities. Skills can also be delivered explicitly or implicitly, relating to how the skills are presented.

Critical thinking and problem solving are taught explicitly, using non-programming activities to facilitate the development of a critical mindset and implicitly through debugging activities. Non-programming activities are generally viewed positively [52], as they allow the development of critical thinking without requiring an understanding of programming first. Implicit methods, such as debugging, promote reasoning around failure, which can act as a catalyst for critical thinking [17]. Critical thinking and problem solving are valuable for programming-specific skills and are core components of learning effectively, as students need the necessary mental models. Part of the learning process was engaging with the tasks and having staff present to facilitate individual skill development.

Communication was taught directly, forming a key part of teaching materials in explicit and implicit approaches. Explicit activities focused on intragroup skills, such as peer discussions, and implicit methods included tools like GitLab that provided students with experience working with software engineering tools and using them collaboratively to maximise effectiveness. Communicating through these tools is a method received positively by students as it offers more accessible communication methods [20].

Online forums and discussion spaces are reported to increase student engagement through increased discussion of sharing solutions and common concerns [23]. We see that these ideas have been used successfully in practice to the benefit of students and educators, as at least two of the universities mentioned using GitLab. Others may also use this tool, but it was not mentioned in interviews.

Ethics was often directly implemented and was predominantly viewed positively by educators. Ethics was often taught in conjunction with communication, encouraging multiple viewpoints to be heard when considering ethical and legal approaches. The perceptions of ethics held by graduates emphasise a poor regard for its role in the workplace [34], despite employers recognising its importance in the workplace [43].

Skills taught indirectly included conflict management, reading, writing, and time management. Conflict management was indirectly taught through groupwork and manifested where tensions or breakdowns occurred.

This skill arose organically as a facet of group dynamics and was not manufactured by staff. Staff perceived conflict as an unavoidable consequence of collaboration but supported students navigating these difficult situations. Reading, writing, and time management are all components of general learning and academic success.

From the direct-indirect interpretation of skills, we are afforded an idea of how skills are taught during the CS undergraduate education. Many skills are taught alongside others, and technical skills or tools were leveraged to highlight the relevance of certain soft skills towards software engineering. Understanding that skills can be taught in these different styles may mean students do not necessarily recognise all instances when they are taught soft skills and may reject activities explicitly about soft skill development. The core skills of critical thinking, problem solving, communication, and teamwork support previous studies reporting similar findings from graduate populations [11, 56], curricula [25], and employment [2, 42].

Teaching implies a student audience, and so it is helpful to recognise how student engagement is perceived by staff, as this can influence how content is delivered. Students were perceived to view their education as a “vocational programming course” (E1), which has been reported previously [29]. This preconception echoes previous findings that students think non-technical skills are unnecessary for technical careers [32]. While educators and employers agree with the importance of soft skills, our findings suggest staff do not perceive students to recognise the value of soft skills, aligning with previous findings where students undervalue soft skills [10] in favour of the skills they enjoy [37].

Students were also perceived to experience a temporal delay between being taught soft skills and recognising their value. It is often only once out of education, and being exposed to situations where either a soft skill is deployed and recognised to be successful, or situations occurs where an absence of a specific soft skill is noticeable that students realise the importance and value of possessing these skills. As many of the events that drive this recognition are industry-based, students are potentially unaware of the skills they should demonstrate in interviews [5].

**3.3.2 What skills are important.** The second research question asked, “What importance do staff ascribe to certain soft skills, in comparison to other skills?”. From the direct-indirect interpretation of how skills are taught, certain skills are recognised as more valuable than others. For skills to be taught directly, educators make conscious decisions to incorporate activities that promote skill development, indicating their perceived value. Indirect skills are typically a by-product of the general learning cycle. The skills referenced as being directly taught are valued enough by educators to ensure they have sufficient space within the curriculum.

**3.3.3 Limitations.** We recognise the likelihood of a self-selection bias for this study, such that participants report favourable attitudes towards soft skills. Indeed, the four educators who declined the interview declared an absence of soft skills in their modules. Consequently, the sample is not fully representative of a teaching department. Despite missing information on what skills are potentially undervalued across curricula, the interviewees offered insights into how soft skills are perceived by educators.

The semi-structured interview approach incorporated, at times, a more conversational interaction, which may have influenced interview topics, resulting in specific skills being mentioned that would not have otherwise. The benefits outweigh potential biases, as it encouraged greater interaction and further information from the interviewee. For transparency, the anonymised transcripts are made available in the online repository.

## 4 STUDY 2

The first study offered important insight into the educator perspective on their modules. Through these interviews, the manner of how soft skills are taught – directly or indirectly, and implicitly or explicitly – surfaces key information about how technical skills are often structured around the non-technical skills and are used as the method to convey the soft skill content. What remains missing, is how this information relates to how soft skills are embedded with a curriculum. In this second study, we report on a natural language processing approach to identify the embedded soft skills in CS curricula, which provides a more descriptive approach to understanding how soft skills present within CS education, and in combination with the results from the first study, it answers the third research question, “How do these skills align with the information available in online courses and module information?”.

Previous studies have used manual detection methods [18, 25], but natural language processing models can expedite the identification process and offer a more uniform detection of keywords across documents. Fareri et al. [19] developed a model that detected the presence of soft skills and applied this to job profile data, finding general skills of leadership, autonomy, conflict management, and communication as highly valued across job profiles. To the authors’ knowledge, similar methods have yet to be directly applied to curriculum information.

### 4.1 Methodology

Data collection targeted websites of the N8 universities, which were scraped for information related to undergraduate CS courses, including course descriptions, core module information, and any publicly available course-relevant data. The academic year 22/23 was targeted because this was the same academic year the interviews were conducted, which was achieved using the Internet Archive. Scripts for scraping these datasets can be found in the OSF repository along with the scraped data: [https://osf.io/25ecz/?view\\_only=917e6de233484d59abaac243647c89a2](https://osf.io/25ecz/?view_only=917e6de233484d59abaac243647c89a2). The Faculty Ethics committee approved this secondary data collection and analysis.

**4.1.1 Analysis.** SkillNER, a skill-based Named Entity Recognition (NER) model [3], was used to extract soft skills. An NER is an algorithm specialising in identifying keywords based on probabilities and co-occurrences. SkillNER was developed from using 31,278 individual terms (336 soft skills, 29,091 hard skills, and 1,851 certifications) taken from job adverts and other employment-related materials. The analysis pipeline is available on OSF. SkillNER was implemented using Python 3.11 and R 4.2.2 for data analysis.

The dataset was passed through SkillNER. Important soft skills that were not identified by the process were modified in the text to increase extraction rates (e.g., “team” was not recognised and was changed to “teamwork”), which was carried out by data exploration. Following data annotation, the soft skills were grouped according to the skills identified in REDACTED FOR REVIEW, as many identified terms were similar (such as “collaborate” and “collaboration”, or “solving problems” and “problem solving”). Some soft skills by REDACTED FOR REVIEW were identified as hard skills by SkillNER (such as “time management”); these were redefined as soft skills. Identified terms related to the teaching and educational aspect of soft skills (such as “teaching” or “module”) were removed. Terms that had a skill probability over .60 were retained.

## 4.2 Results

References to soft skills are not homogeneous across institutions, and soft skills make up between 12.90% and 27.20% of all referenced skills, with a mean average of 17.45% ( $SD = 4.89$ ). The most frequently mentioned soft skills across the combined curricula were *teamwork*, *communication*, *professional skills*, *ethical thinking*, *critical thinking*, and *problem solving* as shown in Figure 1. *Professional skills* were the only soft skill absent in the list developed in Ivory et al. [34]. Visualising the soft skills between institutions is made more explicit in Figure 2, that some universities have a more even spread across mentions of soft skills, but still, the pattern is inconsistent across institutions, and Figure 3 on an institutional level, offering complementary perspectives.

The institutions are disparate in the levels of soft skills reported. The figures indicate little agreement between institutions regarding how frequently soft skills are mentioned. These figures present the frequencies as proportions because different word counts were extracted from each institution. Teamwork has a presence across all institutions. Professional skills are dominated by three institutions (Lancaster, Leeds, and Newcastle), and most institutions refer to each of the key six skills, except an absence of ethical thinking for Sheffield and problem solving for York.

Skill co-occurrences – how often two skills appear in proximity – can be a useful way to understand skill relationships. In Figure 4, the technical and soft skills are plotted together in a network-based upon within-sentence co-occurrence. The more frequently co-occurring skills are denoted by the strength of the connecting edge, and node size is relative to skill frequency.

We observed that soft skills of *teamwork*, *communication*, *professional skills*, *ethical thinking*, *interpersonal skills*, and *critical thinking* are strongly linked. Technical skills of *software engineering* and *software development* are intrinsically linked with the cluster of soft skills, emphasising that these are taught as team-based activities. *Problem solving* was mentioned more frequently alongside *programming* skills and *autonomy*, indicating that programming emphasises problem solving skills and independent working.

Figure 5 displays relationships between the soft skills and modules across all institutions to assess whether soft skills are taught together or separately. The nodes' size indicates the number of associative links, and the most dispersed skills were *communication*, *critical thinking*, *ethical thinking*, *professional skills*, *problem solving*, and *teamwork*. It can also be seen that 38 modules are only associated with a single soft skill.

Figure 6 offers a different perspective by displaying the proportion of modules that include one of the key six soft skills. It is seen that despite teamwork being the most frequently identified skill (by term matches), it is typically present in around 20% of modules. Communication is most broadly embedded, appearing in over a third of modules in five institutions. Professional skills are widely embedded within Newcastle's curricula but less so for other institutions. The data in these two figures shows that the distribution of soft skills throughout curricula is highly dependent on institutions.

Exploring the soft skills taught across the course structure separated by year, in Figure 7, we see little pattern across the different institutions. The heterogeneity is apparent where some years contain no soft skills, suggesting an inconsistent approach to introducing and teaching soft skills. Ethical thinking is rarely included beyond the second year for any institution, despite dissertation projects requiring ethical approval for human-based research.

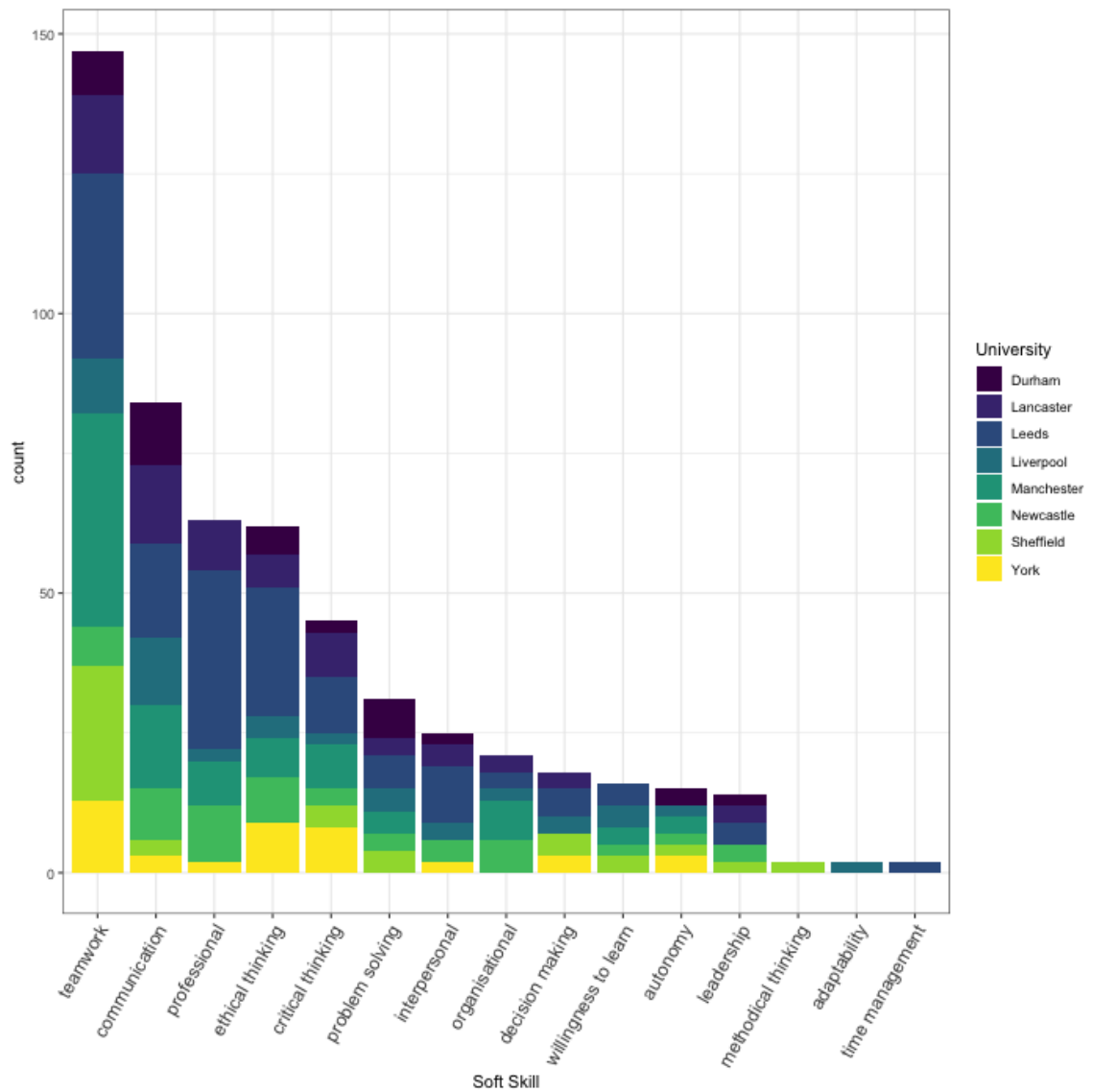


Fig. 1. A stacked column plot indicating the most mentioned soft skills across all curricula. Skills mentioned only once in an institution's curriculum were removed.

### 4.3 Discussion

In this curricula analysis, we extracted soft skills using an NER model. In scraping publicly available information on CS undergraduate courses, we investigated skill frequencies and co-occurrences. The main findings from this exploration are that *communication*, *critical thinking*, *ethical thinking*, *professional skills*, *problem solving*, and *teamwork* were the most frequently mentioned. No clear patterns were identified between or within institutions for how soft skills are mentioned. On an individual institutional level, institutions are



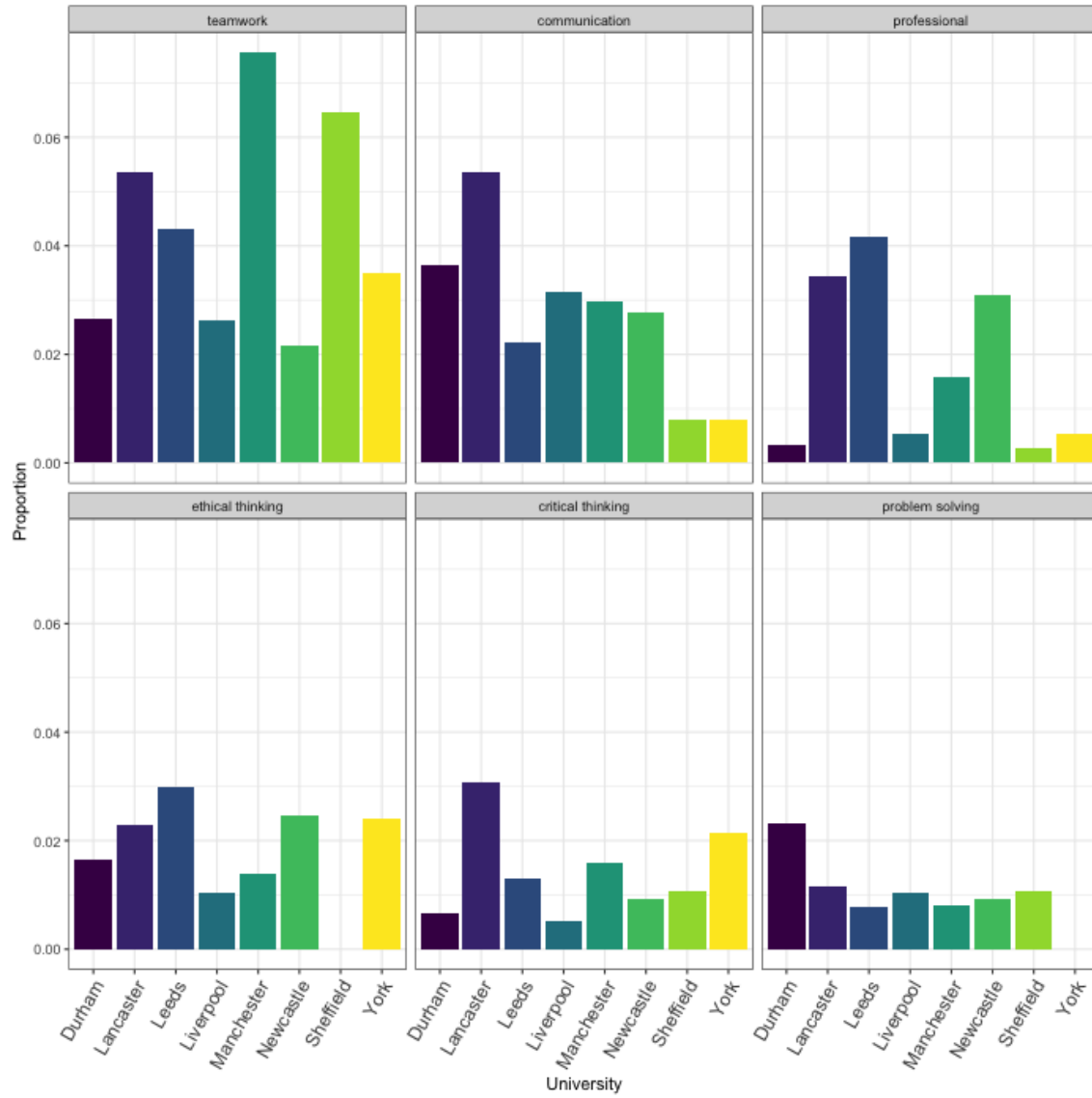


Fig. 2. Separating the frequencies by soft skill shows the patterns (or absence of) across institutions. Only the top six skills are shown.

heterogeneous in referencing soft skills, and this may not necessarily reflect reality. Some institutions may be more descriptive in their module information or reserve information for internal use, making comparisons between institutions difficult. The analysis highlights that soft skills are commonly mentioned in proximity, indicating they are taught or assessed together. The exceptions to this were *problem solving* and *autonomy*, which were more closely associated with programming. It was also seen that software engineering was closely linked to skills of teamwork and communication.

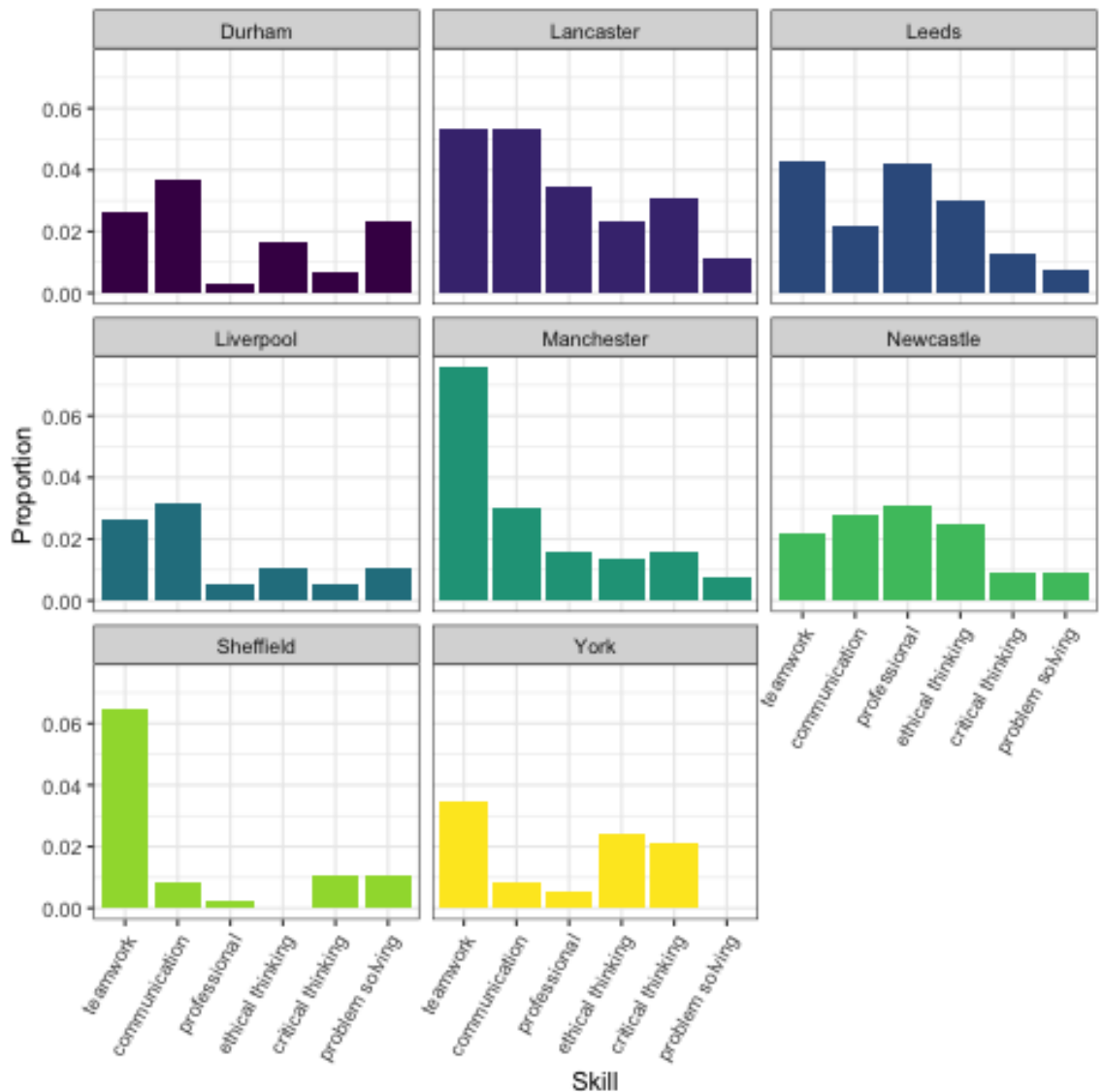
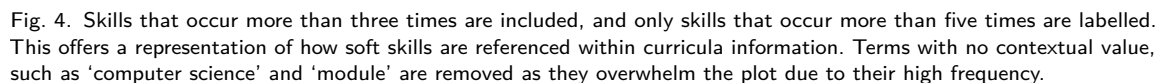


Fig. 3. Presenting soft skills split by their institution offers a different perspective, showing how institutions present different skills within their curricula.

The most frequently mentioned soft skills of *teamwork* and *communication* are those highly valued by software engineering employers [2, 35, 45, 50], as are the skills of *critical thinking* and *problem solving* [18]. This suggests higher education and industry are aligned in the skills valued, supporting the notion that CS undergraduate education is positioned to inculcate students with the appropriate soft skills for software engineering careers. This finding is further supported by previous curriculum analyses identifying *communication*, *teamwork*, and *ethical thinking* [18, 25].



Manuscript submitted to ACM



Fig. 5. Network plot of the soft skills and co-occurrences through modules. The connections indicate skills that are taught in the same modules. Yellow nodes are soft skills, and dark purple nodes are individual modules.

Skills of software engineering and software development were strongly associated with *teamwork* and *communication*, suggesting these are presented to students as social activities. This association reflects employer expectations that software engineering is carried out collaboratively [1].

**4.3.1 Limitations.** The NER model could not identify all references to soft skills without prior processing steps, which was addressed in the analysis pipeline, where the original text was modified to ensure that Manuscript submitted to ACM

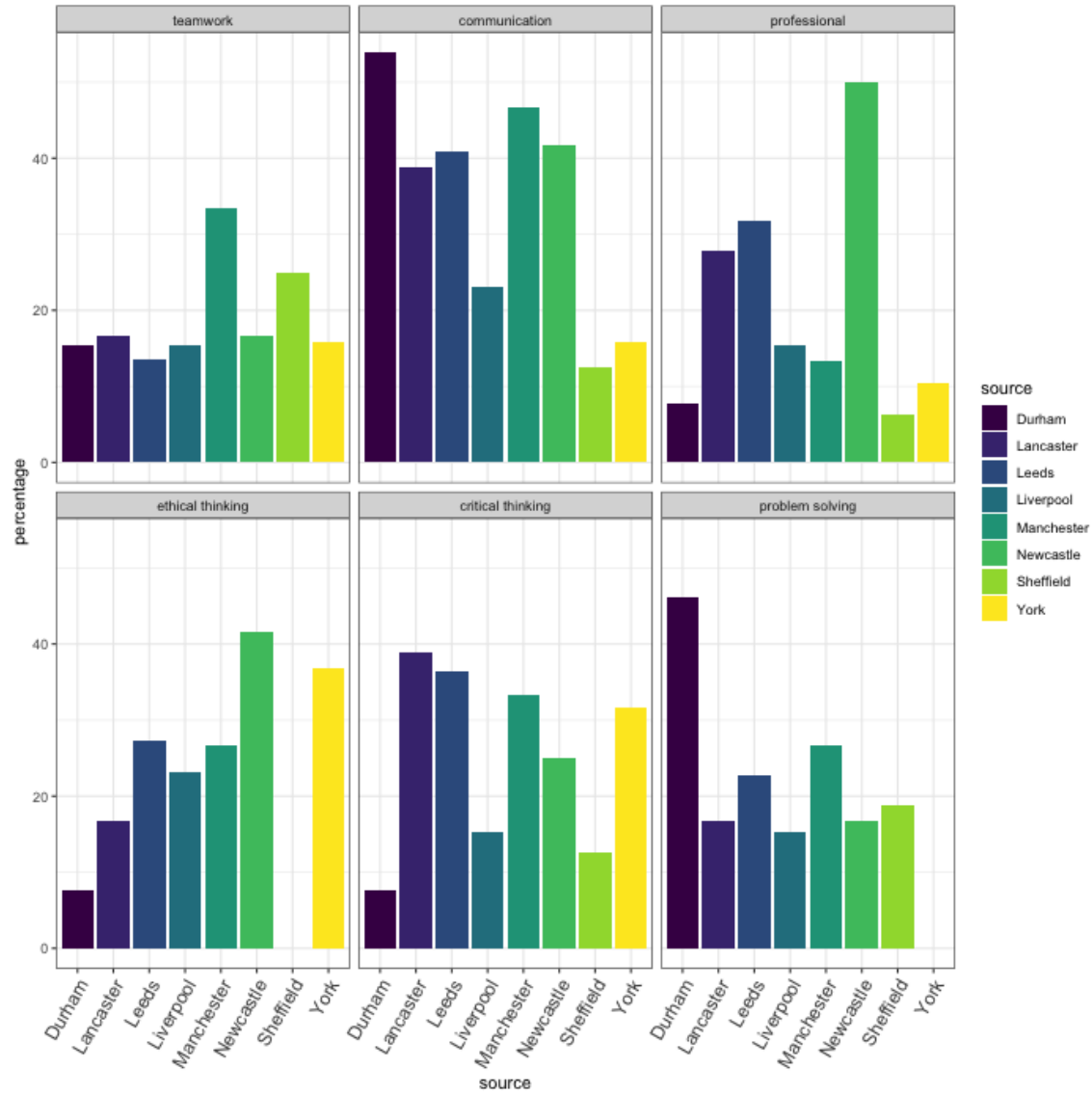


Fig. 6. Proportion of modules within an institution that includes the top six reported soft skills.

SkillNER identified references to teamwork, groupwork, ethical and moral thinking. It remains possible that other terms were not detected. Some skills, such as groupwork or time management, were classed as technical skills by the NER model, but to maintain consistency with previous research, we classified these as soft skills according to the definitions used in REDACTED FOR REVIEW . Both issues indicate a broader issue with soft skill research: they are ephemeral concepts with loose definitions. This is not just a limitation of the present study but also speaks to general soft skills research. The findings indicated reasonable alignment with current understandings of valued soft skills despite model limitations.

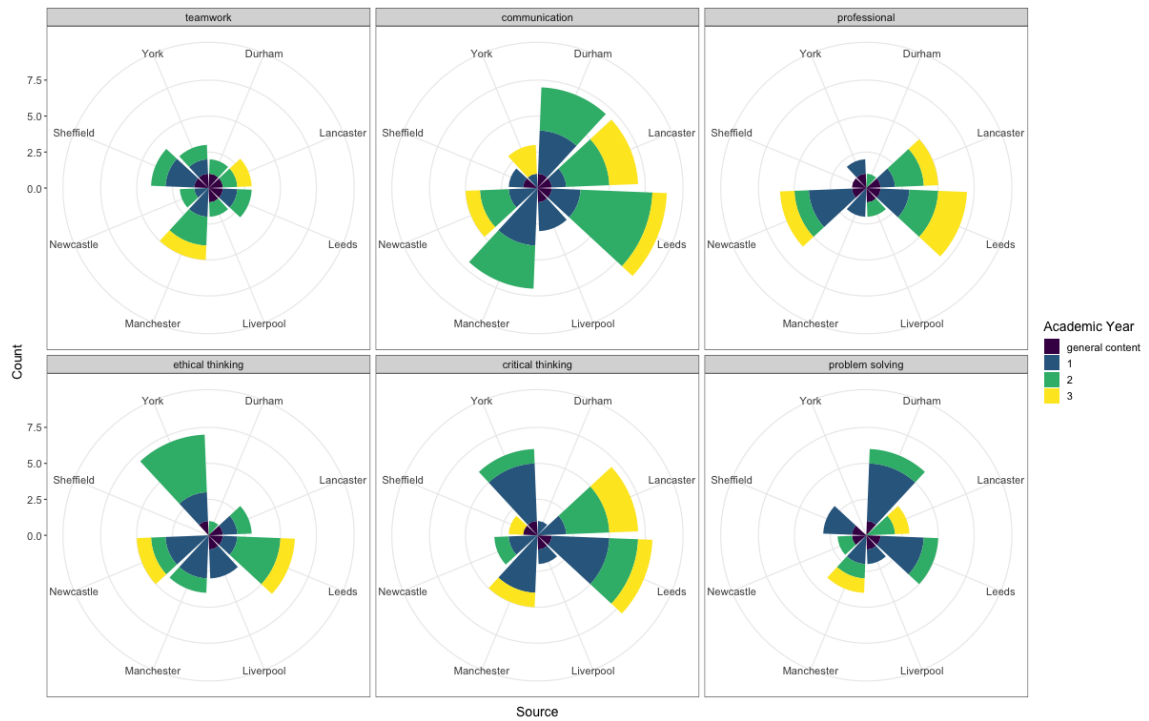


Fig. 7. Inspection of soft skills by course structure (module years), no distinct pattern is observed across skills.

## 5 GENERAL DISCUSSION

In this two-study paper, we first interviewed staff to identify the perceptions of embedded soft skills across core modules, followed by the extraction of soft skills from curricula to converge on valued soft skills. It was seen that staff valued skills including *communication*, *groupwork*, and *critical thinking*, which were mentioned across modules but were also extensively taught directly, highlighting perceived importance. The second study applied an NER model to core module content, highlighting heterogeneous frequencies across different institutions and a strong link between software development and groupwork. In this general discussion, we draw the findings from both studies together, and describe future research.

The first study answered research questions 1 and 2 by interpreting soft skills taught directly or indirectly. For question 1, “How do academic staff understand how soft skills are taught in their core modules?” the interviews highlighted that soft skills are taught in various ways, but combining soft skills with relevant software engineering tools or practices provides ways for students to recognise the value of soft skills. The second research question asked, “What importance do staff ascribe to certain soft skills, in comparison to other skills?”, and the skills intentionally incorporated into the teaching materials can be seen as more valuable. Staff are constrained by competing modules and time, so they must make conscious decisions about how, and which, soft skills are given space within curricula.

Manuscript submitted to ACM

To answer the third research question of “How do important soft skills align with the information available in online course and module information?”, the information from study 1 and study 2 are assessed together. General agreement between the most valued skills is seen as educators spoke about *communication*, *groupwork*, and *critical thinking* as being intentionally included within their materials, and these are also the skills identified most frequently. The course information and curricula are assumed to reflect the skills embedded within the course. Extracting soft skills presents a course-wide representation of the skills that educators value. This alignment indicates that the curriculum information is accurate, and educators are teaching the soft skills they advertise.

No discernible patterns were seen between or within institutions, suggesting that despite frameworks like SE2104, there is little conformity between institutional curriculum design. This is positive in that it highlights unique institutional offerings that can be tailored to their branding, but this absence of uniformity suggests the frameworks are interpreted very differently, to the extent that certain skills are not focused on in certain institutions. This can lead to potential choices that students are required to make in their selection of education at a point in time where their misconceptions may draw them to programmes that reference lower levels of soft skills. It should be encouraged that curricula reflect the true contents of their course and ensure that the relevant skills are targeted and mentioned.

The skills identified are some of the most desired by employers, who desire employees who can communicate and collaborate effectively [2, 41, 45]. Our findings suggest educators are aligned with employer expectations, highlighting that the educators interviewed are aware of which skills are needed to secure employment. This alignment indicates that the soft skills gap may result from student perceptions and how they develop these skills. While staff perceptions influence how students perceive soft skills [10], it is reliant on students to fully realise their value [24].

Many students start their undergraduate education believing a CS degree is primarily a vocational programming course [29], accompanied by the misconception that programming is all that is required for a successful career. Despite these preconceptions, students graduate from their courses with a stronger sense of the breadth of skills needed [38]. Our findings suggest the CS undergraduate course is set up to ensure graduates possess the necessary skills for software engineering, but the difficulty is ensuring students recognise this.

The research implications suggest educators should bring forward the events that motivate students' recognition of soft skills and reduce the temporal gap between being taught and appreciating their relevance. To close the soft skills gap, students must reflect on their own goals and recognise the role of soft skills [13]. Students' preconceptions about the importance of soft skills manifest in technical skill prioritisation over soft skills [33]. While preconceptions can be altered through exposure to soft skills, bringing perceived value up for specific skills [10], this is still limited by internal understanding of their value.

Staff recognise that students fail to value soft skills unless prompted by some event involving soft skill use, so incorporating more industry-specific events into the curriculum can increase students' value of soft skills. There are multiple ways to achieve this; one method would involve industry partners offering industry experiences to contextualise soft skills. Another opportunity could be to integrate short-term industry placements into the curriculum, allowing students to experience real-world situations that offer events to reduce the perceived temporal distance. The event-driven nature of soft skill development should

be integrated into modules through project work, along with ensuring students are aware of the relevance of soft skills [36].

Further implications result from the application of the NER model. Similar approaches can be applied during curriculum designs to ensure that soft skills are embedded throughout the content. Embedding soft skills across core modules would offer repeated exposure to these skills and emphasise their alignment with technical skills. By deploying NER models, staff can ensure that soft skills are suitably distributed across modules and are not localised to specific modules.

Our research offers a unique contribution to the understanding of CS higher education and its role in teaching soft skills. Previous work has explored educator perceptions and what they teach, highlighting that CS courses should prepare students for employment [55] by giving them experience with relevant tools and methods widely used in software engineering [14]. We contribute by exploring educator perceptions about the embedded soft skills and their presence in curricula. We find that skills align with employer expectations, suggesting that students may be the weak link in the soft skill development process.

**5.0.1 Future Work.** Staff reportedly say students reject soft skills until they experience an event that emphasises soft skill relevance. We suggest the soft skills gap observed is formed through students' misunderstanding of the skills' value for employment. By conducting longitudinal studies following students through their academic careers, it may be possible to map soft skill development to their education. Using survey studies following whole cohorts and interviews to pinpoint specific moments or modules that altered a perception, it may be possible to contribute a greater understanding of how soft skills develop and potential reasons for refusal, misunderstanding, or acceptance.

One study limitation noted was the self-selecting sample. Capturing information from across all educators would offer meaningful insight into the distribution of educators' perceptions of soft skills and where these skills exist within a curriculum. It was recognised that not all educators see soft skills as having space within their modules, but gathering representative views is necessary to understand how students are exposed to soft skills.

The NER algorithm was applied to a geographically restricted population. Exploring a larger sample and systematically mapping modules between universities can be used to investigate whether soft skills are taught in similar contexts. This development would offer important information about where students can develop soft skills, complementing existing research. All the future studies detailed here offer significant implications for CS curricula design, ensuring soft skills are effectively embedded and calibrated to allow students to recognise the value of soft skills.

## 6 CONCLUSION

In this two-part preregistered study, we explored the staff perceptions and curriculum presentations of embedded soft skills within the CS undergraduate course. Staff perceived soft skills of *communication*, *critical thinking*, *ethical thinking*, *problem solving*, and *teamwork* as valuable soft skills that warrant including and teaching directly to students. Staff talk about the intentional space made in their modules to teach and develop these skills whilst supporting students. Staff views on student engagement were largely negative, with students rejecting the value of soft skills during education but recognising it once in industry, highlighting the event-driven nature of soft skill development. The NER analysis of the curricula highlighted important soft



skills of *communication, critical thinking, ethical thinking, professional skills, problem solving, and teamwork* showing high association between staff perceptions and the descriptions of the courses. The implications of these findings highlight congruency between employer expectations and the educational course in prioritising specific soft skills. The findings imply that student perceptions likely have a strong influence over their internalisation of soft skills, which results in the reported skill gap. Our findings indicate that the critical factor in causing the soft skills gap may be student perceptions, as we observe a positive relationship between the soft skills described in curricula, the soft skills prioritised by educators, and the skills considered essential in employment.

## APPENDIX

### Appendix A

#### Module Leader and Author Interview Guide

Questions (main questions in bold, follow up questions in italics)

- Tell me about the undergraduate module that you teach. What do you think works well with your course? *What key concepts are taught to students? How is the course delivered?*
- What do you understand the term “soft skills” to mean? *< I can then bring up the definitions of soft skills that I refer to; things such as teamwork or responsibility, problem solving Some of these which are more present than we may think originally >*
- How important do you think these skills to be in Computer Science or Software Development? *Who is responsible for ensuring software developers possess the non-technical skills necessary for industry?*
- **Do you teach or emphasise any specific soft skills in the module, either intentionally or as a result of other aims/goals/external requirements of the module?** *If YES: why are these included in the module? If NOT INTENTIONAL: is there any reason why they are not purposely included? Are these the responsibility of someone else or do the opportunities for these soft skills develop naturally?*
- **What other aspects of the undergraduate programme are you aware of that focus on soft skills and how do you think these contribute to students’ development?**
- **Has the module, under your supervision, been altered or adapted to emphasise or reduce focus on particular soft skills?** *If YES: why has this been altered? If NO: why not? Do you think the balance of teaching requires no further development or inspection?*
- **What soft skills do you prioritise in teaching in the undergraduate programme?** *Do you think this is different to what students prioritise?*
- *Why are they prioritising these skills? How do these skills integrate in the course? Are they introduced and students left to their own development, or is it more intentional and they’re pushed to improve? Contentious perhaps, but do these skills help improve students’ career prospects or are they more ideological?*
- **Do you have any further thoughts or ideas relating to soft skills in software development and computer science?** *Anything that has not been covered by previous questions? Any last thoughts on the balance of technical and non-technical skills.*

## REFERENCES

- [1] Faheem Ahmed, Luiz Fernando Capretz, Salah Bouktif, and Piers Campbell. 2015. Soft Skills and Software Development: A Reflection from Software Industry. *International Journal of Information Processing and Management* 4, 3 (July 2015), 171–191. <https://doi.org/10.48550/arXiv.1507.06873> arXiv:1507.06873
- [2] Faheem Ahmed, Luiz Fernando Capretz, and P. Campbell. 2012. Evaluating the Demand for Soft Skills in Software Development. *IT Professional* 14, 1 (Jan. 2012), 44–49. <https://doi.org/10.1109/MITP.2012.7>
- [3] Anas Ait Aomar. 2023. SkillNER: A (Smart) Rule Based NLP Module to Extract Job Skills from Text. <https://github.com/AnasAito/SkillNER>.
- [4] Deniz Akdur. 2021. Skills Gaps in the Industry: Opinions of Embedded Software Practitioners. *ACM Transactions on Embedded Computing Systems* 20, 5 (July 2021), 43:1–43:39. <https://doi.org/10.1145/3463340>
- [5] Christina Andersson and Doina Logofatu. 2018. Using Cultural Heterogeneity to Improve Soft Skills in Engineering and Computer Science Education. In *2018 IEEE Global Engineering Education Conference (EDUCON)*. 191–195. <https://doi.org/10.1109/EDUCON.2018.8363227>
- [6] Mark Ardis, David Budgen, Gregory W. Hislop, Jeff Offutt, Mark Sebern, and Willem Visser. 2015. SE 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. *Computer* 48, 11 (Nov. 2015), 106–109. <https://doi.org/10.1109/MC.2015.345>
- [7] BCS. 2023. University Computing Departments Met with Record Applicant Numbers as AI Hits the Mainstream — BCS. <https://www.bcs.org/articles-opinion-and-research/university-computing-departments-met-with-record-applicant-numbers-as-ai-hits-the-mainstream/>.
- [8] Pierre Bourque and Richard E. Fairley. 2014. *Guide to the Software Engineering Body of Knowledge*. Technical Report 3.0. IEEE Computer Society. 1–335 pages.
- [9] Virginia Braun and Victoria Clarke. 2006. Using Thematic Analysis in Psychology. *Qualitative research in psychology* 3, 2 (2006), 77–101. <https://doi.org/10.1191/1478088706qp0630a>
- [10] Manuel Caeiro-Rodríguez, Mario Manso-Vázquez, Fernando A. Mikic-Fonte, Martín Llamas-Nistal, Manuel J. Fernández-Iglesias, Hariklia Tsalapatas, Olivier Heidmann, Carlos Vaz De Carvalho, Triinu Jesmin, Jaanus Terasmaa, and Lene Tolstrup Sørensen. 2021. Teaching Soft Skills in Engineering Education: An European Perspective. *IEEE Access* 9 (2021), 29222–29242. <https://doi.org/10.1109/ACCESS.2021.3059516>
- [11] André Calitz, Margaret Cullen, and Jean Greyling. 2015. *South African Alumni Perceptions of the Industry ICT Skills Requirements*.
- [12] Luiz Fernando Capretz and Faheem Ahmed. 2018. A Call to Promote Soft Skills in Software Engineering. *Psychology and Cognitive Sciences - Open Journal* 4, 1 (Aug. 2018), e1–e3. <https://doi.org/10.17140/PCSOJ-4-e011> arXiv:1901.01819
- [13] Secil Caskurlu, Iryna Ashby, and Marisa Exter. 2017. The Alignment Between Formal Education and Software Design Professionals' Needs in Industry: Faculty Perception. In *2017 ASEE Annual Conference & Exposition*.
- [14] Vincent A. Cicirello. 2017. Student Developed Computer Science Educational Tools as Software Engineering Course Projects. *Journal of Computing Sciences in Colleges* 32, 3 (Jan. 2017), 55–61.
- [15] Catalina Cortázar, Iñaki Goñi, Andrea Ortiz, and Miguel Nussbaum. 2024. Are Professional Skills Learnable? Beliefs and Expectations Among Computing Graduates. *ACM Transactions on Computing Education* (Jan. 2024). <https://doi.org/10.1145/3641551>
- [16] Michelle Craig, Phill Conrad, Dylan Lynch, Natasha Lee, and Laura Anthony. 2018. Listening to Early Career Software Developers. *Journal of Computing Sciences in Colleges* 33, 4 (April 2018), 138–149.
- [17] David DeLiema, Maggie Dahn, Virginia Flood, Ana Asuncion, Dor Abrahamson, Noel Enyedy, and Francis Steen. 2019. Debugging as a Context for Fostering Reflection on Critical Thinking and Emotion. In *Deeper Learning, Dialogic Learning, and Critical Thinking*. Routledge, 209–228.
- [18] Rosanne English and Alan Hayes. 2022. Towards Integrated Graduate Skills for UK Computing Science Students. In *Proceedings of the 2022 Conference on United Kingdom & Ireland Computing Education Research (UKICER '22)*. Association for Computing Machinery, New York, NY, USA, 1–7. <https://doi.org/10.1145/3555009.3555018>
- [19] Silvia Fareri, Nicola Melluso, Filippo Chiarello, and Gualtiero Fantoni. 2021. SkillNER: Mining and Mapping Soft Skills from Any Text. *Expert Systems with Applications* 184 (Dec. 2021), 115544. <https://doi.org/10.1016/j.eswa.2021.115544> arXiv:2101.11431
- [20] Joseph Feliciano. 2015. *Towards a Collaborative Learning Platform: The Use of GitHub in Computer Science and Software Engineering Courses*. Thesis.
- [21] Raluca Florea and Viktoria Stray. 2018. Software Tester, We Want to Hire You! An Analysis of the Demand for Soft Skills. In *Agile Processes in Software Engineering and Extreme Programming (Lecture Notes in Business Information Processing)*, Juan Garbajosa, Xiaofeng Wang, and Ademar Aguiar (Eds.). Springer International Publishing, Cham, 54–67. [https://doi.org/10.1007/978-3-319-91602-6\\_4](https://doi.org/10.1007/978-3-319-91602-6_4)

- [22] Matthias Galster, Antonija Mitrovic, Sanna Malinen, and Jay Holland. 2022. What Soft Skills Does the Software Industry \*Really\* Want? An Exploratory Study of Software Positions in New Zealand. In *Proceedings of the 16th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '22)*. Association for Computing Machinery, New York, NY, USA, 272–282. <https://doi.org/10.1145/3544902.3546247>
- [23] Lucia M. M. Giraffa, Marcia Cristina Moraes, and Lorna Uden. 2014. Teaching Object-Oriented Programming in First-Year Undergraduate Courses Supported By Virtual Classrooms. In *The 2nd International Workshop on Learning Technology for Education in Cloud (Springer Proceedings in Complexity)*, Lorna Uden, Yu-Hui Tao, Hsin-Chang Yang, and I-Hsien Ting (Eds.). Springer Netherlands, Dordrecht, 15–26. [https://doi.org/10.1007/978-94-007-7308-0\\_2](https://doi.org/10.1007/978-94-007-7308-0_2)
- [24] Carolin Gold-Veerkamp. 2019. A Software Engineer’s Competencies: Undergraduate Preconceptions in Contrast to Teaching Intentions. In *Proceedings of the 52nd Hawaii International Conference on System Sciences*. <https://doi.org/10.24251/HICSS.2019.937>
- [25] Wouter Groeneveld, Brett A. Becker, and Joost Vennekens. 2020. Soft Skills: What Do Computing Program Syllabi Reveal About Non-Technical Expectations of Undergraduate Students?. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '20)*. Association for Computing Machinery, New York, NY, USA, 287–293. <https://doi.org/10.1145/3341525.3387396>
- [26] Wouter Groeneveld, Hans Jacobs, Joost Vennekens, and Kris Aerts. 2020. Non-Cognitive Abilities of Exceptional Software Engineers: A Delphi Study. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 1096–1102. <https://doi.org/10.1145/3328778.3366811>
- [27] Wouter Groeneveld, Joost Vennekens, and Kris Aerts. 2019. Software Engineering Education Beyond the Technical: A Systematic Literature Review. <https://doi.org/10.48550/arXiv.1910.09865> arXiv:cs/1910.09865
- [28] Orit Hazzan and Gadi Har-Shai. 2013. Teaching Computer Science Soft Skills as Soft Concepts. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE '13)*. Association for Computing Machinery, New York, NY, USA, 59–64. <https://doi.org/10.1145/2445196.2445219>
- [29] Michael Hewner. 2013. Undergraduate Conceptions of the Field of Computer Science. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research (ICER '13)*. Association for Computing Machinery, New York, NY, USA, 107–114. <https://doi.org/10.1145/2493394.2493414>
- [30] Michael Hewner. 2014. How CS Undergraduates Make Course Choices. In *Proceedings of the Tenth Annual Conference on International Computing Education Research*. ACM, Glasgow Scotland United Kingdom, 115–122. <https://doi.org/10.1145/2632320.2632345>
- [31] Francisco Iniesto, Julia Sargent, Bart Rienties, Ariadna Llorens, Araceli Adam, Christothea Herodotou, Rebecca Ferguson, and Henry Muccini. 2021. When Industry Meets Education 4.0: What Do Computer Science Companies Need from Higher Education?. In *Ninth International Conference on Technological Ecosystems for Enhancing Multiculturality (TEEM'21) (TEEM'21)*. Association for Computing Machinery, New York, NY, USA, 367–372. <https://doi.org/10.1145/3486011.3486475>
- [32] Mona Itani and Issam Srour. 2016. Engineering Students’ Perceptions of Soft Skills, Industry Expectations, and Career Aspirations. *Journal of Professional Issues in Engineering Education and Practice* 142, 1 (Jan. 2016), 04015005. [https://doi.org/10.1061/\(ASCE\)EI.1943-5541.0000247](https://doi.org/10.1061/(ASCE)EI.1943-5541.0000247)
- [33] Jim Ivins, Brian R Von Konsky, David Cooper, and Michael Robey. 2006. Software Engineers and Engineering: A Survey of Undergraduate Preconceptions. In *Proceedings. Frontiers in Education. 36th Annual Conference*. 6–11. <https://doi.org/10.1109/FIE.2006.322364>
- [34] Matthew Ivory, John Towse, Miriam Sturdee, Mark Levine, and Bashar Nuseibeh. 2023. What’s in an Undergraduate Computer Science Degree; Alumni Perceptions about Soft Skills in Careers. *Transactions on Computing Education* (2023).
- [35] Jingdong Jia, Zupeng Chen, and Xiaoping Du. 2017. Understanding Soft Skills Requirements for Mobile Applications Developers. In *2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, Vol. 1. 108–115. <https://doi.org/10.1109/CSE-EUC.2017.29>
- [36] Haleh Karimi and Anthony Pina. 2021. Strategically Addressing the Soft Skills Gap Among STEM Undergraduates. *Journal of Research in STEM Education* 7, 1 (July 2021), 21–46. <https://doi.org/10.51355/jstem.2021.99>
- [37] Antti Knutas, Timo Hynninen, and Maija Hujala. 2021. To Get Good Student Ratings Should You Only Teach Programming Courses? Investigation and Implications of Student Evaluations of Teaching in a Software Engineering Context. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. 253–260. <https://doi.org/10.1109/ICSE-SEET52601.2021.00035>
- [38] Clayton Lewis, Michele H. Jackson, and William M. Waite. 2010. Student and Faculty Attitudes and Beliefs about Computer Science. *Commun. ACM* 53, 5 (May 2010), 78–85. <https://doi.org/10.1145/1735223.1735244>

# Everything but Programming; Investigating Academics' Perceptions of Embedded Soft Skills in Computer Science Undergraduate Education

31

- [39] Sherlock A. Licorish, Matthias Galster, Georgia M. Kapitsaki, and Amjed Tahir. 2022. Understanding Students' Software Development Projects: Effort, Performance, Satisfaction, Skills and Their Relation to the Adequacy of Outcomes Developed. *Journal of Systems and Software* 186 (April 2022), 111156. <https://doi.org/10.1016/j.jss.2021.111156>
- [40] Janet Liebenberg, Magda Huisman, and Elsa Mentz. 2014. Knowledge and Skills Requirements for Software Developer Students. *International Journal of Social, Behavioral, Educational, Economic, Business and Industrial Engineering* 8, 8 (2014), 6.
- [41] Gerardo Matturro. 2013. Soft Skills in Software Engineering: A Study of Its Demand by Software Companies in Uruguay. In *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. 133–136. <https://doi.org/10.1109/CHASE.2013.6614749>
- [42] Gerardo Matturro, Florencia Raschetti, and Carina Fontán. 2019. A Systematic Mapping Study on Soft Skills in Software Engineering. *Journal of Universal Computer Science* 25, 1 (2019), 26.
- [43] Anna Mitchell, Dharini Balasubramaniam, and Jade Fletcher. 2022. Incorporating Ethics in Software Engineering: Challenges and Opportunities. In *2022 29th Asia-Pacific Software Engineering Conference (APSEC)*. 90–98. <https://doi.org/10.1109/APSEC57359.2022.00021>
- [44] K. Molokken-Ostfold and M. Jorgensen. 2005. A Comparison of Software Project Overruns - Flexible versus Sequential Development Models. *IEEE Transactions on Software Engineering* 31, 9 (Sept. 2005), 754–766. <https://doi.org/10.1109/TSE.2005.96>
- [45] João Eduardo Montandon, Cristiano Politowski, Luciana Lourdes Silva, Marco Tulio Valente, Fabio Petrillo, and Yann-Gaël Guéhéneuc. 2021. What Skills Do IT Companies Look for in New Developers? A Study with Stack Overflow Jobs. *Information and Software Technology* 129 (Jan. 2021), 106429. <https://doi.org/10.1016/j.infsof.2020.106429>
- [46] Rémi Rampin and Vicky Rampin. 2021. Taguette: Open-Source Qualitative Data Analysis. *Journal of Open Source Software* 6, 68 (Dec. 2021), 3522. <https://doi.org/10.21105/joss.03522>
- [47] Barbara A. Ritter, Erika E. Small, John W. Mortimer, and Jessica L. Doll. 2018. Designing Management Curriculum for Workplace Readiness: Developing Students' Soft Skills. *Journal of Management Education* 42, 1 (Feb. 2018), 80–103. <https://doi.org/10.1177/1052562917703679>
- [48] Mahbub Sarkar, Tina Overton, Christopher Thompson, and Gerry Rayner. 2016. Graduate Employability: Views of Recent Science Graduates and Employers. *International Journal of Innovation in Science and Mathematics Education* 24, 3 (Aug. 2016).
- [49] Christopher Scaffidi. 2018. Employers' Needs for Computer Science, Information Technology and Software Engineering Skills Among New Graduates. *International Journal of Computer Science, Engineering and Information Technology* 8, 1 (Feb. 2018), 01–12. <https://doi.org/10.5121/ijcseit.2018.8101>
- [50] Matt Stevens and Richard Norman. 2016. Industry Expectations of Soft Skills in IT Graduates: A Regional Survey. In *Proceedings of the Australasian Computer Science Week Multiconference (ACSW '16)*. Association for Computing Machinery, New York, NY, USA, 1–9. <https://doi.org/10.1145/2843043.2843068>
- [51] Chiara Succi and Magali Canovi. 2020. Soft Skills to Enhance Graduate Employability: Comparing Students and Employers' Perceptions. *Studies in Higher Education* 45, 9 (Sept. 2020), 1834–1847. <https://doi.org/10.1080/03075079.2019.1585420>
- [52] Sarah Tasneem. 2012. Critical Thinking in an Introductory Programming Course. *Journal of Computing Sciences in Colleges* 27, 6 (June 2012), 81–83.
- [53] Michael Tomlinson. 2017. Student Perceptions of Themselves as 'Consumers' of Higher Education. *British Journal of Sociology of Education* 38, 4 (May 2017), 450–467. <https://doi.org/10.1080/01425692.2015.1113856>
- [54] Sander Valstar, Sophia Krause-Levy, Alexandra Macedo, William G. Griswold, and Leo Porter. 2020. Faculty Views on the Goals of an Undergraduate CS Education and the Academia-Industry Gap. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 577–583. <https://doi.org/10.1145/3328778.3366834>
- [55] Sander Valstar, Caroline Sih, Sophia Krause-Levy, Leo Porter, and William G. Griswold. 2020. A Quantitative Study of Faculty Views on the Goals of an Undergraduate CS Program and Preparing Students for Industry. In *Proceedings of the 2020 ACM Conference on International Computing Education Research (ICER '20)*. Association for Computing Machinery, New York, NY, USA, 113–123. <https://doi.org/10.1145/3372782.3406277>
- [56] Catherine Watson and Kelly Blincoe. 2017. Attitudes Towards Software Engineering Education in the New Zealand Industry. (2017).

## 5.1 Statement of Continuous Thesis Summary

*“Give a man a fish and you feed him for a day; teach a man to fish and you feed him for a lifetime.” - Maimonides*

This chapter presents two studies that explore the embedded soft skills in undergraduate CS degrees. Chapter 4 focuses on university graduates’ perspective, and the present chapter explores the other side, the staff who design and structure the student experience. In the first study, I interview staff perceptions of embedded soft skills in core programme modules. Using thematic analysis, an emphasis on cultivating critical thinking, communication, and teamwork skills emerges. Staff observe the event-driven nature of soft skill development and perceive students to view soft skills negatively at first, which changes once they are exposed to industry situations that demand these skills. The second study applies a Named Entity Recognition (NER) algorithm to curricula and module information. It identifies a pattern in how soft skills are embedded, typically appearing in proximity to other soft skills. *Software engineering* is closely linked to skills of *teamwork* and *communication*, indicating it is taught as a social activity.

This chapter speaks to the soft skills gap (Akdur, 2021) by suggesting educators are aware of the skills desired by industry and consequently incorporate them into their modules. This chapter explores the software learning development process by analysing the curricula and whether they are set up to support students in developing their soft skills. The skills mentioned and embedded within the courses strongly align with SE2014 (Ardis et al., 2015), suggesting a sense of homogeneity across educator perceptions. In offering an answer to the thesis’ first research question, the skills of *communication*, *critical thinking*, *problem solving*, *teamwork*, and *ethical thinking* are presented.

## 5.2 Synthesising Phase 1

In this transitory space between phases 1 and 2, I reflect on the findings from the first phase and focus on two psychological constructs that have potential value for understanding software engineering behaviours. Table 1 reports the top skills from the

Skill	Chapter 4	Chapter 5a	Chapter 5b
Communication	✓	✓	✓
Critical Thinking	✓	✓	✓
Problem Solving	✓	✓	✓
Teamwork	✓	✓	✓
Ethical Thinking		✓	✓
Conflict Management		✓	
Professional Skills			✓
Reading/Writing		✓	
Responsibility	✓		
Time Management		✓	

**Table 1**

*The skills identified in Chapters 4 and 5 as being important for software learning development. 5a refers to study one and 5b to study two in Chapter 5.*

previous chapter and the two studies in the present chapter. The core skills identified are *communication, teamwork, problem solving, and critical thinking*. In the rest of this section, I link these soft skills to two key and well-researched concepts within psychology. These psychological concepts are underutilised in software engineering research and would benefit from greater exposure. These psychological ideas are reviewed in Chapter 2.

### **5.2.1 Communication and Teamwork**

Communication, as defined in Chapter 4, is “being able to convey information to various parties in a manner that is well received and understood, in both written and oral forms”. Communication is the “lifeblood” of an organisation (Goldhaber, 1974) and allows individuals and groups to share information and manage relationships (Pikkarainen et al., 2008). Organisational communication is consistent with the social identity approach, as an individual’s sense of belonging to a team is related to recognising shared values within project teams (Postmes et al., 2001). Through communication, social identities are maintained via the diffusion of shared norms (Lapinski & Rimal, 2005) and the ability to express identities through language (Scott et al., 1998).

Communication is integral to working collaboratively because information needs to be shared between individuals working towards a common goal (Klunder et al., 2016).

Communication requires an internal representation of the knowledge to be conveyed and an understanding of the audience and their internal knowledge construction. The dyadic interaction between a speaker and listener is bounded by their individual perceptions and internal knowledge representation, as well as the consideration of other people's perceptions, offering a broadening of perspectives (Fussell & Krauss, 1992).

Communication has long been acknowledged as necessary within software engineering (Stelzer & Mellis, 1998). Computer-mediated communication can also form identities built around the technology (Amaral & Monteiro, 2002), as seen in the literature surrounding software ecosystems (de Souza et al., 2016). Within software engineering, communication is often computer-mediated; for example, GitHub issues pages and Q&A forums such as StackOverflow are invaluable resources for soliciting advice from others.

The social identity theory is predicated upon the formation of groups that share values and observe group norms. As a result, the relationship between the soft skill of teamwork and social identity theory should be relatively self-evident. For thesis completion, the links are detailed. Teamwork is built upon shared norms and is defined in Chapter 4 as “being capable of working with others effectively and contributing to the end goal; being able to cooperate with others during required tasks”.

Teamwork is a collaborative effort by individuals who share a common goal. Within organisations, teams may exist until a goal is completed (such as delivering a product), or they may be more persistent, reflecting the requirements of a job role. Shared goals allow individuals to identify with others, which sparks the formation of organisational teams (Lembke & Wilson, 1998). These goals may be mandated as part of their role (all programmers share a goal of writing code) or through project teams (designers, programmers, and testers all contributing to the same project). This shared goal becomes a value that draws individuals together.

Large organisations may have multiple project groups working in parallel, and cultivating



an organisation-wide identity can help reduce conflict between internal groups (Haslam & Parkinson, 2005), evidencing that diverse populations of individuals can work together harmoniously where they can identify with some common value. Even within these internal teams, groups are most successful when individuals share an identity, as it aids the self-regulation of the group, reducing the need for supervision (Haslam et al., 2000), which can result in more effective goal-directed behaviour. Social identity theory has strong connections to the soft skills of communication and teamwork (Scott, 2007a), and based on their value for software engineering and security work, social identity theory is appropriate to use in exploring the behaviours of software engineers.

### ***5.2.2 Problem Solving and Critical Thinking***

Problem solving is defined in Chapter 4 as “being able to understand and solve complex problems; being able to evaluate a situation and provide an effective solution”, and Critical Thinking is defined as “being able to critically review information and determine its importance for informing decisions; being able to use evaluative and inferential reasoning to increase the likelihood of a desired outcome”.

Problem solving is often considered one aspect of critical thinking, which comprises the mental processes, strategies, and representations needed to solve problems and make decisions (Sternberg, 1986a). As a critical thinking component, problem solving is the information processing unit, where the task environment determines an individual’s behaviour (Newell & Simon, 1972). It is a cyclic process of individuals recognising a problem, defining it, developing a solution, allocating resources, and monitoring progress before evaluating the solution (Sternberg, 1986b).

Problem solving is not always performed critically and can be deployed uncritically (Bailin & Siegel, 2003). For example, an intuitive and non-critical solution to keeping patient records secure is not allowing anyone access, rendering the database secure *but* unusable. In contrast, a critical perspective would acknowledge that some people require legitimate access, and that the prior solution is non-optimal. Instead, they may choose to

restrict access to named individuals via access control systems. This critical/uncritical distinction suggests the existence of at least two problem solving mechanisms.

To *sufficiently* solve a problem, one must alter their mental representation (Pretz et al., 2003), requiring a critically reflective approach. Critical thinking can be regarded as an approach to making judgments based upon some form of criteria that deems it “good” thinking in that it is reasoned and rational (Bailin & Siegel, 2003). A critically thinking individual is “appropriately moved by reason” (Siegel, 2013) and aspires to engage in reflective practices (Ennis, 1987; Scheffler, 1965). This difference in approaches is explained by the dual processing theory of decision making, which posits the existence of an intuitive default system and an interventionist and more cognitively critical system (Evans, 2003).

Critical thinking is synonymous with rationality (Bailin & Siegel, 2003) as both are interested in how effectively reasoning is applied during decision making (Scheffler, 1965). Rationality is tied to normative behaviours, yet individuals are repeatedly shown to choose non-normative solutions, evidenced numerous times (e.g., Kahneman & Tversky, 1979, 1982; Wason, 1960; Wason & Evans, 1974). Irrationality is not an adaptive trait, for it does not benefit us to willingly make irrational choices that lead to increased risk without suitable reward. One perspective on decision making is that human rationality is bounded by an individual’s knowledge and mental representation at the time of judgment. So people make decisions based on available information (Simon, 1990), which may result in non-optimal decisions because they were considered the best judgment at the time. From this bounded rationality, individuals engage in “satisficing”, the practice of aiming for a good-enough or satisfactory result instead of the optimal one (Simon, 1956).

To satisfice, individuals make use of heuristics to arrive at an ecologically rational answer, and these heuristics form a part of an adaptive toolbox (Gigerenzer, 2002). Notably, the heuristics are considered ecologically rational where they benefit decision making in suitable environments (Gigerenzer, 2015), but where they fail to provide adequate choices,

their poor fit to the decision context results in biased and irrational choices (Gigerenzer & Todd, 1999; Kahneman et al., 1974).

If the psychology literature views problem solving as just one component of critical thinking (Sternberg, 1986a), why are they identified in Chapters 4 and 5 as separate soft skills, to the extent that in Chapter 4, little association was seen between the value of critical thinking and problem solving? One answer is the popular combination of programming and problem solving (an internet search for “programming problem solving” returns over 99 million more results than “programming critical thinking”). Through exposure, these terms become synonymous and familiar, particularly when presented outside the more formal definitions in psychology. This popularity highlights the importance of phase 1, where efforts to identify the most important soft skills captured some ideas that are not commonly mentioned alongside software engineering. This phase signals the ephemeral concept of soft skills in popular use and demonstrates the necessity of these studies for the thesis. Key ideas are exposed by gaining information from software populations using familiar terminology. In the example of cognitive skills, the critical aspect combined with problem solving supports the relevance of dual processing theory. As previously noted, problem solving is not always critical and may be implemented intuitively (Bailin & Siegel, 2003). The distinction between critical and uncritical problem solving in software engineering tends towards the notion that different cognitive systems can be deployed during decision making.

While considered as separate soft skills, problem solving and critical thinking both share a basis in “good” reasoning and rationality. Rationality is bounded by our proximal stimuli and the mental representations we hold at the time of decision making, and rational thinking is linked to more deliberate, reflective processing styles as proposed in the dual processing theory of decision making. I use this to suggest that dual processing theory holds value within software engineering, that warrants further exploration, which is carried out in Chapters 6 and 8.

In this phase, I identified the most valued soft skills for software engineering and CS from graduates and educators of the CS undergraduate programme. The core skills identified were *communication*, *teamwork*, *problem solving*, and *critical thinking*. From this, the psychological theory of social identity offers reasonable grounding for communication and teamwork, and the dual processing theory grounds critical thinking and problem solving. In phase 2, I investigate these theories in secure software engineering, by applying dual processing theory in Chapters 6 and 8, and social identity theory in Chapter 7.

### ***5.2.3 Contribution to Thesis Argument and Forward Trajectory***

This chapter advances the central thesis argument by investigating how soft skills are intentionally embedded within computer science curricula, offering an institutional and educator-based perspective that complements the graduate-focused findings in Chapter 4. By revealing both staff perceptions and curricular patterns, this chapter provides compelling evidence that educators are aware of industry needs and attempt to embed relevant soft skills into programme design, particularly communication, teamwork, critical thinking, and problem solving. Up to this point in the thesis, the argument has progressed from establishing the importance of soft skills through to identifying graduate perceptions of valued soft skills and their developmental sources. Chapter 5 adds another layer by exposing how educators conceptualise and implement these skills, as well as how these skills manifest structurally within academic programmes.

Looking forward, the next chapters transition from general software education to security-focused software engineering. Chapters 6 and 8 operationalise dual processing theory to examine how problem solving and critical thinking play out in security-specific contexts, while Chapter 7 explores social identity theory in relation to teamwork and communication in secure software engineering environments. These theoretical applications further explore the behavioural dimensions identified in this phase, deepening the psychological and domain-specific insights that underpin the thesis.

The foundational study on the perceptions of software graduates and academic staff

regarding essential soft skills - communication, teamwork, problem solving, and critical thinking - serves as a starting point for exploring these skills' influence across diverse software engineering contexts. While the initial sample is drawn from undergraduate graduates and university staff, representing a relatively homogenous academic environment, the identified core skills are well-established across industry literature and professional standards, such as SE2014 (Ardis et al., 2015), supporting their broader relevance. Moreover, these soft skills are inherently transferable and foundational to effective software development irrespective of specific roles, experience levels, or work settings.

The subsequent empirical studies in this thesis draw from populations with differing software experience, including freelance developers and professional practitioners, whose motivations and contexts vary considerably from those in academia. The rationale for extending the soft skills framework to these populations is premised by the notion that the cognitive and social competencies fundamental to software engineering are consistent across different groups, albeit manifested in contextually specific ways. By anchoring later investigations in the initial study's conceptualisation of soft skills, the thesis maintains coherence while enabling nuanced exploration of how these skills interact with psychological constructs and security behaviours across varied populations.

**6 Recognising The Known Unknowns; the interaction between reflective thinking and optimism for uncertainty among software developer's security perceptions**

Ivory, M., Towse, J., Sturdee, M., Levine, M., & Nuseibeh, B. (2023). Recognizing the Known Unknowns; the Interaction Between Reflective Thinking and Optimism for Uncertainty Among Software Developer's Security Perceptions. *Technology, Mind, and Behavior*, 4(3: Winter 2023). <https://doi.org/10.1037/tmb0000122>

## SINGLE-STUDY PAPER

# Recognizing the Known Unknowns; the Interaction Between Reflective Thinking and Optimism for Uncertainty Among Software Developer's Security Perceptions

Matthew Ivory<sup>1</sup>, John Towse<sup>1</sup>, Miriam Sturdee<sup>2</sup>, Mark Levine<sup>1</sup>, and Bashar Nuseibeh<sup>3, 4</sup>

<sup>1</sup> Department of Psychology, Lancaster University

<sup>2</sup> School of Computer Science, St. Andrews University

<sup>3</sup> Department of Computer Science and Information, LERO (Irish Software Research Centre), Limerick, Ireland

<sup>4</sup> School of Computing and Communications, Open University




Software development is a complex process requiring aspects of social, cognitive, and technical skills. Software engineers face high levels of uncertainty and risk during functional and security decision making. This preregistered study investigates behavioral measures of cognitive reflection, risk aversion, and optimism bias among professional freelance software developers and computer science students, to expose relationships between uncertainty-associated language and risk sensitivity. We employ content analysis with a mixed-effect model to understand how psychological dimensions influence risk sensitivity in secure software development. We show an interaction between cognitive reflection and optimism bias in the proportion of uncertainty-related language used. Overly optimistic outlooks combined with higher cognitive reflection drives up expressions of uncertainty, while pessimistic or realistic individuals reduce uncertainty as cognitive reflection increases. Software engineers who hold average or pessimistic views on the security of their code are more likely to speak more intuitively about security and risk. We discuss the potential of our findings in relation to understanding how to leverage language used by engineers as markers of risk aversion. Encouraging increased discourse could be used as a catalyst for increased cognitive reflection and grounding optimistic behaviors, leading to more careful decisions.

**Keywords:** software engineer, cognitive reflection, optimism bias

**Supplemental materials:** <https://doi.org/10.1037/tmb0000122.supp>

**Action Editor:** Nick Bowman was the action editor for this article.

**ORCID iD:** Matthew Ivory  <https://orcid.org/0000-0002-5296-5897>.

**Acknowledgements:** The authors extend their gratitude to Kat Benier and Maha Sweetha Singaravelu Shanmugam for assisting in the data validation process.


**Funding:** Funding was provided by the Engineering and Physical Sciences Research Council for the project, "The Soft Skills of Software Learning Development: The Psychological Dimensions of Computing and Security Behaviours." All research involved in this project was approved by the University Faculty Ethics committee. Participants provided informed consent before commencing with the research study.


**Disclosures:** The authors declare that there is no conflict of interest.


**Data Availability:** This article has been preregistered and can be accessed at <https://doi.org/10.17605/OSF.IO/ZBQE4>. Data, analysis, and any materials required to reproduce this article are openly accessible at <https://doi.org/10.17605/OSF.IO/SJ8BT>. The data have been used previously, and subject to a thematic analysis that complements this article. A

preprint of this can be found at <https://doi.org/10.31234/osf.io/pexvz>, and the analysis method is different enough to necessitate an independent article.

### Open Science Disclosures:

 The data are available at <https://osf.io/sj8bt/files/osfstorage>.

 The experimental materials are available at <https://osf.io/sj8bt/files/osfstorage>.

 The preregistered design and analysis plan (transparent changes notation) is accessible at <https://osf.io/zbqe4>.

**Open Access License:** This work is licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0; <http://creativecommons.org/licenses/by/4.0>). This license permits copying and redistributing the work in any medium or format, as well as adapting the material for any purpose, even commercially.

**Contact Information:** Correspondence concerning this article should be addressed to Matthew Ivory, Department of Psychology, Lancaster University, Lancaster LA14YW, United Kingdom. Email: [matthew.ivory@lancaster.ac.uk](mailto:matthew.ivory@lancaster.ac.uk)

Software is at the foundation of our modern world; we rely on it for almost everything, from communication to financial transactions, to information storage. When software fails or is altered by malicious actors, this has real-world consequences, including psychological harm (Palassis et al., 2021). Consequently, reducing software vulnerabilities—for example, by applying psychological insights and methods to those who create software—is becoming increasingly important.

Secure coding is defined as a programming practice that avoids software vulnerabilities (Rauf et al., 2021), allowing for data exchanges and logic flows to occur without interference by third parties. For our purposes, a security vulnerability is defined as an unexpected logic flow resulting in exploitable situations allowing for unintended access to information or functionality. Reducing vulnerabilities and increasing risk sensitivity in software engineers therefore has the potential for improving secure coding practices.

The solution appears simple—“Write secure software!,” yet such imperatives are usually not sufficient for secure coding (Hallett et al., 2021). It is well established that warnings or commands alone do not motivate action unless an individual perceives the importance of the command (Dash & Gladwin, 2007). Despite software security’s importance, 76% of software possessed at least one well-known vulnerability in a recent review (Veracode, 2020), and 69% of software engineers were unaware of third-party vulnerabilities within their own code (Kula et al., 2018). Our approach works toward a better understanding of the psychological profiles of those who write software, to ultimately facilitate an increase in the salience of security and deployment of secure practices.

Secure coding can be achieved through technical interventions, such as using dedicated testing teams or rigorous workflows, but with over 40% of the app software engineer community comprising of solo or amateur software engineers (van der Linden et al., 2020), it is important to address the individual within the development process, rather than just roles and technical solutions.

When investigating software engineers’ priorities, functionality has repeatedly superseded security (Kirlappos et al., 2013; Lopez, Sharp, et al., 2019), despite implicit security expectations in high-quality software (Tahaei & Vaniea, 2019). Even when given security-specific coding tasks (such as password storage), security is often omitted unless prompted (Hallett et al., 2021; Naiakshina et al., 2019). A reduced security focus has been attributed variously to a perception of effort (Kirlappos et al., 2013), absence of responsibility (Gotterbarn, 2001), and even to mistaken assumptions that their tools are secure (Nadi et al., 2016; Palombo et al., 2020). To our knowledge, the disposition toward risk and uncertainty has not been systematically explored.

To that end, we use a novel approach to highlight links between domain-general psychological constructs such as cognitive reflection, risk aversion, and optimism biases as mediators of how engineers might think about security. This research emphasizes the need to consider individual differences in the psychology of engineers as their actions can have significant impact within the real world, consistent with the suggestion that heuristics and biases impact secure software decision making (Brun et al., 2022; Oliveira et al., 2014, 2018).

By deliberately using abstract cognitive measures as opposed to signals from software-specific activities, we aim to develop an understanding of how cognition shapes or molds those activities. Through a quantitative analysis of language use around security perceptions, we can highlight links between software security and

the language used. Language is grounded in perception and action and can be used to share our internalized models with others (Hagoort, 2023; Jackendoff, 2009). Through studying how software engineers describe security within their work and their personal experiences (or absence of), we can associate their language with cognition, which reflects their internal models and beliefs, and may reflect their real-world behavior, but importantly, it may provide an opportunity to identify potentially insecure coding practices.

## Cognitive Processing Styles

Dual processing theory frames decision making and reasoning in terms of individual propensity to engage in different cognitive styles (Evans, 2003). System 1 processing is driven by intuition and heuristic use, allowing individuals to reduce complex decisions into simpler operations requiring less cognitive effort (Kahneman et al., 1974; Kahneman & Frederick, 2002). Heuristics form “mental shortcuts” that can be used in multiple scenarios to simplify and speedup decisions (Beike & Sherman, 1994). Through heuristic use, System 1 processing is much faster, automatic, and intuitive compared to more drawn-out processing, or System 2. System 2 processing involves deliberate, thought-out, and analytic judgments, allowing for abstract and hypothetical thinking. System 2 is more computationally demanding than System 1, attempting to arrive at optimal solutions by analyzing available information, and is reserved for situations that cannot be resolved with System 1 processing.

A default-interventionist model of dual processing (Evans & Stanovich, 2013; Kahneman & Frederick, 2002) suggests decision making uses System 1 as a default, and System 2 is deployed only when it is sufficiently cued and available (Damjanović et al., 2019; Evans, 2010b). Both systems can possess conscious and unconscious aspects of cognition (Evans, 2010a).

If System 2 is not sufficiently cued, then decision making relies on heuristics. Gigerenzer (2002) suggested that these heuristics form part of an adaptive toolbox, allowing for rational decisions within the constraints of the available information (bounded rationality). Importantly, a core aspect of the adaptive toolbox is that heuristics can be considered ecologically rational if they benefit decision making within specific contexts (Gigerenzer, 2015). When heuristics fail to provide good choices, it is not strictly the cognitive mechanism itself, but rather its poor fit to the decision context (Gigerenzer & Todd, 1999; Kahneman et al., 1974). This is relevant to writing secure code, as it has been suggested previously that heuristics do not provide rational decisions in these contexts resulting in biased decision making (Oliveira et al., 2018).

Biases can be defined as systematic, flawed response patterns that deviate from expected normative performance (Evans, 1984). Not all heuristics invoke biases and poor decisions, but in certain instances, they can result in nonoptimal decisions. One example of a context where heuristics become biases is software security. In daily life, we possess no security heuristic, and by extension, no heuristic exists for secure coding (Oliveira et al., 2014). Oliveira et al. primed developers for security which increased their sensitivity to software vulnerabilities, dual-processing theory would frame this as priming that triggers System 2 processing. Previous work following the claim that software security is heavily impacted by biased thinking has examined links between general cognition and secure coding (Brun et al., 2022; Oliveira et al., 2018), whereas we build upon this



claim by using widely deployed measures of thinking and dual processing to examine the individual differences.

Previous research suggests security prompting can increase code quality (Hallett et al., 2021), specific vulnerability identification (Spadini et al., 2020), and code comprehension (Danilova et al., 2021). Moreover, System 1 processing can be suppressed in favor of System 2 through prompting, either overtly through verbal/written requests (Evans & Stanovich, 2013; Pennycook et al., 2020), or implicitly through metacognitive prompts (Alter et al., 2007; Alter & Oppenheimer, 2008).

So how can we measure the disposition toward types of thinking and biases? We review evidence that *cognitive reflection* examines the activation of the processing systems, *prospect theory* measures risk seeking behaviors, and *optimism bias* is a proxy for intuitive processing.

### Cognitive Reflection

Cognitive reflection is the ability to reflect upon a question before answering and inhibiting the immediate response and is most commonly measured using a form of the Cognitive Reflection Test (CRT; Frederick, 2005). In this article, we take the position that CRT is a useful measure of an individual's propensity to activate System 2 processing in response to a question requiring more deliberate thought, and by virtue of engaging system 2, is also a measure of System 1 suppression (Frederick, 2005).

Specifically, the CRT has been widely deployed as a measure of System 2 engagement, through the suppression of System 1. The CRT presents questions with intuitive, yet incorrect answers and one must reflect on the question to respond correctly. An example of a CRT question is, "A bat and a ball cost \$1.10 in total. The bat costs \$1.00 more than the ball. How much does the ball cost?"<sup>1</sup> The intuitive and incorrect response is the most common answer (Frederick, 2005; Sinayev & Peters, 2015), implying that indeed System 1 thinking is a default mechanism.

CRT has been informative for illuminating decision making in the workplace. Teachers in higher education demonstrated higher CRT being associated with teaching more technology-related materials (Janssen et al., 2019), implying educators with higher reflective thinking are more suited to teaching fields that require high levels of logical thinking (such as information technology). In terms of teaching experience, CRT was associated with increased rationality and greater consideration of future events (Čavojová & Jurkovič, 2017).

To our knowledge, there is no empirical work investigating CRT and software development, however, CRT has been linked with predicting susceptibility to detecting phishing emails (Jones et al., 2019), loss aversion (Frederick, 2005), and forecasting (Moritz et al., 2014).

### Optimism Bias

Bias susceptibility can be used as a proxy to capture the strength of System 1 processing. Unrealistic optimism is a bias where individuals perceive themselves to be less likely to experience negative events compared to others (Sharot, 2011; Weinstein, 1980). People typically exhibit a persistent bias regarding future events, in that they overestimate the likelihood they will experience positive events and underestimate the chances of experiencing negative ones. This bias covers a broad range of events, from life events such as car

accidents to more mundane events, such as estimating the time required to complete a project (Cappos et al., 2014).

For cybersecurity, users have been found to underestimate their risk online, increasing their vulnerability to cybersecurity attacks (West, 2008; Wiederhold, 2014). In a thematic analysis of the same data presented in this article, engineers reported negative events as being damaging to their identity as an engineer (Ivory et al., 2023), indicating that for some, they associate the potential of software vulnerabilities in one's code as being a negative event. As such, a logical assumption would be that software developers are also susceptible to optimistic outlooks toward their work, and underestimating the likelihood of negative events, such as vulnerabilities in their code (as a potential marker of low-quality work).

Optimism in the software development domain has been previously investigated using professionals, with software engineers being worse at estimating time management than those in nontechnical roles (Mølokken & Jørgensen, 2005). Both software engineers and executives experience difficulties in making security-related decisions, demonstrating overconfidence in their software security (Loske et al., 2013). We extend the understanding of optimism bias in a security-specific context by relating this to the language used around software security.

### Risk Aversion

Risk aversion can be linked to prospect theory, which suggests decisions maximize gains and minimize losses (Kahneman & Tversky, 1979). People experience negativity from a loss more strongly than positivity from gains. Accordingly, people typically make choices that minimize loss over maximizing gain (Levy, 1992).

Frederick (2005) reported an association with prospect theory and CRT, showing higher CRT scores accompanied increased risk aversion to losses (i.e., an increased willingness to accept a sure loss than to risk a greater loss). Frederick also showed people make riskier decisions when facing potential gains (willing to risk greater gains than accept sure gains). This supports the idea risk aversion is influenced by heuristics and cognitive biases. Risk aversion has wider generalizability toward, for example, general decision making under risk (Abdellaoui et al., 2007).

Kina et al. (2016) examined software engineer tool adoption practices and found that when new tools are not guaranteed to maximize profit and contain risks, engineers are more risk averse and instead rely upon the known familiarity of familiar tools. This suggests a certainty effect, or bias in play. When examining factors that shape software project decisions, loss magnitude was considered a more significant factor than loss likelihood, highlighting the relevance of choice value (Keil et al., 2000). By measuring risk aversion as a function of risk sensitivity in language used when talking about security, we can examine how security is framed and how the framing manifests in the language used.

### Motivations

There have been calls for more psychological research within computing and software security (Acar et al., 2016; Capretz & Ahmed, 2018), with greater appreciation of the individual involved, and the heterogeneity of behaviors relevant to development roles.

<sup>1</sup> The intuitive response is 10 cents, but the correct answer is 5 cents.

The present research answers these calls by exploring the role of individual cognition and its effect on security perceptions. We address this by investigating the cognitive differences seen in risk sensitivity when talking about security in software. Within software engineering, the role of biases and heuristics has been acknowledged (Chattopadhyay et al., 2020; Petre, 2022), as well as in a systematic review, while noting that research has yet to fully characterize and describe their impact (Mohanani et al., 2020). Psychology is well positioned to respond to this gap through empirical research. Through cognitive measures and the analysis of individual differences, we can apply dual processing theory to better understand how this ultimately explains perceptions of security within software development.

Two different populations were included in our sample, freelance developers, and computer science students, as they represent two different timepoints in the software engineer lifecycle. Students represent early stage, less experienced engineers, and freelancers represent those who have experience from prior projects. The motivation for using both populations is to better understand how security perceptions and risk sensitivity may change (or not change) over the lifecycle of engineers. Increasing awareness of the cognitive features that underpin security perceptions in software across the engineer lifecycle, we are better placed to provide interventions at the most effective points. If risk sensitivity is similar across both populations, then intervening as early as possible may be the solution, whereas if a difference is seen, indicating an event or events altering risk sensitivity, then interventions should look to address these events.

This article is part of a larger project (Ivory, 2022) that examines both cognitive links and risk perception, alongside social identity, and responsibility within software development. The project comprises two independent, self-contained components, with the complementary package focusing on a thematic analysis of responsibility and risk acceptance (Ivory et al., 2023). The overall project information and data can be found at <https://doi.org/10.17605/OSF.IO/P6DY5>.

## Hypotheses

Three hypotheses were created to direct this research:

*Hypothesis 1:* Higher risk aversion scores will be associated with increased awareness and sensitivity to risk in language describing professional work related to software development. Lower risk aversion (higher risk attraction) will be associated with less awareness and sensitivity to risk in language describing professional work.

*Hypothesis 2:* Higher cognitive reflection test scores will be associated with increased awareness and sensitivity to risk in language describing professional work. Lower cognitive reflection (higher risk attraction) will be associated with less awareness and sensitivity to risk in language describing professional work.

*Hypothesis 3:* Mean scores closer to zero on the novel OWASP<sup>2</sup> risk task will be found with higher scores of cognitive reflection. Scores of zero are expected to represent a realistic, unclouded view of risk which should align with more reflective thinking styles.

**Table 1**  
*Reported Ages of Participants Included in the Analysis*

Age	Developer	Student
18–24	28	42
25–34	34	20
35–44	6	8
45–54	1	2

## Methodology

### Participants

As per the preregistration, we required a minimum sample of 122 participants, 61 from each population to meet a desired power level of 80%. A sample of 150 were sought to account for low-quality responses. We recruited 149 participants and excluded data from eight participants; four failed over 50% of attention checks, one preferred not to provide gender information (one observation would not be appropriate to use in analysis to make meaningful implications), and three provided responses of less than 20 words to the questions. Our data corpus thus comprises 141 participants (“software engineers”), 69 of whom were professional freelance software (six females and 63 males) developers and 72 were computer science (CS) students (31 female and 41 male).

All research involved in this project was approved by the University Faculty Ethics committee. Participants provided informed consent before commencing with the research study.

We recruited developers using the freelance website, <https://www.upwork.com/>. We uploaded an advertisement (available on OSF; <https://doi.org/10.17605/OSF.IO/P6DY5>) asking participants to complete a study about their understanding of security in software development. We specified that participants should be currently working in software development, have experience writing secure code and have been involved in noneducation-based software projects. We compensated freelance developers at rate of £10/hr. The student sample was collected from two sources, using internal university mailing lists and the recruitment website, Prolific. We compensated CS students at a rate of £8.50/hr.

Age details can be found in Table 1. Due to data collection issues, nationality data for developers are unavailable, which was substituted with country-level location. Figure 1 shows an approximation of participant location and nationality by presenting continental-level data. Most people do not emigrate from their origin country, with around one in 30 migrating internationally (McAuliffe & Triandafyllidou, 2021). Representing both nationality and location together approximates both data types on a continental scale. Ethnicity and socioeconomic status were not collected in the study design and cannot be reported or discussed.

### Materials

Participants completed the study through the online surveying software Qualtrics. The survey (in study presentation order) included demographic information, an Open Web Application Security Project (OWASP) vulnerability task (OVT), four open-text response questions focused on risk awareness, aversion, and mitigation in

<sup>2</sup> See section OWASP Vulnerability Task for further details.

**Figure 1***Map Displaying Approximated Location/Nationality Data of Participants on a Continent Level*

*Note.* This approximates location and nationality of each participant due to expected immigration movement. “Participant Distribution” by Matthew Ivory, licensed under CC BY 4.0 from <https://doi.org/10.17605/OSF.IO/P6DY5>.

software development, a gambling task and two versions of CRTs (Frederick, 2005; Thomson & Oppenheimer, 2016).

### **OWASP Vulnerability Task**

The OVT is designed to measure unrealistic optimism in software security by leveraging familiar concepts, such as well-known vulnerabilities that are well established and consistently highlighted year-on-year. It can be expected that software engineers will perceive including vulnerabilities in their own code as a more negative event than vulnerabilities created by other developers. The OVT builds upon the finding that individuals tend to underestimate the likelihood of negative events affecting them (Sharot, 2011; Weinstein, 1980). While previous cybersecurity research has focused on users and their optimism (West, 2008; Wiederhold, 2014), we devised a measure that is specific for software developers who engage in secure coding. The OVT is a measure of comparative optimism between an individual’s estimation of their own likelihood of including security vulnerabilities compared to an average developer.

The measure uses the 2021 top five vulnerabilities: Injection flaws, Broken Authentication, Sensitive Data Exposure, XML External Entity flaws, and Broken Access Control.<sup>3</sup> The top five vulnerabilities were used as opposed to the full list of 10 as less experienced developers are more likely to be familiar with these vulnerabilities.

The OVT consists of two parts, separated by a secondary task, and presented in a randomized order to reduce recall effects. The first asks participants to estimate the percentage likelihood of a specific vulnerability existing in web applications released by the “average developer.” The second part asks respondents the likelihood of themselves introducing these vulnerabilities into their own code. Similar measures have been used previously, asking participants for two measures of success, with the first being about personal confidence in correctly answering a question, followed by a second asking about the percentage of other people who would answer correctly (De Neys et al., 2013; Frederick, 2005; Hoover & Healy, 2019).

### **Perceptions of Software Security**

The qualitative questions consisted of four items asking participants about their experiences, thoughts, and opinions on security within software development. Participants had the option to write answers or record audio responses using Phonic (<https://www.phonic.ai>). Participants were asked to write for at least 4 min or speak for 2 min per question. The questions asked are shown in Table 2.

### **Risk Aversion**

The gambling exercise replicated that used by Frederick (2005). Thirteen questions sought preference between two options of gaining or losing money. Eight questions contrasted gain scenarios (“Gain £100 for sure or a 90% chance of £500”), and five contrasted loss scenarios (“Lose £50 for sure or a 10% chance to lose £800”). All questions were presented in a randomized order.

### **Cognitive Reflection**

We employed two CRTs. The original CRT (Frederick, 2005), and the CRT-2 (Thomson & Oppenheimer, 2016), an alternate version designed both to reduce the numerical nature of the questions and address floor effects. We reworded question texts, altering names and values to mitigate attempts to answer questions through an online search for a matching string. The core principle of each question was not modified. An example is changing the word “bat” and “ball” for “article clip” and “elastic band,” respectively. We checked whether participants had seen or answered CRT questions before, although previous research (Bialek & Pennycook, 2018; Stagnaro et al., 2018) suggests scores are stable across repeat exposure. We recorded response times to the CRT questions and

<sup>3</sup> Definitions can be found on the OWASP top ten list—<https://owasp.org/www-project-top-ten/>.

**Table 2**  
*Qualitative Questions Presented to Participants in the Order Listed*

Number	Question
1	Describe a time when you successfully developed and released/launched a software project, either in a professional or personal capacity. This could either be a recent example, or perhaps a project you were particularly proud/happy with. Please include information concerning the purpose of the project and how important security was during development.
2	When considering the process of developing and launching software/web applications, what is at risk of potentially going wrong and how could these risks affect you? You should consider the size or the significance of the potential factors that may go wrong and how this may affect you (e.g., risk of functional failure, financial losses, damage to reputation, etc.)
3	If you were to consider software development as a series of ‘gamblers’ (decisions that confer possible risk), what gambles would be considered worthwhile or worth a risk during the process of developing software? Why? These gambles may be considered from both an individual perspective and as a team. Both decisions that you take individually, or decisions that are enforced by policy, should be considered.
4	What approaches or considerations, do you, or your team, take when aiming to identify potential risks or security vulnerabilities when developing software? What is the reasoning behind these decisions? You should consider the decisions and thought processes behind selecting certain tools (such as static analysis tools), as well as identifying specific tools.

*Note.* Participants were asked to respond to these with either a written response or to record an audio response. Participants were requested to write for at least 4 min or speak for 2 min per question.

instructed participants to complete the CRT questions as fast as possible to reduce internet searching.

### Procedure

Participants were presented through Qualtrics with an information sheet that described aims and intentions, following which they gave their informed consent. Participants then answered demographic questions, completed the first OVT for the average software engineers, split by the qualitative questions, then completed the second OVT. This was followed by the gambling task and the CRT measures. Finally, participants were presented with a debrief sheet

that provided further information, references, and contact details for the researchers. The survey flow is shown in Figure 2.

### Research Design

The study comprises a between-participants observational survey design grouped by a two-level factor of population (freelancer vs. student). Data from two independent populations were collected and contrasted. Data validity can be seen in the OSF repository additional online materials.

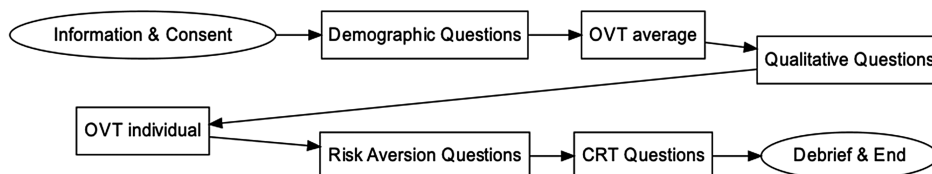
### Data Operationalization

Question responses were operationalized through content analysis. As participants had been asked to write for at least 4 min or speak for 2 min per question, we did not expect short responses. The average response wordcount for developers were on average 81.09 words long ( $SD = 61.71$ ), and the students’ responses were longer at 94.38 words ( $SD = 38.38$ ). As a preregistration deviation, we removed responses of 20 word or less (forming a proxy of low quality) as they were less than one standard deviation from the average response (for developers). Of the short responses, 38 were removed in total, from 22 participants. Two participants (both freelancers) had all responses removed, two participants had three responses omitted, six had two responses removed, and 12 participants had only one response removed.

Responses were unnested into single words, words that appeared five or fewer times were removed, remaining words were stemmed to their shortest form (e.g., “secured” and “secure” were stemmed to “secur”). Stopwords (“and,” “because,” “though”) were removed to reduce noise. Then using the Computer Science Academic Vocabulary List (Roesler, 2020), words were tagged as computer science/software relevant. The remaining words were manually tagged for topics of uncertainty, software products, workplace specific language, finance, geographic references, legal, and shallow skills. Following coding, all tagged words were reviewed for appropriateness, and discussed among the research team. Two independent researchers were given a sample of all the retained words and asked to categorize them using the finalized tags, resulting in 20% of all words being reviewed independently (each researcher reviewing 10%).

Interrater reliability was initially assessed through Cohen’s  $\kappa$  (Cohen, 1960). Yet, this measure does not perform well when handling imbalances in the marginal distributions of confusion matrices (Warrens, 2014). In the present case of assessing the reliability of topic coding, the distributions are greatly skewed in

**Figure 2**  
*Survey Flow Representing the Presentation Order of Measures Given to Participants*



*Note.* All participants received the same order of measures with randomization within the measures (detailed in text where appropriate). OVT = Open Web Application Security Project vulnerability task; CRT = Cognitive Reflection Test.



favor of negative ratings from researchers. Consequently, small deviations from the positive-positive ratings greatly affect the value of Cohen's  $\kappa$ . Since percentage of raw agreement is not reliant on marginal distributions (von Eye & von Eye, 2008) we use this as a preferred and complementary measure of interrater reliability.

To calculate  $\kappa$  and percentage agreements, confusion matrices were constructed with the primary researcher's ratings against the combined ratings of the secondary researchers. As the secondary researchers rated separate sections of the data, there was no overlap between the words and so were combined. From this the percentage of both parties identifying a word belonging to a topic or not provided the interrater reliability. We report the raw percentage agreements in Table 3 as it provides a coarse-grade assessment of potential bias, and we provide the data and the analysis code in the online repository at <https://doi.org/10.17605/OSF.IO/SJ8BT> (Ivory et al., 2022), providing full transparency of data reliability.

Following the reliability assessment, proportional variables were then calculated through topic occurrence divided by word count, creating the variable "proportion of uncertainty-related language" (PURL), as well as proportion of computer science language. This resulted in a value of topic proportion for each question per participant.

### Data Transformations

We used data transformations to prepare variables for suitable analysis. We scaled continuous variables, CRT, OVT, Gambling scores, PURL, and proportion of computer science language between 0 and 1. We transformed variables with substantially nonnormal distributions to improve normality (see Table 4): OVT scores and PURL scores (with values of zero removed—see the Analytic Strategy section). These transformations are a deviation from our preregistration, and for transparency, additional online materials are available at <https://doi.org/10.17605/OSF.IO/SJ8BT> that detail the reasons for requiring these transformations. In short, models that used the untransformed data suffered violations of model assumptions and were less effective than models that used the transformed data.

**Table 3**

*The Raw Percentage Agreement Between the Primary Researcher and the Two Researchers Who Completed the Validation Task*

Topic	Agreement
Risk	.87
Product	.88
Workplace	.86
Finance	.91
Geographic	.97
Legal	.91
Shallow skills	.94

*Note.* The two researchers both completed a separate 10% of the words available and so were combined allowing for the creation of Cohen's  $\kappa$  between the primary researcher and the two researchers combined. Agreement is calculated through the confusion matrix and is the sum of both the primary and secondary researchers rating a word as true or false, divided by the sum total of words.

**Table 4**

*Transformation of Optimism Task and PURL Scores Toward a Normal Distribution*

Measure	Transform power	Shapiro–Wilks		<i>p</i> value	
		Original	Updated	Original	Updated
OVT	1.425	.96	.98	.006	.036
PURL	.05	.75	.99	<.001	.013

*Note.* Transform power is the optimal value identified through the Shapiro–Wilks test provided through using Tukey ladder of powers. The ladder applies a range of power transformations and reassesses the data for heteroscedasticity. The values listed under the Shapiro–Wilks section reflect the value before and after transformation, and the *p* value columns show the pre- and postvalues from the Shapiro–Wilks test. OVT = Open Web Application Security Project vulnerability task; PURL = proportion of uncertainty-related language.

### Analysis

Analysis was conducted in R (Version 4.1.0). Data, analysis scripts, and instructions for reproducing the results seen in this article can be found in the OSF repository here at <https://doi.org/10.17605/OSF.IO/SJ8BT>.

### Analytic Strategy

To address hypotheses one and two, a two-step model was employed due to the zero-inflated nature of PURL induced through a floor effect (i.e., not all responses used language around uncertainty). The first step was a logistic mixed-effect regression model to determine the existence of uncertainty-related language based on other variables. The second step was a linear mixed-effect regression model to understand the impact of cognition on PURL. For addressing Hypothesis 3, a linear model was built to compare vulnerability scores by cognitive reflection.

Mixed-effect models were used as these handle nested data appropriately. As each participant provided four different text responses, it is important to treat these as independent to each other, but dependent on the participant. This allows for greater model fit, and accounts for participant-level variation in PURL as well as at the question level.

## Results

### Addressing Hypotheses 1 and 2: Individual Differences in Risk and Reflection in Software

To test Hypotheses 1 (that generalizable risk perspectives increase risk sensitivity) and 2 (that generalizable reflective decision-making increases risk sensitivity), a two-step or hurdle model, was developed to first identify the presence of uncertainty-related language, followed by the second model for determining PURL, including risk aversion (as operationalized through gambling task scores), and cognitive reflection (operationalized through CRT scores). Hypothesis 1 is assessed through the inclusion of the loss aversion scores in ensuing models, and Hypothesis 2 is assessed through the inclusion of CRT terms in the models.

In the two-step model, the first model represents a hurdle to be passed, which determines whether uncertainty is reflected in the text. If uncertainty is present and the hurdle is "cleared," the second

model provides predictive power for PURL in the zero-truncated data. Exploratory results that were not preregistered are included in the additional online materials found on OSF.

### Step 1

The first step was a binary mixed-effect logistic regression model to identify the presence of uncertainty-related language. Not all sentences written by participants included uncertainty, and so PURL distribution was zero-dominated. This model presents the first hurdle, and if the threshold of zero language is crossed, then data are subject to the second step.

This model was developed using a forward-step approach starting with the null model to model the existence of uncertainty language within responses. For model refinement, outliers were identified using Cook's distance, which measures the influence specific datapoints have over the model fit. Items with a value three times greater than the mean of all the distances were flagged and removed. The final model, with coefficients seen in Table 5, had an Akaike information criterion of 2515.00, explained variance of  $R^2_{\text{conditional}} = .23$  and correct classification of 68.13% of data.

### Step 2

The second step focused on data containing PURL. This enabled a mixed-effects linear regression model for predicting the amount of PURL in a response. The dependent variable, PURL was zero-truncated, assessed for normality, and transformed toward a more normal distribution. Figure 3 shows the zero-truncated distribution, and the subsequent transformed distribution. The nontransformed zero-truncated data's Shapiro–Wilks value = .75,  $p < .001$ , and the transformed data,  $W = .99$ ,  $p = .013$ . While still nonnormal, it is less right-skewed than the untransformed data. Appendix provides examples of how PURL manifested in responses.

Model building began with the null model with terms added sequentially. A mixed-effect model was chosen due to the data's nested structure. Each independent datapoint indicated a sentence

**Table 5**

*Model Coefficients of the Terms Used in the Modelling of the Presence of Uncertainty-Related Language*

Term	$\beta$	Significance
Intercept	−1.35	***
CS prop	8.39	***
Question 2	1.91	***
Question 3	1.84	***
Question 4	0.96	***
Question 2: CSprop	−5.36	**
Question 3: CSprop	−2.6	
Question 4: CSprop (1 participant)	−0.35	

*Note.* The beta value reflects the model coefficient and is given alongside the significance of the term in the model. The terms relating to Questions 2, 3, and 4 are present through the inclusion of a categorical term of question, this is included as the questions ask about different ideas, and likely vary in frequency of PURL language. That is, the terms Questions 2, 3, and 4 represent the variability in the presence of these different questions. The intercept represents the first question, and so the question terms are interpreted in comparison to the first question, the model tells us that for Questions 2, 3, and 4, their beta values are positive compared to the first question indicating a higher likelihood of the presence of uncertainty-related language. PURL = proportion of uncertainty-related language; CSprop = proportion of Computer Science-related language. \* $p < .05$ . \*\* $p < .01$ . \*\*\* $p < .001$ .

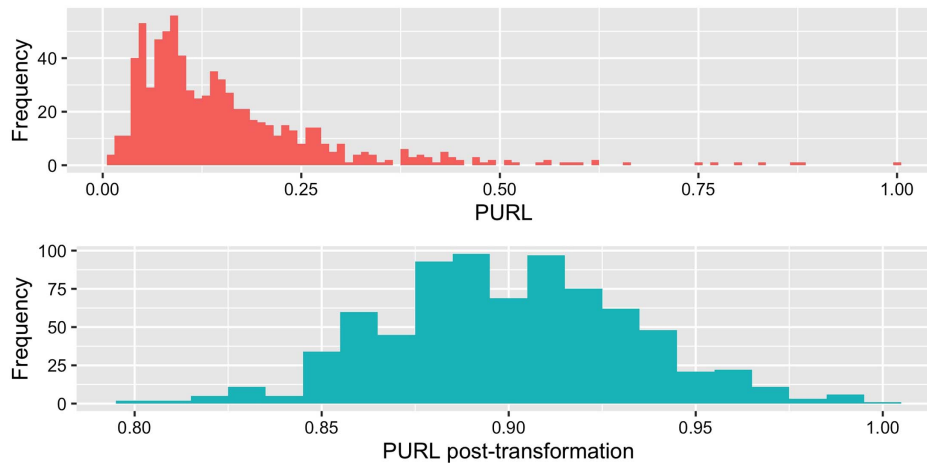
nested within a response, which were nested under participants. For this, a random intercept of participant was included, with a random slope of question to allow for individual differences in language use.

Hat values were used to identify high leverage points, which are datapoints that measure the distance from the observed and fitted values, higher values indicate higher leverage, and we examined values two times greater than the mean hat value. Little relationship was seen, and removal provided negligible effect on the model. Removing influential outliers through Cook's distance had little effect on the model and so all items were retained.

The final model can be seen in Table 6 and reports an  $R^2_{\text{conditional}}$  of .66. The absence of risk aversion scores in the final model indicates

**Figure 3**

*Distribution of the PURL Scores Before and After Transformation Toward a Normal Distribution*



*Note.* The contrasting plots provide a visualization of the difference in distribution made from the transformations. PURL = proportion of uncertainty-related language.

**Table 6**  
Chosen Model for the Second Step Model Using Zero-Truncated PURL as a Dependent Variable

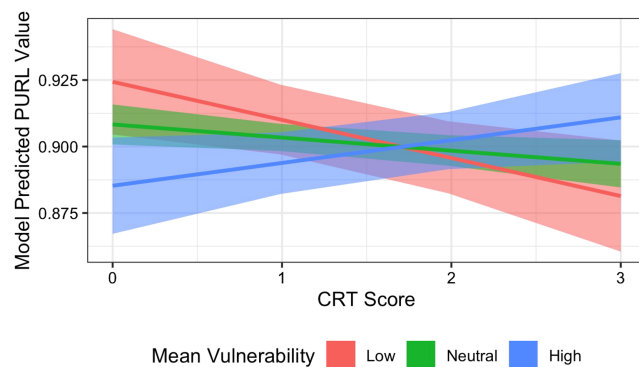
Term	$\beta$	Variance	Significance
Intercept	0.91		***
CRT	-0.04		**
OVT	-0.04		*
CRT $\times$ OVT	0.07		*
Question 2	0.01		***
Question 3	0.01		***
Question 4	0.02		***
Random (Question 1)		.0006	
Random (Question 2)		.0006	
Random (Question 3)		.0006	
Random (Question 4)		.0006	

*Note.* The terms listed are those present in the model for determining PURL in responses, with coefficient values listed in the beta column, along with confidence intervals in the CI column. The variance for the random effects is given providing a value for how much variation these terms provide and finally, the significance of each term for the model is given. The terms relating to the Questions 2, 3, and 4 are present through the inclusion of a categorical term of question, this is included as the questions ask about different ideas, and likely vary in frequency of PURL language. OVT = Open Web Application Security Project vulnerability task; PURL = proportion of uncertainty-related language; CRT = Cognitive Reflection Test; CI = confidence interval.

\* $p < .05$ . \*\* $p < .01$ . \*\*\* $p < .001$ .

no support for Hypothesis 1, and the inclusion of both a CRT term and CRT interaction with optimism supports Hypothesis 2. The model effects are shown in Figure 4, which shows predicted PURL levels for varying strengths of optimism bias. The figure shows a distinct interaction between the optimism bias when measured alongside cognitive reflection. The three levels of OVT presented (low, neutral, and high) serve solely as a visualization aid and not actual groupings made in the analysis. When cognitive reflection was low, PURL was also low, however, as the more optimistic

**Figure 4**  
Interaction Plot Between CRT Score (the Propensity for Reflective Thinking) OVT Score (Optimism Bias) and Model Predicted Values



*Note.* To provide easier interpretation of the interaction, the plot presents three “groups”—of low, neutral, and high OVT scores. In reality, this grouping does not exist and is a continuum, but the grouping represents the extremes effectively. OVT = Open Web Application Security Project vulnerability task; PURL = proportion of uncertainty-related language; CRT = Cognitive Reflection Test.

participants increased in their reflection, they spoke more to uncertainty when discussing software. Those displaying an average, or pessimistic view on vulnerability, tended to speak more about uncertainty intuitively, but cognitive reflection suppressed this language, and in extreme pessimism suppressed language to similar levels of the overly optimistic and intuitive.

### Addressing Hypothesis 3

To test for the effects of optimism bias in software development, first a simple linear regression compared OVT scores between students ( $M = 16.72$ ) and developers ( $M = 18.10$ ) which was found to be nonsignificant,  $F(1, 139) = .12, p = .727, R^2_{\text{adjusted}} = -.01$ . To test whether software engineers demonstrate a general optimism bias to vulnerability inclusions, a one-sample  $t$  test was conducted against a true mean of 0. It was found that the mean score calculated across both developers and CS students, 17.40, 95% CI [13.51, 21.28], was significantly different from an average of zero,  $t(140) = 8.86, p < .001$ .

To test Hypothesis 3, a linear regression model was used to predict OVT scores using both CRT measures. A series of models were built to identify the most parsimonious model using Akaike information criterion as the suitability criterion. No model was constructed that provided a significantly better fit than the null model, indicating no strong relationship between OVT scores and cognitive reflection scores. Consequently, we do not reject Hypothesis 3’s null hypothesis that there is no relationship between OVT scores and CRT.

### Discussion

We asked whether domain-general, psychological measures of cognitive reflection, risk aversion, alongside optimism bias for security vulnerabilities would predict how software engineers talk about risk perception when discussing their software security. Using data collected from freelance software developers and CS students, we analyzed PURL through a mixed-effects hurdle model. The key finding was an interaction between cognitive reflection and unrealistic optimism for PURL. Also, engineers were typically overly = optimistic about personal susceptibility toward security vulnerabilities. Meanwhile, risk aversion was not a strong indicator of PURL, nor did we find a systematic relationship between cognitive reflection and optimism.

### Answering the Call for Increased Psychology in Software Engineering Research

This article answers calls for increased psychology-based research in software development (Acar et al., 2016; Capretz & Ahmed, 2018) by showing that software engineers are subject to the same cognitive constraints as nonengineers. The language used by engineers talking about secure coding can be usefully framed with respect to dual-processing theory.

Additionally, a systematic review (Mohanani et al., 2020) within software engineering points toward the breadth of biases affecting software (Fagerholm et al., 2022; Ralph, 2011). Our quantitative, data-driven study extends the understanding of heuristics and biases for secure software. Moreover, the present study underscores the potential for different cognitive constructs to interact in the software domain, revealing the complexity of how heuristics can shape software risk.

## Risk Aversion

Prospect theory proposes that people are more sensitive to losses over gains (Frederick, 2005; Kahneman & Tversky, 1979). As a group, software engineers show this asymmetry too, yet we did not find evidence to support Hypothesis 1 that greater risk aversion would associate with greater risk sensitivity in language. There are two interpretations we present that may explain this finding.

One interpretation is that engineers do not systematically frame security decisions in terms of gains or losses. Software security is often made up of smaller, independent decisions, each with their own framing and outcome value. Typically, software engineers perceive security as a barrier (Lopez et al., 2022), or lower priority than functionality (Lopez, Sharp, et al., 2019), but this may not translate well into mental models of software, meaning that security is poorly considered in terms of risk aversion. It could also be interpreted that risk aversion is unrelated to risk taking behavior as there are other factors that determine the behavior. In economic research, it is seen that low risk aversion can lead to riskier trading decisions (Hoffmann et al., 2015), which contrasts with our expected findings. Further research and a better understanding of the link between software developers' language and risk behavior is warranted to better understand this finding.

## Cognitive Reflection

The lack of evidence for a direct relationship between CRT and risk sensitivity might reflect (a) the study design, (b) the lack of sensitivity of CRT to capture relevant individual differences, or (c) a more complex relationship than a bivariate link. We believe the evidence points to (c) because we obtained an interaction between CRT and optimism for risk sensitivity. Moreover, this interaction negates the likelihood that (a) or (b) are wholly satisfactory.

The implication is that the specific simple and direct relationship between cognitive reflection and uncertainty language is weak, and a software engineer's ability to reflect on decisions does not manifest in the language used concerning security. Fine-tuning either—or both—of these constructs might enable stronger evidence to emerge of their link. However, we did find an indirect relationship between CRT cognitive reflection and risk sensitivity in language, one that supports the broad thrust of the initial prediction.

## Optimism

Experienced freelancers and CS students collectively demonstrated a significantly greater optimistic belief in their own secure coding behaviors (see Rhee et al., 2012; Weinstein, 1980) with no noticeable differences between the populations. Measuring unrealistic optimism can serve as a proxy of System 1 processing, as the presence of one bias indicates an increased likelihood of other biases being present too (Ralph, 2011). Insofar as both developers and CS students reported similar levels of optimism, the present research supports the findings of Oliveira et al. (2018) and Brun et al. (2022) that experience does not reduce susceptibility in biases in vulnerability detection.

Observing optimism bias in both new and experienced software engineers reinforces our conclusion that this is an intuitive, instinctive perception. It is neither simply a naïve aspiration nor a survivor bias.

## Interaction: Cognitive Reflection and Optimism

The second step in the hurdle model focused on relationships between cognition and risk sensitivity. The key finding was an interaction between cognitive reflection and optimism that provided predictive power for uncertainty-related language. Those who displayed high optimism spoke less frequently about uncertainty, but as cognitive reflection increased, so did uncertainty. Conversely, pessimistic software engineers frequently mentioned uncertainty, but decreased in uncertainty as cognitive reflection increased. In other words, the way that engineers talk about uncertainty and security is dependent on multiple facets of cognition. This provides some support for Hypothesis 2 but not in its entirety.

The finding that cognitive reflection or optimism alone explain very little variance in security perceptions highlights the entangled nature of cognition in the real-world. Despite cognitive reflection and optimism being two clearly defined measures of cognition within the psychological literature, in data collected from a complex domain, we see that these aspects of cognition are linked and cannot be considered totally modular.

Those naturally optimistic about security, and who deploy System 1 thinking, are less likely to discuss uncertainty—perhaps because it is not seen as an issue. Indeed, Assal and Chiasson (2018) found that overly optimistic software engineers view additional security implementation as holding minimal value, as they believe their current level of security to be sufficient. When considering security, therefore, they may require stronger cues or framing to activate System 2 when making security decisions. Meanwhile, those naturally pessimistic about security, talked more about uncertainty and here System 2 thinking is associated with increased uncertainty (Siegrist et al., 2005).

Reduced self-confidence may mean engineers feel overwhelmed when presented with security decisions, which is not helped by a lack of documentation supporting their choices (Acar et al., 2017) or lone working environments (van der Linden et al., 2020). By activating System 2 processing and thinking critically about security decisions, the naturally pessimistic may find increased confidence in their abilities to code securely. One catalyst for System 2 thinking during secure decision making might be peer communication (Shreeve et al., 2022), which allows for the balancing of perspectives and a reduction of biases. Both examples of the naturally optimistic and pessimistic aligns with the dual-processing theory (Evans, 2003), suggesting that software engineers who can easily activate System 2 processing will experience reduced biased judgment during secure decision making.

A practical ramification of this interaction is that discussions within development teams may help to improve security decisions. While developers working alone do not have access to such support, engagement with interactive websites—such as Stack Overflow—can provide a digital community to engage with (Lopez, Tun, et al., 2019) beyond a simple question-and-answer format. Encouraging developers to interact even asynchronously can potentially provide a community that motivates more reflective thinking. Developing an understanding of how social identity affects these communities and increases engagement between peers can help on this front.

## Rewarding Secure Behaviors

Our findings have implications for how software engineer freelancers (as the focus of this research) are rewarded for practicing



secure coding. The implications are also relevant for engineers working in more permanent roles within a company. Rewards can be used as an incentive to encourage certain behaviors and reduce unwanted ones. Rewards can be intrinsic or extrinsic; intrinsic rewards are those internal to the individual and are inherently found in the task itself and upon completion, and extrinsic rewards are external to the task, such as pay or recognition (Ajila & Abiola, 2004).

Intrinsic rewards can boost task performance with motivation also playing a mediating role (Manzoor et al., 2021). Intrinsic rewards can stem from expressions of appreciations by senior employees or promotions based on work quality, and so in nonfreelance engineering roles, this is easily managed by ensuring that employees are recognized for their secure coding practices. Motivation for accuracy or quality can lead to a greater expenditure of cognitive effort (Kunda, 1990), and these motivations can be achieved through an expectation to defend or explain one's decisions (Tetlock & Kim, 1987).

Extrinsic rewards are not as straightforward for freelance engineers as they can be for contracted engineers, as the companies may be less inclined to provide incentives for secure code if they do not see freelancers as part of the company. One way to provide an extrinsic reward is through the recruitment platforms that they use. If freelancers can be publicly recognized for their secure coding practices in the previous work, this may be seen as a potential reward for quality work as it boosts their profile, increasing potential work offers. The exact nature of verifying secure coding practices on freelance platforms is beyond the scope of this article, but if a system can be devised that is universally trusted and easily implementable, then the use of gamification may work as a reward system. Gamification methods have been shown to have long-term behavioral effects (Hamari, 2017), but gamification should be done with caution as not all findings support their efficacy (Barreto & França, 2021). If badges are perceived by platform-users meaningful and trusted, these may act as a reward system that can be used for freelancers.

The use of rewards, either intrinsic or extrinsic, for encouraging secure coding behaviors can be used to increase motivation and subsequently increase task performance. In a secure coding specific context, by rewarding secure behaviors, this can be linked to greater cognitive engagement, and through the dual-processing theory, lead to reduced bias interference. In the interaction seen between cognitive reflection and optimism, by increasing the likelihood of engaged System 2 processing, clients, stakeholders, and employees can focus on managing optimism rather than juggling two aspects of cognition at a time.

### A Comparison Between Professionals and Students

The finding that population was not a significant term in the modeling process supports the assumptions of our key take-home message, that the data offer credibility for using CS samples in future research in place of professional developers.

The persistence of biases and similar approaches to uncertainty around security implies these are general and widespread characteristics, rather than something unique to students that is modified by experience. This signals that psychological interventions may have an effect across the software engineering domain, and not limited to inexperienced or early stage developers. It also shows that individuals who may be predisposed toward more intuitive, impulsive modes of

decision making are not impervious to learning, or seeking support from their work environment to ensure that they make appropriate choices at key points during projects.

Acknowledging that professional developer populations can be harder to access, easier access to a population whose only significant difference is experience provides momentum to further research projects. CS students provide an alternative, cheaper, and easier way to explore hypotheses before confirming findings in professional developers.

Insofar as we proposed novel quantitative hypotheses that addressed general psychological constructs, we have no specific reason to expect that participant diversity (e.g., with respect to the age, gender, and location) would affect performance. Moreover, the profile of these variables broadly corresponds with those of the community—most of our participants were aged 18 and 35 for both groups, which aligns with the Stack Overflow 2022 (<https://survey.stackoverflow.co/2022>) results that reported over 63% of respondents were aged between 18 and 34. Similarly, the gender split reported by our sample was 74% male compared to Stack Overflow's reported 92% male audience. This indicates either a potential shift in the future gender split of software developers, with more female developers currently learning to develop software, or it reflects a potential gender split in survey response propensity with more females choosing to participate in nonstandard freelance work offers. The sample was globally distributed representing the global, diverse nature of software development.

### Reflecting on Heuristics

Our research is built upon the theoretical foundations of dual-processing theory and heuristics. We define heuristics as “mental shortcuts” that can be used in multiple scenarios to simplify and speedup decisions (Beike & Sherman, 1994), but when heuristics do not provide appropriate decisions for certain contexts, then they become biases (Gigerenzer, 2015; Kahneman et al., 1974).

Two main schools of heuristics exist, both of which share commonalities but deviate on other aspects. For a more comprehensive discussion over the major differences, the reader is referred to Hjeij and Vilks (2023) and Samuels et al. (2012). The first school of thought is that proposed by Daniel Kahneman and Amos Tversky in the 1970s (Kahneman et al., 1974), who posit that heuristics are evolutionary mechanisms that use generalizations or rules-of-thumb to reduce cognitive load. They suggest that while they help us make quick decisions, they are often inaccurate. More importantly, we use heuristics even when there is little guarantee they will produce a correct answer. In short, Kahneman and Tversky suggest that human decision making is largely *irrational*.

The second school is that of bounded rationality started by Herbert Simon and continued by Gigerenzer (2015). One of the main contributions is the *adaptive toolbox* which defines heuristics as efficient processes which ignore information in favor of speed, but with one significant difference, that these heuristics are often ecologically *rational* and can provide answers that are nearly, or as-good as optimized decision-making mechanisms. Part of Gigerenzer's toolbox was the concept of fast and frugal heuristics, where less information is more (Gigerenzer, 2008).

The difference between the two schools can be distilled into a difference in research focus. Kahneman and Tversky examined the *decisions* that people make (Kahneman & Klein, 2009),

whereas Gigerenzer examines *cognitive mechanisms* (Gigerenzer & Gaissmaier, 2011). Both agree that if a heuristic is ill-fitted to the context, then they are biased and can be considered irrational. In our present research, we are examining the context of secure coding, for which no heuristic has evolved to handle (Oliveira et al., 2018), and so whether you subscribe primarily to Kahneman and Tversky's definition of heuristics, or Gigerenzer's, both approaches support the idea that in this context, intuitive, heuristic-based decisions are likely to produce incorrect and irrational decisions.

## Limitations

In this section, we draw attention to two domains of limitations: internal validity in terms of sampling and statistics, and external validity in terms of construct accuracy.

With respect to the former, we recognize that the work draws on self-reports of risk. We knew that asking participants about their experiences with developing software in the context of risk and uncertainty might prime security. All participants completed the same survey, and any priming effect would be across all participants. A different issue is that some participants responded to question prompts in a superficial or terse way. We asked respondents to write for at least 4 min or speak for 2 min per each qualitative, text-response question. Yet some responses were only three words long, limiting data quality. Typically, when this occurred, it was across all a participant's text responses rather than specific questions. Drawing on a freelance online marketplace, we may have recruited participants who are focused on completing tasks quickly to maximize their hourly pay. Our preregistered sampling strategy was designed to mitigate data quality differences, proportions of language were used to ensure a relative association between response length and topic frequencies, and only responses longer than 20 words were analyzed. Nonetheless, further work that elicits richer data would be useful.

In the statistical (hurdle) model, the second step produced increased residuals in predicting high PURL, due to scarcer datapoints in the higher ranges. This could be a signal of a violation to the underlying distribution, however, Schielzeth et al. (2020) concluded mixed-effects models are relatively robust to such issues. We suggest this remains an appropriate analytic device, while welcoming any opportunity to complement these findings with convergent statistical approaches.

To measure optimism, we used a novel, software developer specific task, the OVT. The intention was to measure relative optimism between an individual and the "average" developer. Official statistics for the percentage of software applications suffering from specific OWASP vulnerabilities do not exist and so an absolute measure of optimism is unobtainable. By measuring relative optimism, we elicit information on domain-specific security concepts. The measure consists of two sections, one about personal risk and the other of the average developer. This was presented to participants in a fixed order which is like previous work using self-reported ratings of confidence (De Neys et al., 2013; Frederick, 2005; Hoover & Healy, 2019). Simplicity was valued and took research precedence in this regard, though of course it remains an open question as to whether different responses could be obtained through alternating section presentation. Note that a mixed order would have required additional model analytic terms, potentially resulting in overfitting, as well as reducing the analytic focus on the other cognitive measures.

Many studies have used CRT to understand individual differences in cognition. The first of two recurrent concerns is the extent to which performance is confounded by numerical ability. Although CRT associates strongly with numeracy (Liberali et al., 2012; Sinayev & Peters, 2015; Welsh et al., 2013), CRT responses have been shown to measure more than numeracy alone, as people tend to respond with a predictable intuitive and incorrect response (Pennycook & Ross, 2016). A second concern is interpretive; the attribution substitution hypothesis suggests that when using System 1 processing individuals unconsciously substitute complex decisions with computationally simpler ones (Hoover & Healy, 2019, 2021; Kahneman & Frederick, 2002). For our purposes, however, we simply note that showing the systematicity of the association between CRT scores and software cognition is the key first step. Disentangling the nuances of how conscious system switching or question framing might affect conceptual interpretation—of specific CRT questions—is left for more detailed enquiry.

## Future Work

Our future research will extend beyond the research described here, by leveraging preexisting paradigms (Brun et al., 2022) for measuring blindspots within application programming interfaces. By using a paradigm that reflects real-world scenarios, involving code reviews with insecure elements, measures of developers' actions can be taken to associate with cognition. Previous work using this paradigm has not fully explored the relationships with heuristics and dual-processing theory which we anticipate being able to draw clear connections between. This will further evidence the need to examine software security through the lens of psychology to better understand the effect of cognition and individual differences in secure software development.

## Conclusion

Understanding how software engineers talk about risk and security in software is important as it provides insight into how they approach software development. The main finding from this study is that an interaction between cognitive reflection and optimism goes some way to explain risk sensitivity in language used by engineers when discussing software development, risk, and security. It was also seen that software developers and CS do not differ significantly in their approaches to security and risk. Additionally, software developers and CS both exhibit optimistic perceptions on their likelihood to include vulnerabilities in their own software. Future research should expand upon this work by performing similar measures alongside software development tasks.

## References

- Abdellaoui, M., Bleichrodt, H., & Paraschiv, C. (2007). Loss aversion under prospect theory: A parameter-free measurement. *Management Science*, 53(10), 1659–1674. <https://doi.org/10.1287/mnsc.1070.0711>
- Acar, Y., Fahl, S., & Mazurek, M. L. (2016). *You are not your developer, either: A research agenda for usable security and privacy research beyond end users* [Conference session]. IEEE Cybersecurity Development, Boston, MA, United States. <https://main.sec.uni-hannover.de/publications/conf-secdev-acarfm16/>
- Acar, Y., Stransky, C., Wermke, D., Weir, C., Mazurek, M. L., & Fahl, S. (2017). *Developers need support, too: A survey of security advice for*

- software developers [Conference session]. 2017 IEEE Cybersecurity Development (SecDev), Cambridge, MA, United States. <https://doi.org/10.1109/SecDev.2017.17>
- Ajila, C., & Abiola, A. (2004). Influence of rewards on workers performance in an organization. *Journal of Social Sciences*, 8(1), 7–12. <https://doi.org/10.1080/09718923.2004.11892397>
- Alter, A. L., & Oppenheimer, D. M. (2008). Effects of fluency on psychological distance and mental construal (or why New York is a large city, but New York is a civilized jungle). *Psychological Science*, 19(2), 161–167. <https://doi.org/10.1111/j.1467-9280.2008.02062.x>
- Alter, A. L., Oppenheimer, D. M., Epley, N., & Eyre, R. N. (2007). Overcoming intuition: Metacognitive difficulty activates analytic reasoning. *Journal of Experimental Psychology: General*, 136(4), 569–576. <https://doi.org/10.1037/0096-3445.136.4.569>
- Assal, H., & Chiasson, S. (2018). *Motivations and amotivations for software security* [Conference session]. Fourteenth symposium on usable privacy and security (Vol. 4).
- Barreto, C. F., & França, C. (2021). *Gamification in software engineering: A literature review* [Conference session]. 2021 IEEE/ACM 13th international workshop on cooperative and human aspects of software engineering (CHASE), Madrid, Spain. <https://doi.org/10.1109/CHASE52884.2021.00020>
- Beike, D. R., & Sherman, S. J. (1994). Social inference; inductions, deductions, and analogies. In R. S. Wyer & T. K. Srull (Eds.), *Handbook of social cognition* (2nd ed., pp. 209–285) Lawrence Erlbaum.
- Bialek, M., & Pennycook, G. (2018). The cognitive reflection test is robust to multiple exposures. *Behavior Research Methods*, 50(5), 1953–1959. <https://doi.org/10.3758/s13428-017-0963-x>
- Brun, Y., Lin, T., Somerville, J. E., Myers, E. M., & Ebner, N. C. (2022). Blindspots in Python and Java APIs result in vulnerable code. *ACM Transactions on Software Engineering and Methodology*, 32(3), 1–31. <https://doi.org/10.1145/3571850>
- Cappos, J., Zhuang, Y., Oliveira, D. S., Rosenthal, M., & Yeh, K.-C. (2014). *Vulnerabilities as blind spots in developer's heuristic-based decision-making processes* [Conference session]. 2014 workshop on new security paradigms workshop—NSPW '14, Victoria, British Columbia, Canada. <https://doi.org/10.1145/2683467.2683472>
- Capretz, L. F., & Ahmed, F. (2018). A call to promote soft skills in software engineering. *Psychology and Cognitive Sciences—Open Journal*, 4(1), e1–e3. <https://doi.org/10.17140/PCSOJ-4-e011>
- Čavojská, V., & Jurkovič, M. (2017). Comparison of experienced vs. novice teachers in cognitive reflection and rationality. *Studia Psychologica*, 59(2), 100–112. <https://doi.org/10.21909/sp.2017.02.733>
- Chattopadhyay, S., Nelson, N., Au, A., Morales, N., Sanchez, C., Pandita, R., & Sarma, A. (2020). *A tale from the trenches: Cognitive biases and software development* [Conference session]. ACM/IEEE 42nd international conference on software engineering, Seoul, South Korea. <https://doi.org/10.1145/3377811.3380330>
- Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1), 37–46. <https://doi.org/10.1177/001316446002000104>
- Damjanović, K., Novković, V., Pavlović, I., Ilić, S., & Pantelić, S. (2019). A cue for rational reasoning: Introducing a reference point in cognitive reflection tasks. *Europe's Journal of Psychology*, 15(1), 25–40. <https://doi.org/10.5964/ejop.v15i1.1701>
- Danilova, A., Naiakshina, A., Rasgauski, A., & Smith, M. (2021). *Code reviewing as methodology for online security studies with developers—A case study with freelancers on password storage* [Conference session]. Seventeenth USENIX Conference on Usable Privacy and Security, Berkeley, CA, United States.
- Dash, N., & Gladwin, H. (2007). Evacuation decision making and behavioral responses: Individual and household. *Natural Hazards Review*, 8(3), 69–77. [https://doi.org/10.1061/\(ASCE\)1527-6988\(2007\)8:3\(69\)](https://doi.org/10.1061/(ASCE)1527-6988(2007)8:3(69))
- De Neys, W., Rossi, S., & Houdé, O. (2013). Bats, balls, and substitution sensitivity: Cognitive misers are no happy fools. *Psychonomic Bulletin & Review*, 20(2), 269–273. <https://doi.org/10.3758/s13423-013-0384-5>
- Evans, J. S. B. T. (1984). Heuristic and analytic processes in reasoning. *British Journal of Psychology*, 75(4), 451–468. <https://doi.org/10.1111/j.2044-8295.1984.tb01915.x>
- Evans, J. S. B. T. (2003). In two minds: Dual-process accounts of reasoning. *Trends in Cognitive Sciences*, 7(10), 454–459. <https://doi.org/10.1016/j.tics.2003.08.012>
- Evans, J. S. B. T. (2010a). *Thinking twice: Two minds in one brain*. Oxford University Press.
- Evans, J. S. B. T. (2010b). Intuition and reasoning: A dual-process perspective. *Psychological Inquiry*, 21(4), 313–326. <https://doi.org/10.1080/1047840X.2010.521057>
- Evans, J. S. B. T., & Stanovich, K. E. (2013). Dual-process theories of higher cognition: Advancing the debate. *Perspectives on Psychological Science*, 8(3), 223–241. <https://doi.org/10.1177/1745691612460685>
- Fagerholm, F., Felderer, M., Fucci, D., Unterkalmsteiner, M., Marculescu, B., Martini, M., Tengberg, L. G. W., Feldt, R., Lehtelä, B., Nagyvárad, B., & Khattak, J. (2022). Cognition in software engineering: A taxonomy and survey of a half-century of research. *ACM Computing Surveys*, 54(Suppl. 1), 1–36. <https://doi.org/10.1145/3508359>
- Frederick, S. (2005). Cognitive reflection and decision making. *Journal of Economic Perspectives*, 19(4), 25–42. <https://doi.org/10.1257/089533005775196732>
- Gigerenzer, G. (2002). *Adaptive thinking: Rationality in the real world*. Oxford University Press.
- Gigerenzer, G. (2008). Why heuristics work. *Perspectives on Psychological Science*, 3(1), 20–29. <https://doi.org/10.1111/j.1745-6916.2008.00058.x>
- Gigerenzer, G. (2015). *Simply rational: Decision making in the real world*. Oxford University Press.
- Gigerenzer, G., & Gaissmaier, W. (2011). Heuristic decision making. *Annual Review of Psychology*, 62(1), 451–482. <https://doi.org/10.1146/annurev-psych-120709-145346>
- Gigerenzer, G., & Todd, P. M. (1999). *Simple heuristics that make us smart*. Oxford University Press.
- Gotterbarn, D. (2001). Informatics and professional responsibility. *Science and Engineering Ethics*, 7(2), 221–230. <https://doi.org/10.1007/s11948-001-0043-5>
- Hagoort, P. (2023). The language marker hypothesis. *Cognition*, 230, Article 105252. <https://doi.org/10.1016/j.cognition.2022.105252>
- Hallett, J., Patnaik, N., Shreeve, B., & Rashid, A. (2021). “Do this! Do that!, And nothing will happen” Do specifications lead to securely stored passwords? [Conference session]. 43rd international conference on software engineering (ICSE '21), Madrid, ES, Spain. <https://doi.org/10.1109/ICSE43902.2021.00053>
- Hamari, J. (2017). Do badges increase user activity? A field experiment on the effects of gamification. *Computers in Human Behavior*, 71, 469–478. <https://doi.org/10.1016/j.chb.2015.03.036>
- Hjeij, M., & Vilks, A. (2023). A brief history of heuristics: How did research on heuristics evolve? *Humanities and Social Sciences Communications*, 10(1), 1–15. <https://doi.org/10.1057/s41599-023-01542-z>
- Hoffmann, A. O. I., Post, T., & Pennings, J. M. E. (2015). How investor perceptions drive actual trading and risk-taking behavior. *Journal of Behavioral Finance*, 16(1), 94–103. <https://doi.org/10.1080/15427560.2015.1000332>
- Hoover, J. D., & Healy, A. F. (2019). The bat-and-ball problem: Stronger evidence in support of a conscious error process. *Decision*, 6, 369–380. <https://doi.org/10.1037/dec0000107>
- Hoover, J. D., & Healy, A. F. (2021). The bat-and-ball problem: A word-problem debiasing approach. *Thinking & Reasoning*, 27(4), 567–598. <https://doi.org/10.1080/13546783.2021.1878473>



- Ivory, M. (2022). *The soft skills of software learning development: The psychological dimensions of computing and security behaviours* [Conference session]. International conference on evaluation and assessment in software engineering 2022, Gothenburg, Sweden. <https://doi.org/10.1145/3530019.3535344>
- Ivory, M., Sturdee, M., Towse, J. N., Levine, M., & Nuseibeh, B. (2023). *Can you hear the ROAR of software security? How Responsibility, Optimism And Risk shape developers' security perceptions*. PsyArXiv. <https://doi.org/10.31234/osf.io/pexvz>
- Ivory, M., Towse, J. N., Sturdee, M., Levine, M., & Nuseibeh, B. (2022). *Cognitive reflection, optimism and uncertainty in software development*. <https://doi.org/10.17605/OSF.IO/SJ8BT>
- Jackendoff, R. S. (2009). *Language, consciousness, culture: Essays on mental structure*. MIT Press.
- Janssen, E. M., Meulendijks, W., Mainhard, T., Verkoeijen, P. P. J. L., Heijltjes, A. E. G., van Peppen, L. M., & van Gog, T. (2019). Identifying characteristics associated with higher education teachers' Cognitive Reflection Test performance and their attitudes towards teaching critical thinking. *Teaching and Teacher Education*, 84, 139–149. <https://doi.org/10.1016/j.tate.2019.05.008>
- Jones, H. S., Towse, J., Race, N., & Harrison, T. (2019). Email fraud: The search for psychological predictors of susceptibility. *PLOS ONE*, 14(1), Article e0209684. <https://doi.org/10.1371/journal.pone.0209684>
- Kahneman, D., & Frederick, S. (2002). Representativeness revisited: Attribute substitution in intuitive judgment. In T. Gilovich, D. Griffin, & D. Kahneman (Eds.), *Heuristics and biases* (1st ed., pp. 49–81). Cambridge University Press. <https://doi.org/10.1017/CBO9780511808098.004>
- Kahneman, D., & Klein, G. (2009). Conditions for intuitive expertise: A failure to disagree. *American Psychologist*, 64, 515–526. <https://doi.org/10.1037/a0016755>
- Kahneman, D., Slovic, P., & Tversky, A. (Eds.). (1974). *Judgment under uncertainty: Heuristics and biases* (1st ed.). Cambridge University Press.
- Kahneman, D., & Tversky, A. (1979). Prospect theory: An analysis of decision under risk. *Econometrica*, 47(2), 263–291. <https://doi.org/10.2307/1914185>
- Keil, M., Wallace, L., Turk, D., Dixon-Randall, G., & Nulden, U. (2000). An investigation of risk perception and risk propensity on the decision to continue a software development project. *The Journal of Systems and Software*, 53, 145–157. [https://doi.org/10.1016/S0164-1212\(00\)00010-8](https://doi.org/10.1016/S0164-1212(00)00010-8)
- Kina, K., Tsunoda, M., Hata, H., Tamada, H., & Igaki, H. (2016). *Analyzing the decision criteria of software developers based on prospect theory* [Conference session]. 2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER), Osaka, Japan. <https://doi.org/10.1109/SANER.2016.115>
- Kirlappos, I., Beauteament, A., & Sasse, M. A. (2013). “Comply or die” is dead: Long live security-aware principal agents. In A. A. Adams, M. Brenner, & M. Smith (Eds.), *Financial cryptography and data security* (pp. 70–82). Springer. [https://doi.org/10.1007/978-3-642-41320-9\\_5](https://doi.org/10.1007/978-3-642-41320-9_5)
- Kula, R. G., German, D. M., Ouni, A., Ishio, T., & Inoue, K. (2018). Do developers update their library dependencies?: An empirical study on the impact of security advisories on library migration. *Empirical Software Engineering*, 23(1), 384–417. <https://doi.org/10.1007/s10664-017-9521-5>
- Kunda, Z. (1990). The case for motivated reasoning. *Psychological Bulletin*, 108, 480–498. <https://doi.org/10.1037/0033-2909.108.3.480>
- Levy, J. S. (1992). An introduction to prospect theory. *Political Psychology*, 13(2), 171–186.
- Liberali, J. M., Reyna, V. F., Furlan, S., Stein, L. M., & Pardo, S. T. (2012). Individual differences in numeracy and cognitive reflection, with implications for biases and fallacies in probability judgment. *Journal of Behavioral Decision Making*, 25(4), 361–381. <https://doi.org/10.1002/bdm.752>
- Lopez, T., Sharp, H., Tun, T., Bandara, A., Levine, M., & Nuseibeh, B. (2019). *Talking about security with professional developers* [Conference session]. 2019 IEEE/ACM joint 7th international workshop on conducting empirical studies in industry (CESI) and 6th international workshop on software engineering research and industrial practice (SER IP), Montreal, QC, Canada. <https://doi.org/10.1109/CESSE-IP.2019.00014>
- Lopez, T., Sharp, H., Tun, T., Bandara, A. K., Levine, M., & Nuseibeh, B. (2022). Security responses in software development. *ACM Transactions on Software Engineering and Methodology*, 32(3), 1–29. <https://doi.org/10.1145/3563211>
- Lopez, T., Tun, T., Bandara, A., Mark, L., Nuseibeh, B., & Sharp, H. (2019). *An anatomy of security conversations in stack overflow* [Conference session]. 2019 IEEE/ACM 41st international conference on software engineering: Software engineering in society (ICSE-SEIS), Montreal, QC, Canada. <https://doi.org/10.1109/ICSE-SEIS.2019.00012>
- Loske, A., Widjaja, T., & Buxmann, P. (2013, December 18). Cloud computing providers' unrealistic optimism regarding IT security risks: A threat to users? *ICIS 2013 proceedings*. Association for Information Systems. <https://aisel.ai/snet.org/icis2013/proceedings/BreakthroughIdeas/11>
- Manzoor, F., Wei, L., & Asif, M. (2021). Intrinsic rewards and employee's performance with the mediating mechanism of employee's motivation. *Frontiers in Psychology*, 12, Article 563070. <https://doi.org/10.3389/fpsyg.2021.563070>
- McAuliffe, M., & Triandafyllidou, A. (2021). *World migration report 2022*. International Organisation For Migration. <https://publications.iom.int/books/world-migration-report-2022>
- Mohanani, R., Salman, I., Turhan, B., Rodríguez, P., & Ralph, P. (2020). Cognitive Biases in Software Engineering: A Systematic Mapping Study. *IEEE Transactions on Software Engineering*, 46(12), 1318–1339. <https://doi.org/10.1109/TSE.2018.2877759>
- Mølokken, K., & Jørgensen, M. (2005). Expert estimation of web-development projects: Are software professionals in technical roles more optimistic than those in non-technical roles? *Empirical Software Engineering*, 10(1), 7–30. <https://doi.org/10.1023/B:EMSE.0000048321.46871.2e>
- Moritz, B., Siemsen, E., & Kremer, M. (2014). Judgmental Forecasting: Cognitive Reflection and Decision Speed. *Production and Operations Management*, 23(7), 1146–1160. <https://doi.org/10.1111/poms.12105>
- Nadi, S., Krüger, S., Mezini, M., & Bodden, E. (2016). *Jumping through hoops: Why do Java developers struggle with cryptography APIs?* [Conference session]. 38th international conference on software engineering, Austin, Texas. <https://doi.org/10.1145/2884781.2884790>
- Naiakshina, A., Danilova, A., Gerlitz, E., von Zezschwitz, E., & Smith, M. (2019). “If you want, I can store the encrypted password”: A password-storage field study with freelance developers [Conference session]. 2019 CHI conference on human factors in computing systems, Glasgow, Scotland. <https://doi.org/10.1145/3290605.3300370>
- Oliveira, D. S., Lin, T., Rahman, M. S., Akefirad, R., Ellis, D., Perez, E., Bobhate, R., DeLong, L. A., Cappos, J., & Brun, Y. (2018). *API blindspots: Why experienced developers write vulnerable code* [Conference session]. Fourteenth USENIX Conference on Usable Privacy and Security (SOUPS '18), Baltimore, MD, United States. <https://doi.org/10.5555/3291228.3291253>
- Oliveira, D. S., Rosenthal, M., Morin, N., Yeh, K.-C., Cappos, J., & Zhuang, Y. (2014). *It's the psychology stupid* [Conference session]. 30th annual computer security applications conference, New Orleans, Louisiana, United States. <https://doi.org/10.1145/2664243.2664254>
- Palassis, A., Speelman, C. P., & Pooley, J. A. (2021). An exploration of the psychological impact of hacking victimization. *SAGE Open*, 11(4). <https://doi.org/10.1177/21582440211061556>
- Palombo, H., Tabari, A. Z., Lende, D., Ligatti, J., & Ou, X. (2020). *An ethnographic understanding of software (in)security and a co-creation model to improve secure software development* [Conference session]. Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020), Berkeley, CA, United States. <https://doi.org/10.5555/3488905.3488917>

- Pennycook, G., McPhetres, J., Zhang, Y., Lu, J. G., & Rand, D. G. (2020). Fighting COVID-19 misinformation on social media: Experimental evidence for a scalable accuracy-nudge intervention. *Psychological Science*, 31(7), 770–780. <https://doi.org/10.1177/0956797620939054>
- Pennycook, G., & Ross, R. M. (2016). Commentary: Cognitive reflection vs. calculation in decision making. *Frontiers in Psychology*, 7, Article 9. <https://doi.org/10.3389/fpsyg.2016.00009>
- Petre, M. (2022). Exploring cognitive bias “in the wild”: Technical perspective. *Communications of the ACM*, 65(4), Article 114. <https://doi.org/10.1145/3517215>
- Ralph, P. (2011). Toward a theory of debiasing software development. In S. Wrycza (Ed.), *Research in systems analysis and design: Models and methods* (pp. 92–105). Springer. [https://doi.org/10.1007/978-3-642-25676-9\\_8](https://doi.org/10.1007/978-3-642-25676-9_8)
- Rauf, I., Petre, M., Tun, T., Lopez, T., Lunn, P., Van Der Linden, D., Towse, J., Sharp, H., Levine, M., Rashid, A., & Nuseibeh, B. (2021). The case for adaptive security interventions. *ACM Transactions on Software Engineering and Methodology*, 31(1), 1–52. <https://doi.org/10.1145/3471930>
- Rhee, H.-S., Ryu, Y. U., & Kim, C.-T. (2012). Unrealistic optimism on information security management. *Computers & Security*, 31(2), 221–232. <https://doi.org/10.1016/j.cose.2011.12.001>
- Roesler, D. (2020). *A computer science academic vocabulary list* [Master’s thesis].
- Samuels, R., Stich, S., & Bishop, M. (2012). Ending the rationality wars; How to make disputes about human rationality disappear. In S. Stich (Ed.), *Collected papers: Vol. 2. Knowledge, rationality, and morality, 1978–2010* (pp. 236–268). Oxford University Press.
- Schielzeth, H., Dingemanse, N. J., Nakagawa, S., Westneat, D. F., Allee, H., Teplitsky, C., Réale, D., Dochtermann, N. A., Garamszegi, L. Z., & Araya-Ajoy, Y. G. (2020). Robustness of linear mixed-effects models to violations of distributional assumptions. *Methods in Ecology and Evolution*, 11(9), 1141–1152. <https://doi.org/10.1111/2041-210X.13434>
- Sharot, T. (2011). The optimism bias. *Current Biology*, 21(23), R941–R945. <https://doi.org/10.1016/j.cub.2011.10.030>
- Shreeve, B., Gralha, C., Rashid, A., Araujo, J., & Goulão, M. (2022). Making sense of the unknown: How managers make cyber security decisions. *ACM Transactions on Software Engineering and Methodology*, 32(4), 1–33. <https://doi.org/10.1145/3548682>
- Siegrist, M., Gutscher, H., & Earle, T. C. (2005). Perception of risk: The influence of general trust, and general confidence. *Journal of Risk Research*, 8(2), 145–156. <https://doi.org/10.1080/1366987032000105315>
- Sinayev, A., & Peters, E. (2015). Cognitive reflection vs. calculation in decision making. *Frontiers in Psychology*, 6, Article 532. <https://doi.org/10.3389/fpsyg.2015.00532>
- Spadini, D., Çalikli, G., & Bacchelli, A. (2020). *Primers or reminders? The effects of existing review comments on code review* [Conference session]. 2020 IEEE/ACM 42nd international conference on software engineering (ICSE), Seoul, South Korea.
- Stagnaro, M., Pennycook, G., & Rand, D. G. (2018). Performance on the Cognitive Reflection Test is stable across time. *Judgment and Decision Making*, 13, 260–267. <https://doi.org/10.1017/S1930297500007695>
- Tahaei, M., & Vaniea, K. (2019). *A survey on developer-centred security* [Conference session]. 2019 IEEE European symposium on security and privacy workshops (EuroS&PW), Stockholm, Sweden. <https://doi.org/10.1109/EuroSPW.2019.00021>
- Tetlock, P. E., & Kim, J. I. (1987). Accountability and judgment processes in a personality prediction task. *Journal of Personality and Social Psychology*, 52, 700–709. <https://doi.org/10.1037/0022-3514.52.4.700>
- Thomson, K. S., & Oppenheimer, D. M. (2016). Investigating an alternate form of the cognitive reflection test. *Judgment and Decision Making*, 11(1), 99–113. <https://doi.org/10.1017/S1930297500007622>
- van der Linden, D., Anthonysamy, P., Nuseibeh, B., Tun, T., Petre, M., Levine, M., Towse, J., & Rashid, A. (2020). *Schrödinger’s security: Opening the box on app developers’ security rationale* [Conference session]. ACM/IEEE 42nd International Conference on Software Engineering, Seoul, South Korea. <https://doi.org/10.1145/3377811.3380394>
- Veracode. (2020). *State of software security* (Vol. 11). <https://www.veracode.com/sites/default/files/pdf/resources/soossreports/state-of-software-security-volume-11-veracode-report.pdf>
- von Eye, A., & von Eye, M. (2008). On the marginal dependency of Cohen’s K. *European Psychologist*, 13(4), 305–315. <https://doi.org/10.1027/1016-9040.13.4.305>
- Warrens, M. J. (2014). On marginal dependencies of the  $2 \times 2$  kappa. *Advances in Statistics, 2014*, Article e759527. <https://doi.org/10.1155/2014/759527>
- Weinstein, N. D. (1980). Unrealistic optimism about future life events. *Journal of Personality and Social Psychology*, 39(5), 806–820. <https://doi.org/10.1037/0022-3514.39.5.806>
- Welsh, M. B., Burns, N. R., & Delfabbro, P. H. (2013). *The Cognitive Reflection Test: How much more than numerical ability?* [Conference session]. Cognitive science conference (Vol. 7).
- West, R. F. (2008). The psychology of security. *The Psychology of Security*, 51(4), 34–40. <https://doi.org/10.1145/1330311.1330320>
- Wiederhold, B. K. (2014). The role of psychology in enhancing cybersecurity. *Cyberpsychology, Behavior, and Social Networking*, 17(3), 131–132. <https://doi.org/10.1089/cyber.2014.1502>

(Appendix follows)

## Appendix

### Examples of Sentences and Their Transformed PURL Score

Subject ID	PURL	Sentence	Keywords
01	.00	I was assigned a project to handle a drinks factory day-to-day purchases, sales, and employee records three months ago.	N/A
89	.24	The main considerations I take when aiming to identify potential risks or security vulnerabilities when developing software are to consider the ways in which information could be compromised.	Risks, security, compromised
126	.35	Where personal experience is not available, I would regularly visit a risk assessment template throughout the development process and ensure that potential risks are regularly identified during the development lifecycle.	Risk, risks
09	.58	Good decisions always make software development smooth and secure, team lead would be a big factor, his ideas and decisions matters most	Decision, secure, decisions
143	.75	I would say that user data collected from already existing secured software will provide additional protection from risks, such as Gmail, bank accounts, and other similar places where there is already an existed security.	Secured, protection, risks, security

*Note.* PURL = proportion of uncertainty-related language; ID = identification number; N/A = not applicable.

Received April 7, 2023  
Revision received September 4, 2023  
Accepted September 11, 2023 ■

## 6.1 Statement of Continuous Thesis Summary

*“Whatever stance one adopts on the contentious normative issues of whether a preference can be ‘wrong’ and whether more reflective people make ‘better’ choices, respondents who score differently on the CRT make different choices, and this demands some explanation.”*  
- Frederick, 2005

In this chapter, I report on a study exploring the security perceptions of freelance software engineers in relation to dual processing theory. Developers are asked how they recognise and mitigate security risks during development and complete measures of cognitive reflection, risk aversion, and optimism bias. I identify an interaction between cognitive reflection and optimism through mixed-effect regression modelling in the language used around uncertainty. Optimistic outlooks combined with higher cognitive reflection increase uncertainty, but realistic or pessimistic views accompanied by increased reflection reduce uncertainty. Software engineers with average or pessimistic views on the security of their code are more likely to speak intuitively about risk.

This study contributes to the thesis by providing the first evidence about the psychological dimensions of security behaviours. The interaction between psychological variables emphasises the complexity of secure coding behaviours yet also supports the framework that dual processing theory is informative with respect to secure behaviours, confirming ideas from Petre (2022) and Robins (2022). This chapter offers one answer to the second research question, “What influence do the identified psychological dimensions have on security behaviours within software engineering?”. Developers tend to overemphasise their ability to code securely compared to a general developer population, indicating they are typically biased towards security vulnerabilities. Optimism combined with cognitive reflection found an interaction for risk awareness, suggesting that awareness is not easily reduced into measures of cognition.

The chapter also strengthens the overarching thesis contribution by empirically testing how cognitive traits underpin security behaviours in a software context. It introduces

measurable psychological variables of cognitive reflection and optimism that link back to the soft skills identified earlier in the thesis (e.g. decision-making). By doing so, this chapter helps demonstrate that soft skills are not only socially valued (as seen in the earlier chapters) but also behaviourally consequential in shaping how developers perceive and respond to risk. This bridges the soft skill frameworks introduced in Chapters 4 and 5 with psychological theory, contributing to a more integrative understanding of developer behaviour.

I explore the same rich text data in the upcoming chapter through a thematic analysis. Initially, it was envisaged that Chapters 6 and 7 would be a combined, single empirical chapter. However, it became apparent that the depth and richness of the analysis required separate treatment. Combining *all* the different ideas would overburden the reader and undermine the strength of the conclusions one hopes to emphasise. As well as clarifying the different ideas, separating the analyses also offered an opportunity to disseminate research strategically for different audiences to emphasise the interdisciplinary nature of the work. The present chapter is published in an APA psychology journal, whereas the upcoming chapter is under review in a software engineering journal. Addressing two different audiences draws them to an important interdisciplinary research space.

### ***6.1.1 Contribution to Thesis Argument and Forward Trajectory***

This chapter advances the thesis argument by being the first to empirically explore dual processing theory as a psychological dimension of secure software development. Through the interaction of optimism and cognitive reflection, the study demonstrates that cognitive processing styles can meaningfully shape how developers conceptualise and talk about risk. This supports the thesis's broader argument that secure software engineering is influenced not only by technical competence but also by underlying cognitive traits, thereby reinforcing the need to understand and support software development in context-specific, psychological terms.



By connecting security-related behaviour to specific psychological mechanisms, this chapter contributes directly to the thesis's central claim: that psychological theory offers a valid and underexplored lens through which to explain secure software practices. It extends the thesis beyond normative discussions of skills into explanatory models that account for how developers think, decide, and act under uncertainty. Chapter 6 marks the beginning of an exploration of the psychological underpinning of soft skills, where soft skills are no longer treated generically, but rather are embedded within the behavioural realities of software engineering under uncertainty.

The next chapter builds directly on this work by analysing the same interview dataset through the lens of social identity theory, focusing specifically on how developers discuss responsibility, optimism, and risk. This shift provides a complementary, socially-oriented interpretation of secure behaviour and allows the thesis to examine both individual cognition and social identity as distinct but interacting forces in secure software development.

## **7 Can You Hear The ROAR of Software Security? How Responsibility, Optimism and Risk shape developers' security perceptions**

Ivory, M., Sturdee, M., Towse, J., Levine, M., & Nuseibeh, B. (*in review*). Can you hear the ROAR of software security? How Responsibility, Optimism And Risk shape developers' security perceptions. *Empirical Software Engineering*.  
<https://doi.org/10.31234/osf.io/pexvz>

## Can you hear the ROaR of software security? How Responsibility, Optimism and Risk shape developers' security perceptions

MATTHEW IVORY, Lancaster University, Great Britain

MIRIAM STURDEE, University of St Andrews, Great Britain

JOHN TOWSE, Lancaster University, Great Britain

MARK LEVINE, Lancaster University, Great Britain

BASHAR NUSEIBEH, LERO, Republic of Ireland and Open University, United Kingdom

How do the psychological perceptions of software developers shape secure software development? We engage freelance software engineers and computer science students about their security posture and reveal how self-defined social identities alongside heuristics and biases affect approaches to development. We also identify behaviours indicative of increased risk of project delays or failure. A thematic analysis extracted three core themes of Responsibility, Optimism and Risk (ROaR). We show how language about responsibility for security is framed through psychological concepts of diffusion, displacement, and acceptance of responsibility. We also examine the way developers orientate to risk awareness, appetites for risk, and risk mitigation strategies. Examples of unrealistic optimism biases are highlighted and discussed. The data underline the importance of acknowledging the human values within the socio-technical system of software development. We emphasise the relevance and applicability of core psychological research and theories for software engineering and offer examples of how these contribute to a deeper understanding of the software development cycle.

CCS Concepts: • **Security and privacy** → **Social aspects of security and privacy**;

Additional Key Words and Phrases: thematic analysis, responsibility, risk, optimism, security

### ACM Reference Format:

Matthew Ivory, Miriam Sturdee, John Towse, Mark Levine, and Bashar Nuseibeh. 2024. Can you hear the ROaR of software security? How Responsibility, Optimism and Risk shape developers' security perceptions. 1, 1 (February 2024), 38 pages. <https://doi.org/10.475/123.4>

---

Authors' addresses: Matthew Ivory, Lancaster University, Department of Psychology, Lancaster, Lancashire, LA1 4YW, Great Britain, [matthew.ivory@lancaster.ac.uk](mailto:matthew.ivory@lancaster.ac.uk); Miriam Sturdee, University of St Andrews, School of Computer Science, St Andrews, Fife, KY16 9SX, Great Britain, [ms535@st-andrews.ac.uk](mailto:ms535@st-andrews.ac.uk); John Towse, Lancaster University, Department of Psychology, Lancaster, Lancashire, LA1 4YW, Great Britain, [j.towse@lancaster.ac.uk](mailto:j.towse@lancaster.ac.uk); Mark Levine, Lancaster University, Department of Psychology, Lancaster, Lancashire, LA1 4YW, Great Britain, [mark.levine@lancaster.ac.uk](mailto:mark.levine@lancaster.ac.uk); Bashar Nuseibeh, LERO, Ireland, Lancaster, Lancashire, LA1 4YW, Republic of Ireland, Open University, United Kingdom, [bashar@lancaster.ac.uk](mailto:bashar@lancaster.ac.uk).

---

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

© 2023 Association for Computing Machinery.

Manuscript submitted to ACM

## 1 INTRODUCTION

Developing secure software is a complex process, affected by a combination of social, cognitive, and technical factors [70]. One method of achieving secure software is through secure coding, a programming practice that avoids exploitable software vulnerabilities [69]. Despite efforts from software engineering communities to provide developers with tools [53], documentation [2], and practices [89] for improving secure behaviours, software engineers still produce insecure software [76]. Alarming, two-thirds of software contains at least one OWASP top ten vulnerability [86], underlining failures even with respect to common and well-known vulnerabilities. In this paper, we provide a complementary approach to understanding and facilitating secure coding - through a focus on developers' individual perceptions and beliefs. We argue that developers' own framing of the concepts of Responsibility, Optimism And Risk (ROaR) provides an important contribution to the challenge of security.

Cybersecurity protocols exist for the purpose of ensuring secure computer communication, but often the breakdown in security stems from people's behaviours, rather than the software itself [61]. Despite the abundance of efforts to improve security, human behaviour is still often considered the weak link in the chain [72]. Much of previous work focuses on improving user behaviour in cybersecurity, but we turn the spotlight towards the developers. If we acknowledge that developers are also *users* [73], we can more easily accept the heterogeneity in their approach to security, which we can use to better design interventions and tools for promoting secure behaviours.

A frequent finding in software engineering research is that functionality is prioritised over security [40, 57, 77]. Software security is often invisible to end-users, often assuming security defaults are enough [25]. In a similar vein, security during development is often treated as a non-functional requirement, in that it is an implicitly assumed attribute of high-quality software [78]. It is therefore increasingly accepted that security requires more than reliance on technical competence and must incorporate human factors of social behaviours and decision making. Software developers are a core component in the software development cycle, whose actions can have direct consequences.

Only once we adequately appreciate how developers view security can we begin implementing effective interventions [68], whether psychological or technical. Security can be seen as a barrier to achieving goals, and even if not, it is typically considered to be a low priority. By considering these behaviours in a psychological frame, we draw out ideas that have been implicitly touched upon in previous research, but not explicitly framed.

Our research deployed a thematic analysis on the perception and understanding of risks in the software development cycle, as articulated by software engineer freelancers and computer science (CS) students with an emphasis on security through a survey study. We collected data from both freelancers and students as they represent a diversity of software engineering experience, from the early stages through to those developing software as a career. These form important groups insofar as their views on software security are unencumbered by company processes, groups, and structures [41]. The complementary, quantitative analysis of the data indicated no real differences between the two groups [33] that suggests security is perceived similarly across a broad range of experiences. Solo or freelance developers are a large percentage of the developer community [85], and if we only explore security studies with professional, established developers, their perceptions may be conflated or influenced by their employer's motivations or policies, not reflecting

the true individual differences in security. Freelancers are often employed in situations where they may be the sole software engineer within a project, such as making changes or updates to small business websites, and in such scenarios, the employers may be unaware of or unable to articulate security requirements properly, leaving the developer responsible for ensuring that their software remains secure.

**RQ1: How do social identities and group memberships affect the way that risk is approached by software developers, particularly when considering software security?** To address our first question, we examined the data through psychological theories of individual and group interactions. By mapping these theories to the security perceptions of software developers in their interaction with software development, we sought to provide explanations for these views based in psychology. It is important to recognise that software is not developed in isolation, and developers not only communicate with one another, but possess rich unique social cultures that shape and form the way software is produced [43, 74]. We utilised the Social Identity Approach (SIA) to frame ways that software developers form social groups, think about themselves in relation to others, and how they interact with these groups.

SIA posits that to understand social behaviours and individual's perceptions, we cannot focus on the individual as an *individual*, instead it is necessary to focus on how behaviours and perceptions are tied to *social identities*, that are their self-definitions of social group memberships [30]. SIA is built upon Social Identity Theory which describes the complexities of group-based behaviours, and how individuals ascribe membership to groups and how these impact real-world behaviours [79]. Social identities are the self-categorisation of group memberships and can be defined by an individual's understanding that they belong to certain social groups through shared personal and emotional values [1].

Social groups are perceived differently by an individual depending on whether they see themselves as a member of a group (the ingroup) or whether they do not self-identify as a member of the group (the outgroup). People typically show more positive, altruistic behaviours to ingroup members [39, 84], and more negative, discriminatory behaviours to outgroup members [14]. To the extent that psychological evidence shows that developers construct and dynamically change their affiliative relationships with others, it is important to explore how they express their roles set against "others", as this provides opportunity to better understand the social identities ascribed to themselves but also to others involved in the software development process.

Social identities are not congruent with typecasts, but are fluid, self-defined group memberships to allow individuals to associate with others who share emotionally significant norms and values [1]. The saliency of an identity fluctuates depending on the social context an individual experiences, giving way to different memberships. For example, one may self-perceive as a "computer scientist" in academic conferences, but affiliate more as a "software engineer" when conducting research. Social identity reflects perceived group affiliations within social circumstances, allowing individuals to associate or distance themselves from others, rather than reflecting immutable stereotypes.

Individuals who see themselves as having a *software developer* identity will ascribe to common norms and values considered to be associated with being a software developer. The identity of *software developer* does not necessarily overlap with others in their organisation, with Backevik et al. [7] finding a clear "us versus them" view between developers and stakeholders. They found that software developers did not identify with stakeholders in the company and rated their ingroup members (other developers) more positively than other

teams (with whom they shared no identities). This positive rating can reinforce the social group [71], further promoting trust in others, creating a social feedback loop that reinforces security behaviours.

**RQ2: *How do cognition, cognitive heuristics and biases affect the way that risk is approached by software developers, particularly when considering software security?*** To address the second question, we applied cognitive psychology theories to developers’ security beliefs to understand how cognition shapes security perceptions. We understand from psychological research that prior knowledge, perceptions, and environmental feedback affect our decision-making abilities [38], and software development is full of decision-making opportunities, such as solution implementation, optimisation or choosing appropriate frameworks. We examine the data from the perspective of dual-processing theory [22, 34].

Heuristics are cognitive “shortcuts” people rely upon during decision making. Heuristics make use of frameworks and cognitively simple processes to make intuitive, instinctive judgments [35]. The use of heuristics as a decision-making style has been termed *system 1* processing [34]. As heuristics rely upon simple cognitive processes, when they fail to be effective, they become biases, which are the systematic errors in decision-making.

To prevent biases, people need to use more conscious, deliberate decision-making processes, called *system 2* processing. These processing styles cannot occur simultaneously, and the default intuitive system 1 processing is suppressed when system 2 processing is activated. System 2 processing is computationally more demanding of our cognitive capacity which is why it is not the automatic processing style, but it allows for abstract and hypothetical reasoning [22], which is critical for complex tasks such as software code production.

Evidence shows heuristics and biases shape software development and vulnerability [57] and yet there are individual differences both in propensity to use heuristics and the influence of domain contexts in which they may manifest themselves. That is, they are not inherently and ubiquitously observed. Accordingly, we consider evidence for two distinct types of bias, specifically optimism and risk appetite, amongst developers.

In answering these two research questions, we contribute a psychological analysis of responsibility, risk, and optimism in secure coding behaviours through the lens of ROaR. We also provide a demonstration of how theories of social and cognitive psychology can be integrated within software engineering, deepening our understanding of behaviours and perceptions exhibited.

Our analysis extracted three themes forming ROaR: *Responsibility*, *Optimism*, and *Risk & Uncertainty*. Responsibility is operationalised with respect to the mechanisms of refusing and accepting responsibility, with three subthemes of diffusion, displacement, and acceptance of responsibility. The theme of Optimism provides evidence for different manifestations of the unrealistic optimism bias. The third theme of Risk & Uncertainty shows how risk behaviours are exhibited, including subthemes of risk appetite, risk mitigation, direction of risk, and bad code versus bad actors. Our themes resonate with research findings and arguments, but we are unaware of previous efforts to integrate these with psychology frameworks as attempted here.

This paper is structured as follows. In Section 2 we present the methodology. Section 3 presents the findings from our thematic analysis. Section 4 contextualises our research alongside related work. Section 5 discusses the findings, and the paper concludes in Section 6.

## 2 METHODOLOGY

Ivory et al. [33] reported a quantitative analysis of software developers and their cognitive reflection and generic optimism bias. They showed an interaction between these variables and the uncertainty expressed in

Manuscript submitted to ACM

their language. However, such a quantitative analysis of individual differences doesn't naturally address the attributions made about software development risk. In the present analysis, we use a qualitative approach by examining the rich text responses, offering a complementary analysis that provides a more nuanced understanding of how security and risk are perceived by software developers.

Our study was preregistered on the Open Science Framework (OSF), lodging details on the study design, data collection strategies and analytic plans. The preregistration can be accessed at <https://doi.org/10.17605/OSF.IO/ZBQE4>. Moreover, to promote open science, data, analysis scripts, and supplementary materials can be found at <https://doi.org/10.17605/OSF.IO/4MQKD>.

## 2.1 Participants

Data from 145 participants were used (73 professional freelance developers and 72 CS students). Freelancers were recruited from Upwork, a freelance developers' website<sup>1</sup>. An advert invited participants to submit applications to a study regarding security understanding. Freelance participants were required to be working in software development where security in software was included and to have fluent English writing skills. They were asked to confirm their experience with writing secure code (e.g., sanitising inputs) and to provide details of a project they had been involved in to confirm eligibility. Freelancers were compensated at £10/hour, reflecting their professional experience and time. Each request was reviewed, five applicants were declined for not meeting the requirements, and one for providing identical responses to a previous applicant.

Students were collected using internal university mailing lists and the participant recruitment website Prolific. Criteria for participation required fluent English, current studentship, and to be studying computer science. The research offer was not advertised to those who do not meet these criteria. Six were recruited from mailing lists and 66 from Prolific. Student participants were compensated at £8.50/hour.

When referring to both groups collectively in this paper, the term "software developer" is used. Questions invited participants to consider what can go wrong during software development, how these may affect themselves, what risks are worth taking during the software development process, and the approaches they take to identify risks.

All data collection occurred in early 2021 before large language models such as Chat-GPT were publicly available, reducing any likelihood that the data analysed were generated by non-human participants. Due to research restrictions at the time due to social and physical distancing being required, in-person interviews were not appropriate, resulting in the survey design used.

The ages reported by participants are shown in Table 1. Reflecting the global nature of freelance work, the nationality and location of participants was diverse. Due to data collection issues, nationality data is unavailable for the developer sample, which is replaced with country-level location data. Figure 1 shows an approximation of location/nationality of participants by presenting the continent-level data. Most people do not leave the country they were born in, with one in 30 migrating internationally [46], and so representing a mix of both nationality and location data together approximates both data types on a continent scale.

<sup>1</sup>Similar participant pools have been previously used in research. For examples, see Hallett et al. [28] and Rauf et al. [68]

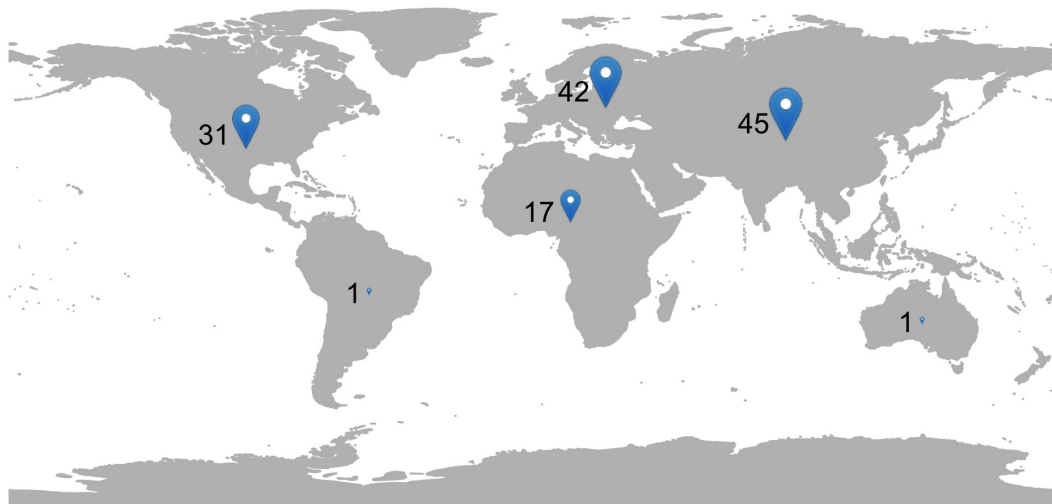


Fig. 1. Map displaying the combined location and nationality data of participants on a continent level. This represents an approximation of both location and nationality of each participant due to typically low immigration movement. Most participants were from North America, Europe, or Asia.

Table 1. Reported age of participants

Age	Developer	Student
18 - 24	30	42
25 - 34	36	20
35 - 44	6	8
45 - 54	1	2

## 2.2 Data Collection

Qualtrics, an online surveying platform, was used to collect data. All participants were told the study would take around 30 minutes to complete. All methods and materials were approved for use by the University ethics committee prior to data collection.

**2.2.1 Materials.** Following sign-up, participants were presented with an information sheet detailing the study and what was expected. Participants gave informed consent, then completed the survey. The survey consisted of basic demographic information (age, gender), alongside software development specific demographics, such as developer experience, team size, and programming languages known. They responded to a set of cognitive measures not included in the present analysis but used in Ivory et al. [33]. This was followed by the four questions listed below. Finally, participants were debriefed and provided researcher contact details. The survey can be seen at <https://doi.org/10.17605/OSF.IO/P6DY5>.

The questions did not deceive participants as to the study's intent, because if participants were not primed for talking about security, useable data would likely be sparse. This research decision was made based on the understanding that security-related data is often sensitive and requires a specific context to be shared. Manuscript submitted to ACM



upon the research findings of others, where an absence of priming results in little to no security behaviours witnessed [28, 52]. Whilst this choice may have resulted in a self-selection of security-conscious participants, this would result in more useable data than if they did not mention security at all.

1. Describe a time when you successfully developed and released/launched a software project, either in a professional or personal capacity. This could either be a recent example, or perhaps a project you were particularly proud/happy with. Please include information concerning the purpose of the project and how important security was during development.
2. When considering the process of developing and launching software/web applications, what is at risk of potentially going wrong and how could these risks affect you? You should consider the size or the significance of the potential factors that may go wrong and how this may affect you (e.g., risk of functional failure, financial losses, damage to reputation etc.)
3. If you were to consider software development as a series of 'gambles' (decisions that confer possible risk), what gambles would be considered worthwhile or worth a risk during the process of developing software? Why? These gambles may be considered from both an individual perspective and as a team. Both decisions that you take individually, or decisions that are enforced by policy, should be considered.
4. What approaches or considerations, do you, or your team, take when aiming to identify potential risks or security vulnerabilities when developing software? What is the reasoning behind these decisions? You should consider the decisions and thought processes behind selecting certain tools (such as static analysis tools), as well as identifying specific tools.

### 2.3 Analysis

A thematic analysis identified core themes surrounding how software development was perceived by developers. We followed the analysis steps proposed by Braun and Clarke [13] of familiarisation, code generation, theme generation, theme review, and theme naming; the first author was responsible for familiarisation, code generation and initial theme generation, all authors were involved in discussions around refining and reviewing themes until agreement was achieved. Data familiarisation was carried out by reading through all participant responses grouped by individual, then grouped by question to develop familiarity with the relationships within and between participants. During this stage, initial codes (sections of text containing potential themes or topics) were noted to focus further readings. The data were revisited and coded with labels highlighting meaningful content. This process was iterated until further readings provided no additions to existing codes or the creation of new codes, and all content had been inspected for content relevant to each code. Initial themes were generated, and codes were merged, renamed, or separated where necessary to align with the themes. Themes were reviewed, defined, and named, and subthemes were grouped together under umbrella themes. Finally, the themes and subthemes were reviewed once more for coherency, independence, and distinctiveness to ensure multiple themes did not cover the same content and ideas. Following this, theories from psychology that explained the themes appropriately were identified.

Data were coded using Taguette, an open-source qualitative analysis tool [67]. The codebook is available in Appendix A with primary and auxiliary themes and codes.

### 3 RESULTS

Thematic analysis was applied to the data, which led to the identification of three key concepts, specifically *Responsibility*, *Optimism*, and *Risk & Uncertainty*. These themes were chosen as they captured and represented the data effectively, and in doing so addressed the research questions. The order of the acronym ROaR does not indicate a priority but serves as a memorable label for these themes.

Responsibility highlights how the ownership of consequences is viewed in software development and how individuals either refuse or accept this ownership. Three subthemes are included: *diffusion of responsibility*, *displacement of responsibility*, and *acceptance of responsibility*. Optimism captures instances of predispositions towards positive views, or overly optimistic perceptions of software in both security and functionality contexts. As such it highlights examples of the unrealistic optimism bias and how these persist within software development. Risk & Uncertainty looks at how the existence of unknown consequences is managed and how related behaviours are expressed. Subthemes include *risk appetite*, *risk mitigation*, *direction of risk*, and *bad code versus bad actors*.

To aid reader comprehension spelling errors are corrected or context is provided using square parentheses. Freelancer quotes are denoted with an F, and students with an S.

#### 3.1 Responsibility

This theme highlights three psychological representations of responsibility: diffusion, displacement, and acceptance. Diffusion of responsibility refers to the psychological phenomenon where increases in group size reduces the personal responsibility felt by an individual by increasing feelings of anonymity and reducing accountability. Displacement of responsibility is where an individual minimises their responsibility by transferring it to another who they consider having greater seniority or authority to themselves. Acceptance of responsibility demonstrates instances where individuals assume responsibility for actions and consequences, typically through emotional, moral, or social motivations.

We define responsibility as the assumed ownership of one's actions and associated consequences. Causal responsibility is the direct relationship between an actor and their actions but does not account for emotional or moral motivations. Moral responsibility is grounded in philosophical research and speaks to a willingness to operate in a morally acceptable manner through reducing potential harm and maximising good [23]. Subsequently, responsibility refers to the moral or ethical responsibilities of developers as opposed to their causal responsibility. Responsibility can be either accepted or refused, and two manners of refusing responsibility are described below.

Personal responsibility can be diffused or displaced resulting in reduced accountability, which has been linked to increases in making risky decisions for others [58]. Similarly, lower levels of responsibility often result in people reporting less control over a task or situation [6], decreasing the likelihood someone will act, and potentially rely more on others to take control and responsibility [45]. Decreased responsibility minimises a person's perceived role in any harmful consequences of poor or insecure code. If an individual refuses to acknowledge their contribution to a negative outcome, any harm can be attributed to others instead, exonerating themselves [50].

**3.1.1 Diffusion of Responsibility.** Diffusion of responsibility is defined as a socio-psychological event where an individual experiences reduced responsibility for an action (or absence of action) when they believe others

to be present or involved [32]. In a group context, diffusion of responsibility reduces the teams' ability to achieve goals when individuals experience decreased responsibility towards their contributions.

Diffusion can be a result of an organisation's workflow, with assumptions that others in the team will spot issues and correct these. These assumptions are likely combined with a diminished sense of responsibility over ensuring their code is as good as possible. It is acknowledged that in many organisations it is standard procedure to have code reviewed by others, but code review is reliant on the reviewers being able to comprehend the code and identify weak areas. With diffused responsibility, security and testing teams experience a higher workload than if everyone took responsibility for their own contribution. Participant S88 highlights their personal expectations and refers to those responsible as *other* teams, with their language indicating distance between themselves and the others assumed to also consider security. From this, S88 could see reviews as not part of their role or identity, and so pushes this task away from themselves, without clear identification of who should take responsibility.

*"I would expect the program to be mulled over by many other Junior/Senior Designer and Admin teams" - S88*

Trust can also be invoked through social identity. S112 refers to trusting *teammates* instead of using a less personal term (such as *others*). A shared identity, such as working on the same team, leads to increased levels of trust in group members [8]. This increased trust could have an unintended consequence of promoting reliance and subsequent diffusion of responsibility. A reliance on others only works when all group members share in the same values as others, but if a member feels less connected to the group, they can deviate from group expectations on security [70], which could result in diffusion compromising the software.

*"Trusting your teammates to have followed the procedure for writing the code correctly and to trust them to verify the functionality of their code" - S112*

Tools and third-party software can also be seen as a catalyst for diffusion of responsibility. These objects represent the work and efforts of other developers, and consequently an extension to an individual's social team involved in their software development. This reliance on other people's code can be linked to the belief that published tools are secure by design [60], which is not always the case. By virtue of tools being available, developers may infer this to mean that they must have been carefully tested and reviewed.

*"When using official and up to date tools and frameworks majority of known security risks are already mitigated" - F08*

F08 utilised technology as a diffusion mechanism. Often diffusion occurs between individuals rather than an individual and object, but software produced by *official* well-known organisations could be viewed as output from ingroup members. This deferment may be built upon a belief the individual shares social ties with developers working within other organisations, emphasising the trust given to ingroup members [8]. Similar styles of social identities are seen in the open-source software domain, with shared social identity driving engagement in a project with others, despite never having met them [31].

If diffusion of responsibility is a danger to secure code production, then how might it be reduced? One suggestion is to deanonymize or hold individuals clearly accountable [58, 59]. Developers may find themselves accountable to either the ingroup or the outgroup, which can affect the behaviours exhibited in these intergroup and intragroup relations. In our findings, sometimes respondents described the concern of being visible (and thus potentially accountable) to the outgroup. On other occasions, their accountability was to the ingroup.

*“I took serious measures on security before I even started the development and had to look for quicker ways to implement the system, but with advanced security to avoid client looking for me after product delivery.” – F01*

*“There were some APIs that are not hidden away in a VPC because I didn’t have the time to configure them. Being the only guy on the team to handle all this I was always concerned that I might have missed something and that it would come back to bite me.” – S77*

For F01 and S77, we can see the way they orientate to intergroup or intragroup concerns. F01 presents a clear view of the clients as an outgroup, which shapes their responsibility for their work. The client outgroup can identify the individual as being responsible, increasing an individual’s accountability, which the individual may seek to reduce. Conversely, S77 identifies a concern for their relationship to the in-group. They feel they are the only one responsible for the software and might end up letting the team down. Both quotes present a reduction of diffusion through increased accountability and reduced anonymity. However, for one, the inter-group relations are key, while for the other the intra-group relations are the motivator. F01 appears keen to avoid further interaction with the outgroup and so highlights their use of “advanced security” to avoid this. This could be interpreted as including higher levels of security than may be necessary, to avoid future interactions with clients. S77 acknowledges that they are the only one capable of completing a certain task, and with this comes a heightened sense of accountability. This outgroup identity (in the sense that others on the team are not handling the software development) forces S77 to be acutely aware of the software security. Both quotes represent different ways in how the social groups can shape the way that responsibility is felt by the individual.

**3.1.2 Displacement of Responsibility.** Displacement of responsibility is a form of refusal to accept responsibility for personal actions, when they are operating within a delegate role carrying out decisions made by authority figures [10]. The stronger the perceived legitimacy of an authority figure, the weaker the self-responsibility for an individual’s actions. Displacement stems from social cognitive theory and moral disengagement, where individuals choose not to exercise self-regulatory mechanisms when faced with enacting irresponsible decisions [9, 90].

Responsibility can be displaced upwards through hierarchies, especially to senior executives, managers, and team leads. Diffusion occurs throughout a group, whereas displacement occurs where responsibility can be “passed up the chain” onto management or team leaders, institutional policies, or even stakeholders and clients. By displacing responsibility towards the organisation [44], developers are provided an opportunity to see their task as simply the implementation of someone else’s intentions (typically management), removing personal connections, allowing developers to ignore implications of the project outcomes [7].

The way individuals refer to their actions provides insights into their identity perception. F08’s language emphasises their exclusion from the group, referring to “my goal”, “my concerns” and displacing responsibility “away from me”. They explicitly refer to an active aim to distance themselves from responsibility through the reporting mechanisms commonly seen in hierarchical organisations, placing social distance between themselves and others involved. Many freelancers may have little influence over the project beyond their contractual obligation, and so reporting issues is the extent of their ability, but the language used by F08 signals a clear distance and absence of any personal responsibility for any issues.

*“My goal is to assess the potential risks, fix them, or at least report the risk and my concerns. This way I remove responsibility away from me.” – F08*

Management or leadership roles provide obvious opportunities for displacing responsibility, particularly when considering the implementation of software features and security. Displacement of responsibility to those in management roles is evidenced by F25 who perceives little connection between their work and any potential risk because a senior figure is seen as responsible for identifying issues in their code. In the example of F09, they directly reference the team leader as being responsible for decision making during software development. The references to an outgroup with authority, such as team leads or senior developers, indicates a clear displacement pathway.

*“Damage to reputation financial losses were never in my mind while developing any feature maybe as that is not directly linked to me and someone senior to me is always there to review those aspects.” – F25*

*“Good decisions always make software development smooth and secure. Team lead would be a big factor, his ideas and decisions [matter] most.” – F09*

Institutional documentation, such as policies or workflows, can also be used as a method of displacement. These are often formal documents designed to encourage consistency and standardisation. These can be perceived as an extension of management, as often they are approved by those in positions to determine team-wide or company-wide policies. As such, they present a clear displacement opportunity, representing a proxy for management. Like the diffusion seen using objects and technology, policies can be considered delegates of authority figures.

*“While developing software all the standard set of procedures related to security are followed as per mentioned by our company policies” – F45*

*“In general it is better to stick to best practises defined by internal company policy or at least settled in the developer community” – F61*

Other avenues for displacement responsibility are through clients or stakeholders. When developers are hired for their skills to implement the technical aspects of software and nothing more, developers can feel absolved of any need to query decisions made by others and focus purely on implementation [26]. In the example provided by F69, they mention that their employer does not require security, but more importantly, they do not mention including security in their work regardless. This implies that they may be displacing responsibility to their employer by creating functional, yet insecure software. This displacement is further evidenced by a clear divide of responsibility in the language used, as they repeatedly refer to themselves as “I” and their employee as “they”. They even identify the responsibility structure in the words, “I was their mobile developer”. F69 presents a strong ingroup and outgroup within their response, with their employee as an outgroup and themselves as the mobile developer.

*“I have worked with a company in USA and they were developing a social media platform for events projects etc and I was their mobile developer... They did not force me to obey any security rules and guidelines and the only thing they wanted was fully working application” – F69*

Displacement of responsibility requires a salient authority figure to be identified as responsible. Where diffusion of responsibility allows for responsibility to be unassumed, displacement occurs when developers transfer the responsibility for their personal actions towards a senior figure, such as management, policies, or clients. Displacement of responsibility can occur for both functional and security related aspects of software.

**3.1.3 Acceptance of Responsibility.** Acceptance of responsibility can result from various factors, with the shared outcome that individuals assume more responsibility for their own actions and consequences. Acceptance of responsibility is often linked with increased moral, emotional or social valence.

When it comes to accepting responsibility, one of the strongest motivators is possessing a moral or emotional connection to their actions, which can be self-imposed or imposed through social connections with others, such as users. Self-imposed connections stems from a desire to reduce personal consequences and avoiding negative emotions. F15 presents a social identity of *app developer* with a value of “protecting users from harm”. Their use of personal language indicates that they see this value as being part of the role of app developer.

*“As app developers we are sometimes responsible for some intimate parts of human life of the users. So by not protecting the app properly we may easily ruin someone’s life and that’s a huge burden to carry.” - F15*

Others accept responsibility through social motives, such as user engagement or empathy for users. F69 mentions that users trust developers to keep their information secure and this can be seen as the motivator, contributing to accepting responsibility. They use language such as “our” and “us”, indicating their identity claim as a software developer. Associated with this identity are norms and values, one of which is providing quality, secure software. To identify as a software developer, an individual is required to meet these norms, which motivates an individual’s desire to protect users.

*“Our responsibility as a software developer is protecting the user data from stealing and prevent these vulnerabilities and fix them and patch them. . . the things that users trust us. And we should protect their data from [being stolen] because they are important for them.” - F69*

Others demonstrate a moral connection with their work, without mention of personal consequences, focusing more on the potential damage to others. This may be motivated by developers being empathetic to the user base and implicitly understanding the expectations of users, as indicated by F23.

*“The security of a website is [important] that’s why we [usually] care about user data first [because] no one want to have his data published or used for any reason without his [permission].” - F23*

In contrast to instances of diffusion when using third party tools, some individuals refused opportunities for diffusion by being cautious of using code or software that they were not involved in. Whilst not strictly an acceptance of responsibility, it is a refusal of diminishing personal responsibility, increasing personal responsibility. F17 demonstrates this caution by mentioning the need to be certified secure, rather than simply selecting tools for their functionality.

*“Development team need to use only those third-party tools which are security certified and have no potential security glitch. Both security analysis tools and third-party libraries must be security flaws proof. Reason for this process is to mitigate all the potential risks beforehand.” - F17*

Acceptance of responsibility was linked with security conscientiousness and an emotional connection to the consequences. Developers felt that users trusted or relied upon them to produce secure, effective software which in turn cultivates a feeling of responsibility. A common theme seen in the examples of acceptance of responsibility is the personal language used, such as “we” or “our responsibility”, evidencing their identity claim is strongly linked with responsible development.

### 3.2 Optimism

The theme of optimism highlights potential software development-related examples of unrealistic optimism bias [88]. The unrealistic optimism bias is where individuals see themselves to be less likely to experience negative events than others, and more likely to experience positive events. This has been shown across a wide range of scenarios including failure to properly calculate the time required to complete a task, known as the planning fallacy [16], and genuine optimistic belief that personal outcomes are more favourable than someone else in the same circumstance.

We use unrealistic optimism as just one example of how cognition can affect our perceptions, certainly around security. Acknowledging that software security is not supported by specific cognitive processes or heuristics [33], it is an activity that will more easily succumb to biases. Potential instances of unrealistic optimism are more easily identifiable in discussions over behaviours, but the presence of one bias suggests an increased likelihood of others influencing an individual's behaviour.

*"We need to write code in such way that no hacker can expose those" - F4*

If developers refuse to accept responsibility for their code, then testing behaviours are likely reduced and security issues missed. Unrealistic optimism in software development can be risky, as it allows developers to have an increasingly relaxed view about security. Not only can this prove to be problematic for stakeholders and clients who may be expecting secure, robust software, but also for the developer, who may find that opportunities for work slowly dry up if they become associated with poor software.

Others refuse the possibility they may be victims, such as F29, by ignoring malicious actors such as those who may target software recreationally [18]. They acknowledge their own, and others, lack of security knowledge and the accompanying belief that being a target is very unlikely. Others are optimistic that securing their software according to the threats they are already aware of (and comfortable dealing with) is sufficient. If this reduces their security awareness/training incentives, future software may be secured to an outdated standard.

*"most of us don't know much about security and they may ignore it apart some basic [development] rules to respect because we don't believe we could someday become a [victim] of hacking." - F29*

Developers expressing an optimism bias often spoke about security and risk in absolutes and dismissed uncertainty or insecure code as issues of other developers, but not for themselves. Or they expressed confidence in their processes that meant they restricted their views on security in the development cycle. This may be associated with a security blindness, in that optimism prevents developers from reflecting upon their development choices.

### 3.3 Risk & Uncertainty

The theme of Risk & Uncertainty explores risk-related behaviours within software development. This includes subthemes of risk appetite, risk mitigation, direction of risk, and bad code versus bad actors. Risk appetite is defined as the willingness to engage in risk-taking behaviour, acknowledging the risk and uncertainty of consequences. Larger risk appetites imply greater acceptance of risk. Risk mitigation covers behaviours that are directly aimed at reducing risk and uncertainty. Risk direction focuses on whom the risk affects, whether it is directed towards the developer, or externally towards users and stakeholders. Bad code versus bad

actors explores how security issues are framed in terms of insecure code being at fault or intentional bad actors being the problem.

**3.3.1 Appetite.** Appetite for risk exists for both security and functionality. Risk appetite can be characterised by conscious decisions resulting in increased risk. These decisions may be active or passive, in that active decisions are those which intentionally reduce efforts in specific areas, and passive decisions are those that increase risk through focusing efforts in areas that cause reduction of attention elsewhere. Choosing to focus on optimisation or choosing to reduce security testing may result in the same insecure code, but for different reasons. Understanding risk appetite provides context for which risks are considered acceptable, and we can use this to focus on reducing dangerous risk appetites where they are unnecessary.

F16 demonstrates a passive risk appetite, with security being reduced in favour of functionality. They acknowledge this potential risk, but believe they produce secure enough code naturally, and so the value of increased security-related behaviours does not seem appropriate.

*“We don’t take any special processes to identify risks or security vulnerabilities. I’m not sure what the exact reasoning behind these decisions are but I think we are more focused on developing and maintaining the software. Furthermore, I feel like our processes are already pretty secure.” - F16*

F47 also demonstrates a passive risk appetite, by delaying the inclusion of security measures. This risk appetite is hinged upon the expectation that developing security later does not disrupt software functionality. In excessively prioritising functionality, securing the code may never be fully achieved, resulting in skipping over aspects of security, or becoming more reliant on external sources to identify security issues [4].

*“I would also build something first quickly before [securing] it. I wouldn’t leave it to the last moment but there [definitely] has to be a [proof of concept] of the product before you start thinking of [securing] it.” - F47*

Active risk appetite is characterised by choosing to increase risk, which can be through or through using new ideas for improving software at the risk of increased errors such as that mentioned by S145. Active risk appetites were commonly associated with increasing functionality within given time constraints.

*“Something that could dramatically improve the speed of a process but could also introduce quite a few new bugs I would say that is a risk worth taking.” - S145*

Through time constraints, software developers need to decide what to prioritise. This decision-making process must consider the risks associated with choices that consume time available for the project. This can be linked to the planning fallacy, which is the underestimation of the required time to complete a task [16].

*“For less security critical software it may be justified to reduce the level of scrutiny the software is placed under if time or cost pressure is an issue” - S118*

Technical debt is the sacrifice of software maintainability in favour of producing a minimum viable product [83]. S107 presents an example of technical debt being chosen through the decision-making process, and choosing to accept a risk appetite that affects themselves or others who may need to refactor software later. This highlights a trade-off between having software produced fast, or having software produced well with expectations of longevity. Technical debt arises from a series of decisions which have been linked to biases such as anchoring, optimism and confirmation bias [12].

*“Whenever we are developing software we have to make quick and difficult decisions. We don’t have the time or budget to build a perfect system so we do have to weigh up which aspects are most important.”*



*Maintainability is something we often tradeoff, we gamble the future of our software in rushing to get it out as quickly as possible.” - S107*

**3.3.2 Mitigation.** Risk mitigation refers to actions taken to reduce or minimise uncertainty and risk. Risk mitigation was most referenced alongside planning and testing, and heavily linked to security. By demonstrating awareness for potential risks, and actively planning for these, developers take on responsibility.

The initial planning stages allows developers to exert conscious, deliberate judgements over decisions, reducing potential biases. Previous work highlights that by asking developers to create design specifications prior to writing secure code, this can prompt people into thinking more about security, acting as a priming effect [28]. Without a conscious awareness of potential security issues, this results in a restricted view on risk, which can result in blindspots [15, 56], where the developer is unable to see or understand where risks are present. By including risk management documents in planning stages, it can prompt developers to make more reflective judgements.

*“Before starting our project, we will list and filter out problems and vulnerability issues that are concerned with our system or software with their cost also then we will study how we can approach these problems” - F66*

Software testing provides opportunities to evaluate software, and to identify issues. It offers a chance to ensure software meets requirements prior to launch or pushes to production. Testing is a reactive process, dealing with code and software in its present form, which is different to the proactive planning stage. During the testing process, risk mitigation strategies may be at risk of being affected by confirmation bias, the tendency to seek evidence to confirm a prior belief, rather than to disprove [87]. This reactive view during testing could be a cause of confirmation bias where the testing involved only focuses on confirming beliefs of how the software should work, as opposed to attempting to break and actively find faults.

*“I perform tests for the applications that I make, I always try to handle all the security issues that I have come to be accustomed with before going forward with the application deployment.” - F27*

Risk mitigation can also be approached using third parties or products, such as security consultants, vulnerability detection tools or online information sources, such as the OWASP project. These can be seen as expert sources, which are trusted to provide information not easily available to the developer. Using these resources can demonstrate a developer's awareness of their own ignorance, and in doing so requires developers to trust in both the source's competence and benevolence [19]. Interactions with those who prioritise security, increases the likelihood of adopting security tools and behaviours [92].

*“We consider tools to identify security issues that are recommended in a large number of online articles and also having low cost.” - F25*

Risk mitigation strategies are diverse and cover the entire spectrum of software development, from initial planning to later-stage testing. These strategies require developers to be aware of risks and to be motivated to reduce these risks. If the risk is completely unknown, then it cannot be addressed, and if motivation is not present, developers are not going to be willing to expend extra effort in mitigating risks.

**3.3.3 Direction of Risk.** Risk direction refers to where consequences of a risk are directed, towards oneself or others. Understanding risk direction provides knowledge on why certain risks are considered acceptable. Research indicates that decisions made involving others can often be riskier than decisions involving the

self, and that the circumstances of the decision are critical to understanding when self-other decisions vary, specifically in the perceived social identity of others, and in framing the decision as either a gain or loss [64].

Risks affecting the self are a significant concern having potential real-world implications. Developers are likely to try and reduce self-risk, as these are experienced on an emotional level. Losses are characteristically avoided during decision making, and people exert reasonable effort into minimising losses [36]. Experiencing failures or having to abandon projects due to self-directed risks could challenge a developer's perception of their competence, and so developers would seek to reduce self-directed risk [21].

*"If any of my software get hacked or damage it will [affect] me financially as well as damage my reputation"*  
- F13

*"If data is stolen you risk being sued for damages by individuals and corporations potentially. Your reputation as a company would be decimated also and it could easily bankrupt the company."* – S107

Risk to others was typically presented without any personal emotional connection. Developers are aware of the potential risks but keep an emotional distance. This can be evidenced through the lack of personal language when talking about these risks. This contrasts to the self-directed risk examples (F13 and S107) who use language such as "my software", "my reputation", "you risk", "your reputation", indicating a stronger emotional and social tie to the consequences of risks. When talking about risk to others, the decreased personal language implies reduced social connectivity, and whilst they acknowledge the risk and its impact on others, beyond this they do not consider this as a risk to themselves personally. In doing so, this can reduce a developer's sense of responsibility for risk over others which can lead to potentially harmful code [7]. For F52 and F56, they express no personal language that links their actions to the consequences of insecure code.

*"having the access of admin panel could hamper the reputation cause the client was school children. And the students will be exposed to harmful contents"* - F52

*"there are several risks such as any development flaws by the developer or any bug where the hackers can enter into the system etc. These factors can affect the users directly"* - F56

**3.3.4 Bad Code versus Bad Actors.** This subtheme examines two main perspectives taken when considering security in software: bad code and bad actors. Bad code refers to security vulnerabilities that are caused by insecure or poor code, without specific reference to malicious intent. Bad actor refers to security issues that are the direct result of somebody trying to gain unauthorised access, or to deliberately damage software. Both present different views, with bad code being a direct result of developers' own actions and bad actors being a result of external forces.

*"Many beginner developers are not very focused on security and they just want to finish the project because of this misunderstanding there can be many issues like cross site scripting, DDoS attacks, and so many this type of vulnerabilities will lead them to big failures"* - F66

Bad code was associated with risk appetite, particularly in increasing the likelihood of bad code resulting from speeding up the development process, or simply prioritising deadlines rather than ensuring code was secure. Bad code can be associated with inexperience and misunderstanding the importance of security, mentioned by F66, or it can be associated with irresponsibility and a lack of care, as presented by F61.

*"Irresponsibility of workers can lead to the presence of misconfiguration vulnerabilities and bug"* - F61

When referencing bad actors in a security context, a common idea was viewing software from the perspective of a bad actor to see where vulnerabilities were. F48's comment on edge cases fits in line with the definition of security blindspots by Oliveira et al. [57], of vulnerabilities being edge cases that are not seen through our normal use of heuristics and decision making.

*"There are some edge cases which neither developer nor tester sometimes able to find it, so such things can severely affect the whole functionality of... software and someone may use that with bad intentions."* - F48

The other common connection with bad actors was self-risk, more so than risk to others despite it often being users' data that is at risk. The risk of having software that is successfully attacked by bad actors was associated with consequences that affected the developer themselves, more so than affecting the stakeholders.

*"Any time you have authentication and store an email address or personal information there is a risk that the information could be accessed by unauthorized parties. This can lead to liability and a loss of customers. Depending on the type of personal data that is accessed there could be lawsuits causing significant legal bills."* - S143

Mentions of bad actors and bad code were associated with different ideas. Bad code was linked with risk appetite and a willingness to develop less than ideal code in circumstances where they saw a gain in the development cycle, or simply through sacrificing quality under time pressures. Bad actors were presented as a constant uncertainty in the development process. These present two different views on what causes vulnerabilities in software, with bad code being the responsibility of the developer, whereas bad actors are seen as a persistent threat that one simply must accept as a risk in software development. In viewing bad actors in this way, developers can feel less responsible for vulnerabilities that are actively abused by others due to the ability to distribute blame wider than themselves [26].

## 4 RELATED WORK

Our work is positioned in the intersection of software engineering and psychology, building upon the foundations, and contributing to our understanding of the psychology of those involved in creating software, rather than consumers. Within software engineering, there is a large body of work exploring software security as well as the behaviours exhibited by the engineers. Some of the research explicitly draws upon psychological theories, such as cognitive load or social identity, whereas some implicitly use these theories to draw attention to behaviours and perceptions observed. In the present work, we use data collected on the perceptions of software security and use this to exemplify how cognitive and social theories can be used within software engineering.

In this section, we explore the research that both implicitly and explicitly use psychology, highlighting that it is more pervasive than perhaps initially thought. We present the works grouped by their social or cognitive nature, highlighting that even without mentioning specific theories, research into software development and security is abound with psychology.

### 4.1 Cognition in Software Engineering

Software developers have been found to write code intuitively and impulsively [66], indicating that the decision-making processes engaged during code creation are not necessarily reflective and conscious, but rather based upon heuristics or unconscious choices. This reflects ideas from dual-processing theory [22],

where the default system 1 uses heuristics and instinctive mechanisms for decision-making. Decision making does not only concern conscious, deliberate decisions but also the unconscious decisions that seemingly just happen.

System 1 processing is dominated with heuristics, but these can easily become biases when they are ill-suited for the context. Biases result in systematic errors that provide less than ideal results. Biases have been seen across the software development process from design [80] to testing [91], with common biases including confirmation bias, anchoring, and overconfidence or optimism bias [47, 65]. The presence of these biases across software development highlights that when developers are coding intuitively, they are prone to make these mistakes.

A programme of work has explored the idea of how heuristic-based thinking can affect a developer's ability to detect software vulnerabilities, [17]. They hypothesised that without engaging in security-aware, deliberate thinking, vulnerabilities will often be missed during code reviews. This was explored in both Java [56] and Python [15] with findings that neither programming experience nor expertise modulate the ability to detect software vulnerabilities.

Other works support the idea of cognition playing a role in secure coding practices and can be interpreted using dual processing theory. Research into prompting or priming participants for security increases their secure coding practices, both through explicitly requesting security [51], as well as implicitly by boosting the planning stages to encourage enhanced security awareness [28]. The priming process can act as a cue to engage in more deliberate, reflective thinking styles which can increase security awareness. The necessary cues for engaging system 2 processing for specific tasks can also be altered and made more sensitive, with experiences of security issues in software increasing the awareness over future potential vulnerabilities, leading developers to take increased care towards security [5].

Further supporting the idea of dual processing theory in software development is that security behaviours are associated with an increased need for cognition and rational thinking, as well as decreased avoidant, procrastinatory behaviour [20, 27]. These psychological measures are linked to more conscious, reflective styles of thinking aligned with system 2 processing. An interaction between measures of the two systems, cognitive reflection and optimism was seen for increased awareness over risk in secure development [33].

Cognitive load, the amount of information capable of being held in one's working memory, should be accounted for when handling complex software [94] and tools should consider reducing unnecessary information or data that does not align with general human cognition. Others have considered using cognitive load theory to address the creation of source code directly [82] through minimising the complexity of functions and decisions made during code creation.

These studies highlight the importance of cognition during the development process, with some referencing psychological theories and some not. In both situations, the presence of psychology is obvious, with many of the findings taken together pointing to the applicability of lesser-used theories, such as dual processing theory of decision making [22]. The works mentioned here contextualise the current research as we provide examples of how common themes seen in the development process can be interpreted using theories from psychology.

## 4.2 Social Psychology in Software Engineering

Developers who possess a strong social connection with members of their group have been seen to produce more secure code [70], with strong social and institutional structures creating heightened accountability, which results in stronger group membership. Shared values amongst group members also means that each individual trusts that their teammates will value security as strongly as they do [40].

In organisations where security is the priority, such as cryptographic software, it has been found that strong security cultures permeated the organisations with a shared consensus of the importance of security [29], speaking to the idea of social identities altering developers' values without explicating stating this in psychological terms. In non-security specific workplaces, it has been shown that an absence of security culture reduces the efficacy of security awareness [62].

Software developers often use online forums, such as StackOverflow for information gathering, and they also form communities with relationships being developed through the conversations had on these forums [42]. These communities are built around positive social interactions, which boosts the perception of a shared social identity within the community [49].

Backevik et al. [7] explored social identities within agile project teams, finding that successful teams were built upon a strong team identity, with individuals rating each other positively and seeing each other as developers. They found those outside of the development team were not considered to belong to shared social groups, representing a strong ingroup-outgroup dynamic within these projects. Assal and Chiasson [5] found in security-positive teams of engineers, they saw security as a shared responsibility, reflecting the shared value of security within their shared social identity.

When exploring the security decisions made by managers or those with responsibility in software projects, Shreeve et al. [75] found that even with minimal cybersecurity expertise, senior management figures can make appropriate decisions using logic and risk planning strategies. One of the strategies used was communication, and engaging in interactions can be beneficial in exploring multiple options. This has been seen elsewhere, where increased communication prompted engaging with different perspectives providing a more holistic awareness of security [48].

The social structures of a company have been seen to impact upon how developers make decisions, with individuals seeking advice from people perceived to be their peers rather than security experts, over fears of time-wasting [55]. This could become problematic when peers provide incorrect information. Risks may end up being downplayed or disregarded by those without the required knowledge.

## 5 DISCUSSION

In our analysis, three key emergent themes were identified in how software developers discussed their development of secure software. We present these as a trio of equal-weighted aspects of security perceptions, ROaR: Responsibility, Optimism, and Risk. We highlight three subthemes of responsibility: where developers diffuse responsibility between teams or technology; displace responsibility towards senior employees; and acceptance of responsibility. The theme of optimism highlighted several instances where individuals demonstrated potentially overly optimistic views about the development process, particularly in terms of security. Finally, risks and uncertainty in the development process were framed in terms of appetite and the risks that individuals deem acceptable, how they mitigated issues, and the direction of which risks faced.

Our research and the findings provide examples of how general theories such as the social identity approach can be mapped and deployed in software engineering domains with a focused and practical manner. The theories we use are not the only theories appropriate for interpreting the data but represent how they can be applied. Previous work has suggested the psychology literature is vast but affords little applicability to software engineering [94], however we provide evidence to show that within the language used by developers, instances of responsibility, optimism and risk can be identified, and interpreted using general theories from psychology.

## 5.1 Responsibility

Responsible software development is critical because the use of software can have tangible consequences for society [54]. While some developers may perceive software development solely as a problem-solving exercise, allowing them to reduce personal responsibility for the ethical impacts of software [26], the reality is that software can have a profound impact on the people who use it or are affected by its (mis)behaviour.

Through the analysis of the data, we identified three theories of responsibility that can be applied to the language used by software developers around security. We demonstrated that examples of diffusion, displacement, and acceptance can be seen within the language used. The examples can also be linked to the more general theory of social identity, with acceptance of responsibility typically being linked to an increase in shared identities between individuals.

Diffusion of responsibility was exemplified through expectations of others reviewing code, using personal language (“us” and “teammates”, not “them”) indicating increased trust, and diffusion through trusting in tools and technology created by others. Where diffusion was seen through expectations, self-perceived responsibility is typically reduced as groups or teams grow larger. Solo freelancers cannot assume others will handle responsibility for the simple fact that no one else is involved in the project. In these situations, we highlighted examples of where developers diffused using the third-party tools and technology.

Others trust in their peers to follow procedures and to test their work as well. This can break down through diffusion and social loafing if everyone believes that they do not need to fully test their code as it will be picked up by their peers, whom they trust to operate to high standards. Depending on group factors, such as size or dispersion, diffusion behaviours can be increased as teams get larger or less connected (geographically or through communication), and individual’s feeling like their actions contribute little to the team goal [3].

We suggest that it may be possible that developers use tools developed by other software engineers and make assumptions they have been tested for functionality and security. Whilst the sample consisted of freelancers, not all were solely independent workers, indicated by some referencing working in groups or in larger projects. Security has been described as *event-driven* [40], with a reliance on psychologically external cues to initiate security behaviours, such as static analysis tools or stakeholder requirements. As a result, developers were found to lack ownership over security which can be interpreted as a rejection of responsibility via diffusion or displacement. Lopez et al. [40] found developers often rely upon security defaults within the tools used which reflects the diffusion via technology presented in the current research.

Social identity and the notion of shared identities can also play a role in diffusion, where a mistaken assumption from an individual that another shares an identity that values security results in an expectation that security will be assessed in the group. When the shared value is not present, it is easier for developers

to focus on their own values (such as fast project completion) than to consider others. Xie et al. [93] also identified developers' perceptions of others being responsible for security, referring to this as a misplaced sense of trust.

In displacing responsibility, developers mentioned senior employees or managers, using institutional policies or documents, or towards clients and stakeholders. It is important to note that we are not suggesting that these are inappropriate avenues to entrust responsibility, but instead we offer examples of where displacement *may* occur, resulting in weakened software. When referring to management or senior employees, the personal language used indicated a difference in social identities, with a distinction between the personal and the other ("they" and "them"). The absence of a shared identity affords the developer an easier way to refuse a responsibility for security.

With institutional policy, there are circumstances where this is best practice or gold standard, but there are also instances where this is not the case. One respondent (F69) found their employer had very loose requirements over security, to which they made no mention of ensuring security themselves. As Poller et al. [63] found, developers prioritised aligning their behaviours with management's policies over enacting personal values of security, highlighting instances of displacing responsibility.

The acceptance of responsibility can be linked to the concept of shared values and how intergroup and intragroup relations are experienced. The trust placed in developers by users becomes an aspect of user expectations, which motivate responsibility for security [93]. In doing so, developers may perceive users as being more identifiable and possessing some form of shared group identity, perhaps through the common interest in the software, which enhances developers' motivation to meet user expectations [81]. These ideas reflect the present research findings of acceptance of responsibility, reflecting greater attention given towards tasks and an awareness of the consequences of their decisions, which can increase engagement and performance [71].

The acceptance of responsibility, and handling of risk can also be linked with social identity. The social connection with risks directed towards others was practically non-existent in participant responses, indicating minimal concern for consequences to others. When considering risks towards oneself, the language used was personal and more emotional, indicating that a risk such as software failure was a direct attack on the individual's identity, resulting in efforts to regain their self-perceived identity [21]. Where a value of the software developer identity is "producing quality software", failing to achieve this value would directly challenge their identity.

Treating developers as ethically responsible individuals, rather than simply rewarding good behaviour encourages developers to internalise the ethical and social implications of their actions, with less need to reject responsibility [11]. Managers should motivate developers to consider the social implications of their code and the dangers of assuming others are responsible. Freelancers can also apply these findings by creating questions for potential clients or stakeholders about the software's social implications to not only create a discussion space, but to provide self-reinforcing responsibility. Increasing their own accountability by de-anonymising themselves can also help increase responsibility.

In practice, employers can use these findings to encourage team leaders and managers to foster a stronger security culture. A strong top-down culture ensures that values relating to security considered important by management will be adopted by all members of the organisation. This small change in work environment

can improve personal responsibility through social identity theory. Further work may be required to identify effective ways of nurturing and developing these cultures in existing workplaces.

**Takeaway Message 1:** *Theories of responsibility can be applied to software development cultures, and employers and employees should consider how to motivate responsibility throughout an organisation when considering security.*

## 5.2 Optimism Bias

Optimism bias was identified throughout the responses from developers and represents one of the many potential biases affecting software development [47]. The presence of biases indicates the use of system 1 processing, the intuitive, heuristic-based style of cognition. The unrealistic optimism bias was linked with contrasting opinions to reality. Developers saw themselves as capable of identifying security issues and resolving these in the short term, whereas research showing developers are typically blind to security issues [56], suggesting our participants experienced optimistic perspectives. Another optimistic perspective was about likelihood of being a victim of insecure code or malicious actors. Unrealistically optimistic views have strong implications on software security, as this can feed into motivation reduction and responsibility to code securely. Optimism bias was also seen in risk appetites, where developers believed they provide secure code by default and so focussing on security is not worth considering, or in optimism that they are only at risk of the vulnerabilities they have dealt with in the past.

Examples included reduced security testing through optimism over not being a victim of hackers, or through a belief that they possess enough security knowledge and that they cover all possibilities. The presence of optimistic beliefs suggests that system 1 processing is being engaged during the software development process, meaning that there is the possibility for reduced awareness of the need for security. We highlight that dual processing theory of decision making is an appropriate psychological theory to be applied to the software development cycle. Despite it being a general theory typically tested and built using general populations, we identify examples of system 1 processing via optimism bias. By mapping this theory onto software engineering, we offer a fresh perspective into why security is often handled in less-than-ideal ways.

**Takeaway Message 2:** *Optimism bias can affect developer's perceptions and behaviours, in turn influencing the software they produce. Being aware this can be identified through language; the way developers talk about security can provide insight into whether increased awareness of security is needed.*

## 5.3 Risk & Uncertainty

The psychological literature draws links between system 2 processing, the more reflective, deliberate processing style, and awareness of risk [24]. By drawing out references to risk appetites, mitigation, direction of risk, and those responsible for vulnerabilities, we present evidence of the presence of system 2 processing in software development. This further supports our suggestion that dual processing theory is a relevant theory to apply to software development.

Passive risk appetites were linked with prioritising functionality and delaying security implementation. Active appetites involved increasing functional requirements, reducing security testing through time constraints, or entering technical debt. Decision makers should be aware of decisions that deprioritise security through passive or active decisions. Passive risk appetites can be dangerous as the focus is not about reducing



security but increasing functionality which is viewed positively. By acknowledging the existence of these risky decisions, security can potentially be seen as more salient during decision making. Risk appetites can be influenced by both system 1 or system 2 processing, and without further testing, it is not possible to draw strong inferences from our present findings. Our findings do indicate the presence of different methods of handling decisions around security, and the mention of active and passive risk appetites suggests that theories of decision making are appropriate to apply here. Drawing upon other research, we know that priming developers for secure coding has a positive influence [57], indicating the link between security and system 2 processing.

Security has been seen to be considered a lower priority than functionality [37], and this may be attributed to the more visible nature of functional requirements [78], which means that during the interaction with software, security becomes less salient, which ties into our findings on passive risk appetites. If developers behave passively during the software development process, they are at greater risk of producing insecure code.

Risk mitigation was commonly mentioned in line with planning and testing with a security perspective, but a variety of biases can affect successful mitigation practices, such as unrealistic optimism or confirmation bias. To successfully mitigate risks, developers need to possess security knowledge, a willingness to look for issues and understand how to mitigate the issues. By creating a top-down security emphasis, group behaviours will align with security awareness, increasing the likelihood of deliberate judgements being made, reducing biases, and promoting system 2 thinking.

Consequences of risk were associated with personal risk more than risk to others. Individuals are motivated to avoid losses [36], and so decisions viewed as potential losses motivate developers to attend to these risks. Risks to others were commonly associated with an absence of emotional connection. By keeping this social distance, developers can accept greater risks for others [64]. Whilst risk is predominantly associated with cognition and decision making, in terms of consequences we also draw upon social identity theory, where a lack of shared identity affords the developers to keep an emotional perspective away from their code.

The theme of bad code versus bad actors highlights different representations of potential risk in a security context. Bad code was associated with instances where developers reduced actions around security inclusion or testing when faced with deadlines, providing further evidence to the idea that functionality is considered more important than security [40] and is knowingly prioritised by developers. Others attributed poor code to a lack of care or awareness in their code, indicating an increased need for clear attribution of responsibility. In doing so, and as highlighted in the responsibility theme, increased responsibility may lead to increased care and awareness of an individual's actions and choices.

Mentions of bad actors impacting software mentioned edge cases, indicated the existence of security blindspots, which have been seen to exist in previous empirical work [56]. Other mentions of bad actors were associated with the consequences of having software compromised or data stolen. This could be linked to a notion that bad actors often utilise attack vectors that are not anticipated, or even non-technical, such as using phishing attempts to gain user credentials for easier access. Some developers mentioned that their software was only used internally, and so security was not necessary despite the potential for bad actors to use social engineering to gain access to information or data.

***Takeaway Message 3: Passive risk appetites and poor risk mitigation strategies are likely linked with intuitive system 1 processing and less reflective thinking. Dual processing theory holds value for interpreting software developer's decision making around risk.***

In this research, we propose that two general psychological theories, social identity theory and dual processing theory of decision making can be successfully mapped and deployed within software development research. It is important to make these key links between well-established theories in social and cognitive psychology and the domain of software development. It is well understood that software security is a complex topic, with vulnerabilities constantly being identified [86], but we demonstrate in the present study, that the theories of social identity and dual processing have relevant applications within software security research.

## 5.4 Threats to Validity

**5.4.1 Internal.** The study design meant that participants could provide very short answers of ten or less words (nine participants wrote at least one short response), so some responses were not as rich as expected. This was seen more in the developer sample and may be due to the recruitment method used. Upwork is primarily for software development jobs rather than research, and so participants may be motivated to finish as fast as possible to maximise pay rewards. Other methods for eliciting high quality qualitative data need to be considered for future, such as in-person interviews, however it is acknowledged that the present study design allowed for many participants to be included, reducing the effect of the individual short responses on the overall data quality.

Differences in how language is used can affect the interpretation of the responses. Applying a Western-centric, native English-speaking view on these responses may alter the original participants' beliefs. When reading and coding data, multiple readings were used to try and gain the best understanding of the underlying point, rather than focussing on face value of the text. Differences in perspective and language skill will always occur in thematic analyses, and this is acknowledged. Further research may look to focus on smaller communities of developers to build a localised representation of their views.

The study design meant that participants were aware that the study was interested in software security, as it was stated in both the title and description of the study. Consequently, participants were primed for security through information sheets and the questions asked, potentially resulting in them emphasising their security and downplaying dangerous risk appetites. Despite this, responses were not homogeneous, and did not all present a pro-security, zero-risk view. If the security aspect of the research attracted security-conscious participants, we would have anticipated watertight responses regarding their security in software. As this isn't seen, it is unlikely that the absence of obfuscation significantly affected the data quality.

The data analysis was carried out by the first author, and consequently the codes and themes are potentially impacted by their personal biases and perspectives. The coder's perspective is acknowledged to be that primarily of a psychological interest. The initial codes and themes that emerged are consequently psychologically related. During theme review, the interdisciplinary combination of the authors was leveraged through discussions over the relevance and impact of the themes.

**5.4.2 External.** A key limitation of the study is *generalisability*. The study was carried out via an online survey asking freelance developers and CS students about their security and risk during the software development process. Not all developers fall into these categories, and most of the software that is distributed

Manuscript submitted to ACM

on a large, global scale is written by employed developers in organisations. It is important to understand how the findings may relate to other populations of developers not addressed in this research. Freelancers are a community used in previous research as an easier population to collect data from than organisations. One benefit of using freelancers is that they represent a diverse community, with potentially less influence from their employer (conducting a study within a single organisation limits generalisability to the organisation's security culture).

A potential threat to data validity are the sources. Upwork is primarily for recruiting freelancers for software projects not academic research, and vice versa for Prolific. Data quality might be affected by source – for example we lacked the means to verify claims of expertise and experience of freelancers, while Prolific has built-in quality control processes and guardrails. For Upwork, there are no such controls, and users are primarily incentivised by receiving positive feedback on their work to continue receiving job offers. Five applicants were removed due to not meeting criteria, and one suspected of being a duplicate account. Nonetheless, Upwork is a recruitment tool that has been used previously in software engineering research and represents a method of recruiting freelance engineers from a diverse pool.

Our findings are reliant on the idea that the identified themes and language used by participants reflects their actions within software development, but without further data these links cannot be made in the present study. Key objectives from this project were, first, to consider whether developers' views about software projects produced coherent themes and, second, to evaluate how these might speak to psychological constructs from different domains. The claims or conclusions from our work therefore lay the foundation upon which other research can test, confirm, complement, or challenge the generality of the conclusions here. We are reassured that the findings from the present study have good support in the psychology, cybersecurity, and software engineering research communities as noted in the discussion.

## 5.5 Future Research

Our future research will investigate how risk aversion, responsibility and biases identified in language used around software development are reflected in the code written by the same developers. Understanding whether developers who demonstrate greater risk aversion produce more secure software than those with larger risk appetites is important for identifying areas of interest to focus on.

A study combining a code comprehension or code writing paradigm, with a scenario-based measure of perceived responsibility and risk appetite, along with a qualitative examination of developers' responsibility and risk aversion (as used in the present research) could achieve this result. A quantitative examination of secure code comprehension or production, through developers' self-reported perceptions could allow us to understand how developers' actions are reflected in the language used in discussing secure software.

## 6 CONCLUSION

Developers' understanding of responsibility, security and risk was examined, with links to psychological theories for explanations. We identified ROaR, three themes were identified of responsibility, optimism, and risk & uncertainty. Evidence for the influences of how responsibility was viewed, how risk was framed and handled, and core cognitive biases was presented. Implications spoke to motivating developers to take responsibility for their contribution to software, as well as identifying areas of risk that are likely to occur and reducing over-optimistic views on security. Through these themes, we present examples of how two core

theories of psychology, social identity theory and dual processing theory, can be applied to software security. We highlight how these general theories can be mapped to the complex environment of software development with success, advocating the continued application of these theories in software engineering research.

## REFERENCES

- [1] Dominic Abrams and Michael Hogg. 1990. An Introduction to the Social Identity Approach. In *Social Identity Theory: Constructive and Critical Advances*. Harvester Wheatsheaf, London, United Kingdom, 1–9.
- [2] Yasemin Acar, Christian Stransky, Dominik Wermke, Charles Weir, Michelle L. Mazurek, and Sascha Fahl. 2017. Developers Need Support, Too: A Survey of Security Advice for Software Developers. In *2017 IEEE Cybersecurity Development (SecDev)*. IEEE, Cambridge, MA, USA, 22–26. <https://doi.org/10.1109/SecDev.2017.17>
- [3] Omar A. Alnuaimi, Lionel P. Robert, and Likoebe M. Maruping. 2010. Team Size, Dispersion, and Social Loafing in Technology-Supported Teams: A Perspective on the Theory of Moral Disengagement. *Journal of Management Information Systems* 27, 1 (July 2010), 203–230. <https://doi.org/10.2753/MIS0742-1222270109>
- [4] Vaibhav Anu, Kazi Zakia Sultana, and Bharath K. Samanthula. 2020. A Human Error Based Approach to Understanding Programmer-Induced Software Vulnerabilities. In *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. 49–54. <https://doi.org/10.1109/ISSREW51248.2020.00036>
- [5] Hala Assal and Sonia Chiasson. 2018. Security in the Software Development Lifecycle. In *Proceedings of the Fourteenth USENIX Conference on Usable Privacy and Security*. USENIX Association, Baltimore, MD, USA, 281–296. <https://doi.org/10.5555/3291228.3291251>
- [6] Ann Elisabeth Auhagen and Hans-Werner Bierhoff. 2002. *Responsibility: The Many Faces of a Social Phenomenon*. Taylor & Francis, Oxfordshire, United Kingdom.
- [7] Andreas Backevik, Erik Tholén, and Lucas Gren. 2019. Social Identity in Software Development. In *12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, Montreal, QB, Canada, 107–114. <https://doi.org/10.1109/chase.2019.00033>
- [8] Daniel Balliet, Junhui Wu, and Carsten K. W. De Dreu. 2014. Ingroup Favoritism in Cooperation: A Meta-Analysis. *Psychological Bulletin* 140 (2014), 1556–1581. <https://doi.org/10.1037/a0037737>
- [9] Albert Bandura. 1986. *Social Foundations of Thought and Action: A Social Cognitive Theory*. Prentice-Hall, Inc, Englewood Cliffs, NJ, US. xiii, 617 pages.
- [10] Albert Bandura. 2002. Selective Moral Disengagement in the Exercise of Moral Agency. *Journal of Moral Education* 31, 2 (June 2002), 101–119. <https://doi.org/10.1080/0305724022014322>
- [11] Melissa S. Baucus and Caryn L. Beck-Dudley. 2005. Designing Ethical Organizations: Avoiding the Long-Term Negative Effects of Rewards and Punishments. *Journal of Business Ethics* 56, 4 (Feb. 2005), 355–370. <https://doi.org/10.1007/s10551-004-1033-8>
- [12] Klara Borowa, Andrzej Zalewski, and Szymon Kijas. 2021. The Influence of Cognitive Biases on Architectural Technical Debt. In *2021 IEEE 18th International Conference on Software Architecture (ICSA)*. IEEE, Stuttgart, Germany, 115–125. <https://doi.org/10.1109/ICSA51549.2021.00019>
- [13] Virginia Braun and Victoria Clarke. 2006. Using Thematic Analysis in Psychology. *Qualitative research in psychology* 3, 2 (2006), 77–101. <https://doi.org/10.1191/1478088706qp063oa>
- [14] Marilynn B. Brewer. 2017. Intergroup Discrimination: Ingroup Love or Outgroup Hate? In *The Cambridge Handbook of the Psychology of Prejudice*. Cambridge University Press, New York, NY, US, 90–110. <https://doi.org/10.1017/9781316161579.005>
- [15] Yuriy Brun, Tian Lin, Jessie Elise Somerville, Elisha M. Myers, and Natalie C. Ebner. 2023. Blindspots in Python and Java APIs Result in Vulnerable Code. *ACM Transactions on Software Engineering and Methodology* (April 2023). <https://doi.org/10.1145/3571850>
- [16] Roger Buehler, Dale Griffin, and Michael Ross. 1994. Exploring the "Planning Fallacy": Why People Underestimate Their Task Completion Times. *Journal of personality and social psychology* 67, 3 (1994), 366. <https://doi.org/10.1037/0022-3514.67.3.366>
- [17] Justin Cappos, Yanyan Zhuang, Daniela Seabra Oliveira, Marissa Rosenthal, and Kuo-Chuan Yeh. 2014. Vulnerabilities as Blind Spots in Developer's Heuristic-Based Decision-Making Processes. In *Proceedings of the 2014 Workshop on New Security Paradigms Workshop - NSPW '14*. ACM Press, Victoria, British Columbia, Canada, 53–62. <https://doi.org/10.1145/2683467.2683472>
- [18] Samuel Chng, Han Yu Lu, Ayush Kumar, and David Yau. 2022. Hacker Types, Motivations and Strategies: A Comprehensive Framework. *Computers in Human Behavior Reports* 5 (March 2022), 100–167. <https://doi.org/10.1016/j.chbr.2022.100167>
- [19] Hein Duijf. 2021. Should One Trust Experts? *Synthese* 199, 3 (Dec. 2021), 9289–9312. <https://doi.org/10.1007/s11229-021-03203-7>
- [20] Serge Egelman and Eyal Peer. 2015. Scaling the Security Wall: Developing a Security Behavior Intentions Scale (SeBIS). In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. Association for Computing Machinery, New York, NY, USA, 2873–2882. <https://doi.org/10.1145/2702123.2702249>

- [21] Naomi Ellemers, Russell Spears, and Bertjan Doosje. 2002. Self and Social Identity. *Annual review of psychology* 53, 1 (2002), 161–186. <https://doi.org/10.1146/annurev.psych.53.100901.135228>
- [22] Jonathan St. B. T. Evans. 2003. In Two Minds: Dual-Process Accounts of Reasoning. *Trends in Cognitive Sciences* 7, 10 (Oct. 2003), 454–459. <https://doi.org/10.1016/j.tics.2003.08.012>
- [23] John Martin Fischer and Mark Ravizza. 1998. *Responsibility and Control: A Theory of Moral Responsibility*. Cambridge University Press.
- [24] Shane Frederick. 2005. Cognitive Reflection and Decision Making. *Journal of Economic perspectives* 19, 4 (2005), 25–42. <https://doi.org/10.1257/089533005775196732>
- [25] Alisa Frik, Juliann Kim, Joshua Rafael Sanchez, and Joanne Ma. 2022. Users’ Expectations About and Use of Smartphone Privacy and Security Settings. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems (CHI ’22)*. Association for Computing Machinery, New York, NY, USA, 1–24. <https://doi.org/10.1145/3491102.3517504>
- [26] Donald Gotterbarn. 2001. Informatics and Professional Responsibility. *Science and Engineering Ethics* 7, 2 (June 2001), 221–230. <https://doi.org/10.1007/s11948-001-0043-5>
- [27] Margaret Gratian, Sruthi Bandi, Michel Cukier, Josiah Dykstra, and Amy Ginther. 2018. Correlating Human Traits and Cyber Security Behavior Intentions. *Computers & Security* 73 (March 2018), 345–358. <https://doi.org/10.1016/j.cose.2017.11.015>
- [28] Joseph Hallett, Nikhil Patnaik, Benjamin Shreeve, and Awais Rashid. 2021. “Do This! Do That!, And Nothing Will Happen” Do Specifications Lead to Securely Stored Passwords?. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 486–498. <https://doi.org/10.1109/ICSE43902.2021.00053>
- [29] Julie M Haney, Mary F Theofanos, Yasemin Acar, and Sandra Spickard Prettyman. 2018. “We Make It a Big Deal in the Company”: Security Mindsets in Organizations That Develop Cryptographic Products. In *USENIX Conference on Usable Privacy and Security (SOUPS ’18)*. USENIX Association, Baltimore, MD, USA, 17. <https://doi.org/10.5555/3291228.3291257>
- [30] S. Alexander Haslam. 2012. The Social Identity Approach. In *Psychology in Organizations: The Social Identity Approach* (2 ed.). SAGE Publications Ltd, London, 17–39. <https://doi.org/10.4135/9781446278819>
- [31] Guido Hertel, Sven Niedner, and Stefanie Herrmann. 2003. Motivation of Software Developers in Open Source Projects: An Internet-based Survey of Contributors to the Linux Kernel. *Research Policy* 32, 7 (July 2003), 1159–1177. [https://doi.org/10.1016/S0048-7333\(03\)00047-7](https://doi.org/10.1016/S0048-7333(03)00047-7)
- [32] Michael Hogg and Graham Vaughan. 2017. *Social Psychology*. Pearson Education, Limited, Harlow, United Kingdom.
- [33] Matthew Ivory, John Towse, Miriam Sturdee, Mark Levine, and Bashar Nuseibeh. 2023. Recognizing the Known Unknowns; the Interaction Between Reflective Thinking and Optimism for Uncertainty Among Software Developer’s Security Perceptions. *Technology, Mind, and Behavior* 4, 3: Winter 2023 (Dec. 2023). <https://doi.org/10.1037/tmb0000122>
- [34] Daniel Kahneman. 2011. *Thinking, Fast and Slow*. Farrar, Straus and Giroux, New York, NY, USA.
- [35] Daniel Kahneman, Paul Slovic, and Amos Tversky (Eds.). 1974. *Judgment under Uncertainty: Heuristics and Biases* (1st ed.). Cambridge University Press, Cambridge, United Kingdom.
- [36] Daniel Kahneman and Amos Tversky. 1979. Prospect Theory: An Analysis of Decision under Risk. *Econometrica* 47, 2 (1979), 263–291. <https://doi.org/10.2307/1914185> arXiv:1914185
- [37] Iacovos Kirlappos, Adam Beaument, and M. Angela Sasse. 2013. “Comply or Die” Is Dead: Long Live Security-Aware Principal Agents. In *Financial Cryptography and Data Security*, David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Andrew A. Adams, Michael Brenner, and Matthew Smith (Eds.). Vol. 7862. Springer Berlin Heidelberg, Berlin, Heidelberg, 70–82. [https://doi.org/10.1007/978-3-642-41320-9\\_5](https://doi.org/10.1007/978-3-642-41320-9_5)
- [38] Markus Knauff and Ann G. Wolf. 2010. Complex Cognition: The Science of Human Reasoning, Problem-Solving, and Decision-Making. *Cognitive Processing* 11, 2 (May 2010), 99–102. <https://doi.org/10.1007/s10339-010-0362-z>
- [39] Mark Levine, Amy Prosser, David Evans, and Stephen Reicher. 2005. Identity and Emergency Intervention: How Social Group Membership and Inclusiveness of Group Boundaries Shape Helping Behavior. *Personality & Social Psychology Bulletin* 31, 4 (April 2005), 443–453. <https://doi.org/10.1177/0146167204271651>
- [40] Tamara Lopez, Helen Sharp, Thein Tun, Arosha K. Bandara, Mark Levine, and Bashar Nuseibeh. 2019. “Hopefully We Are Mostly Secure”: Views on Secure Code in Professional Practice. In *12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, Montreal, QB, Canada, 61–68. <https://doi.org/10.1109/chase.2019.00023>
- [41] Tamara Lopez, Helen Sharp, Thein Tun, Arosha K. Bandara, Mark Levine, and Bashar Nuseibeh. 2022. Security Responses in Software Development. *ACM Transactions on Software Engineering and Methodology* (Aug. 2022). <https://doi.org/10.1145/3563211>

- [42] Tamara Lopez, T. Tun, A. Bandara, L. Mark, B. Nuseibeh, and H. Sharp. 2019. An Anatomy of Security Conversations in Stack Overflow. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*. 31–40. <https://doi.org/10.1109/ICSE-SEIS.2019.00012>
- [43] Tamara Lopez, Thein Than Tun, Arosha K. Bandara, Mark Levine, Bashar Nuseibeh, and Helen Sharp. 2020. Taking the Middle Path: Learning About Security Through Online Social Interaction. *IEEE Software* 37, 1 (Jan. 2020), 25–30. <https://doi.org/10.1109/MS.2019.2945300>
- [44] Kai-Uwe Loser and Martin Degeling. 2014. Security and Privacy as Hygiene Factors of Developer Behavior in Small and Agile Teams. In *ICT and Society (IFIP Advances in Information and Communication Technology)*. Springer, Berlin, Heidelberg, 255–265. [https://doi.org/10.1007/978-3-662-44208-1\\_21](https://doi.org/10.1007/978-3-662-44208-1_21)
- [45] Donald G. Marquis. 1962. Individual Responsibility and Group Decisions Involving Risk. *Industrial Management Review* 3, 2 (1962), 8.
- [46] Marie McAuliffe and Anna Triandafyllidou. 2021. *World Migration Report 2022*. international organisation for migration, 17 Route des Morillons, 1211 Geneva 19, Switzerland.
- [47] Rahul Mohanani, Iflaah Salman, Burak Turhan, Pilar Rodríguez, and Paul Ralph. 2020. Cognitive Biases in Software Engineering: A Systematic Mapping Study. *IEEE Transactions on Software Engineering* 46, 12 (Dec. 2020), 1318–1339. <https://doi.org/10.1109/TSE.2018.2877759>
- [48] Kjetil Møløy and Magne Jørgensen. 2005. Expert Estimation of Web-Development Projects: Are Software Professionals in Technical Roles More Optimistic Than Those in Non-Technical Roles? *Empirical Software Engineering* 10, 1 (Jan. 2005), 7–30. <https://doi.org/10.1023/B:EMSE.0000048321.46871.2e>
- [49] Sohaib Mustafa, Wen Zhang, and Muhammad Mateen Naveed. 2023. What Motivates Online Community Contributors to Contribute Consistently? A Case Study on Stackoverflow Netizens. *Current Psychology* 42, 13 (May 2023), 10468–10481. <https://doi.org/10.1007/s12144-022-03307-4>
- [50] Clifford Mynatt and Steven J. Sherman. 1975. Responsibility Attribution in Groups and Individuals: A Direct Test of the Diffusion of Responsibility Hypothesis. *Journal of Personality and Social Psychology* 32, 6 (Dec. 1975), 1111–1118. <https://doi.org/10.1037/0022-3514.32.6.1111>
- [51] Alena Naiakshina. 2020. Don't Blame Developers! Examining a Password-Storage Study Conducted with Students, Freelancers, and Company Developers. (Dec. 2020).
- [52] Alena Naiakshina, Anastasia Danilova, Eva Gerlitz, Emanuel von Zeischwitz, and Matthew Smith. 2019. "If You Want, I Can Store the Encrypted Password": A Password-Storage Field Study with Freelance Developers. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–12. <https://doi.org/10.1145/3290605.3300370>
- [53] Duc Cuong Nguyen, Dominik Wermke, Yasemin Acar, Michael Backes, Charles Weir, and Sascha Fahl. 2017. A Stitch in Time: Supporting Android Developers in Writing Secure Code. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, Dallas Texas USA, 1065–1077. <https://doi.org/10.1145/3133956.3133977>
- [54] Steven P. Nichols. 1997. Professional Responsibility: The Role of the Engineer in Society. *Science and Engineering Ethics* 3, 3 (Sept. 1997), 327–337. <https://doi.org/10.1007/s11948-997-0039-x>
- [55] James Nicholson, Lynne Coventry, and Pam Briggs. 2018. Introducing the Cybersurvival Task: Assessing and Addressing Staff Beliefs about Effective Cyber Protection. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*. 443–457.
- [56] Daniela Seabra Oliveira, Tian Lin, Muhammad Sajidur Rahman, Rad Akefirad, Donovan Ellis, Eliany Perez, Rahul Bobhate, Lois A. DeLong, Justin Cappos, and Yuriy Brun. 2018. {API} Blindspots: Why Experienced Developers Write Vulnerable Code. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*. 315–328.
- [57] Daniela Seabra Oliveira, Marissa Rosenthal, Nicole Morin, Kuo-Chuan Yeh, Justin Cappos, and Yanyan Zhuang. 2014. It's the Psychology Stupid. In *Proceedings of the 30th Annual Computer Security Applications Conference*. Association for Computing Machinery, New Orleans, LA, USA, 296–305. <https://doi.org/10.1145/2664243.2664254>
- [58] Julius Pahlke, Sebastian Strasser, and Ferdinand M. Vieider. 2012. Risk-Taking for Others under Accountability. *Economics Letters* 114, 1 (Jan. 2012), 102–105. <https://doi.org/10.1016/j.econlet.2011.09.037>
- [59] Julius Pahlke, Sebastian Strasser, and Ferdinand M. Vieider. 2015. Responsibility Effects in Decision Making under Risk. *Journal of Risk and Uncertainty* 51, 2 (Oct. 2015), 125–146. <https://doi.org/10.1007/s11166-015-9223-6>
- [60] Hernan Palombo, Armin Ziaie Tabari, Daniel Lende, Jay Ligatti, and Xinming Ou. 2020. An Ethnographic Understanding of Software ({In}Security) and a {Co-Creation} Model to Improve Secure Software Development. In *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*. 205–220.
- [61] Andrew S. Patrick, A. Chris Long, and Scott Flinn. 2003. HCI and Security Systems. In *CHI '03 Extended Abstracts on Human Factors in Computing Systems (CHI EA '03)*. Association for Computing Machinery, New York, NY, USA, 1056–1057. <https://doi.org/10.1145/765891.766146>

- [62] Andreas Poller, Laura Kocksch, Katharina Kinder-Kurlanda, and Felix Anand Epp. 2016. First-Time Security Audits as a Turning Point? Challenges for Security Practices in an Industry Software Development Team. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA '16)*. Association for Computing Machinery, New York, NY, USA, 1288–1294. <https://doi.org/10.1145/2851581.2892392>
- [63] Andreas Poller, Laura Kocksch, Sven Törpe, Felix Anand Epp, and Katharina Kinder-Kurlanda. 2017. Can Security Become a Routine?: A Study of Organizational Change in an Agile Software Development Group. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*. ACM, Portland Oregon USA, 2489–2503. <https://doi.org/10.1145/2998181.2998191>
- [64] Evan Polman and Kaiyang Wu. 2020. Decision Making for Others Involving Risk: A Review and Meta-Analysis. *Journal of Economic Psychology* 77 (March 2020), 102184. <https://doi.org/10.1016/j.joep.2019.06.007>
- [65] Paul Ralph. 2011. Toward a Theory of Debiasing Software Development. In *Research in Systems Analysis and Design: Models and Methods (Lecture Notes in Business Information Processing)*, Stanisław Wrycza (Ed.). Springer, Berlin, Heidelberg, 92–105. [https://doi.org/10.1007/978-3-642-25676-9\\_8](https://doi.org/10.1007/978-3-642-25676-9_8)
- [66] Paul Ralph and Ewan Tempero. 2016. Characteristics of Decision-Making during Coding. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering (EASE '16)*. Association for Computing Machinery, New York, NY, USA, 1–10. <https://doi.org/10.1145/2915970.2915990>
- [67] Rémi Rampin and Vicky Rampin. 2021. Taguette: Open-Source Qualitative Data Analysis. *Journal of Open Source Software* 6, 68 (Dec. 2021), 3522. <https://doi.org/10.21105/joss.03522>
- [68] Irum Rauf, Tamara Lopez, Helen Sharp, Marian Petre, Thein Tun, Mark Levine, John Towse, Dirk van der Linden, Awais Rashid, and Bashar Nuseibeh. 2022. Influences of Developers' Perspectives on Their Engagement with Security in Code. In *2022 IEEE/ACM 15th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, Washington, D.C., USA, 86–95. <https://doi.org/10.1145/3528579.3529180>
- [69] Irum Rauf, Marian Petre, Thein Tun, Tamara Lopez, Paul Lunn, Dirk Van Der Linden, John Towse, Helen Sharp, Mark Levine, Awais Rashid, and Bashar Nuseibeh. 2021. The Case for Adaptive Security Interventions. *ACM Transactions on Software Engineering and Methodology* 31, 1 (Sept. 2021), 9:1–9:52. <https://doi.org/10.1145/3471930>
- [70] Irum Rauf, Dirk van der Linden, Mark Levine, John Towse, Bashar Nuseibeh, and Awais Rashid. 2020. Security but Not for Security's Sake. In *ICSEW'20: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*. Association for Computing Machinery, Seoul, Republic of Korea, 141–144. <https://doi.org/10.1145/3387940.3392230>
- [71] Richard M Ryan and Edward L Deci. 2000. Self-Determination Theory and the Facilitation of Intrinsic Motivation, Social Development, and Well-Being. *American Psychologist* 55, 1 (2000), 67. <https://doi.org/10.1037/0003-066X.55.1.68>
- [72] M. A. Sasse, S. Brostoff, and D. Weirich. 2001. Transforming the 'Weakest Link' — a Human/Computer Interaction Approach to Usable and Effective Security. *BT Technology Journal* 19, 3 (July 2001), 122–131. <https://doi.org/10.1023/A:1011902718709>
- [73] Christine Satchell and Paul Dourish. 2009. Beyond the User: Use and Non-Use in HCI. In *Proceedings of the 21st Annual Conference of the Australian Computer-Human Interaction Special Interest Group: Design: Open 24/7*. ACM, Melbourne Australia, 9–16. <https://doi.org/10.1145/1738826.1738829>
- [74] Helen Sharp, Hugh Robinson, and Mark Woodman. 2000. Software Engineering: Community and Culture. *IEEE Software* 17, 1 (Jan. 2000), 40–47. <https://doi.org/10.1109/52.819967>
- [75] Ben Shreeve, Catarina Gralha, Awais Rashid, João Araujo, and Miguel Goulão. 2022. Making Sense of the Unknown: How Managers Make Cyber Security Decisions. *ACM Transactions on Software Engineering and Methodology* (Aug. 2022). <https://doi.org/10.1145/3548682>
- [76] S.W. Smith. 2003. Humans in the Loop: Human-Computer Interaction and Security. *IEEE Security & Privacy* 1, 3 (May 2003), 75–79. <https://doi.org/10.1109/MSECP.2003.1203228>
- [77] Mohammad Tahaei, Adam Jenkins, Kami Vaniea, and Maria Wolters. 2020. "I Don't Know Too Much About It": On the Security Mindsets of Computer Science Students. In *International Workshop on Socio-Technical Aspects in Security and Trust*. Springer, Copenhagen, Denmark, 20. <https://doi.org/10.1007/978-3-030-55958-8>
- [78] Mohammad Tahaei and Kami Vaniea. 2019. A Survey on Developer-Centred Security. In *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, Stockholm, Sweden, 129–138. <https://doi.org/10.1109/EuroSPW.2019.00021>
- [79] Henri Tajfel and John C. Turner. 2010. An Integrative Theory of Intergroup Conflict. In *Rediscovering Social Identity* (1st ed.), Tom Postmes and Nyla Branscombe (Eds.). Routledge, London, United Kingdom, 56–65.
- [80] Antony Tang. 2011. Software Designers, Are You Biased?. In *Proceedings of the 6th International Workshop on SHaring and Reusing Architectural Knowledge*. ACM, Waikiki, Honolulu HI USA, 1–8. <https://doi.org/10.1145/1988676.1988678>
- [81] Martin Tanis and Tom Postmes. 2005. A Social Identity Approach to Trust: Interpersonal Perception, Group Membership and Trusting Behaviour. *European Journal of Social Psychology* 35, 3 (2005), 413–424. <https://doi.org/10.1002/ejsp.256>



- [82] Alberto Luiz Oliveira Tavares de Souza and Victor Hugo Santiago Costa Pinto. 2020. Toward a Definition of Cognitive-Driven Development. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 776–778. <https://doi.org/10.1109/ICSME46990.2020.00087>
- [83] Edith Tom, Aybüke Aurum, and Richard Vidgen. 2013. An Exploration of Technical Debt. *Journal of Systems and Software* 86, 6 (June 2013), 1498–1516. <https://doi.org/10.1016/j.jss.2012.12.052>
- [84] John C. Turner, Rupert Brown, and Henri Tajfel. 1979. Social Comparison and Group Interest in Ingroup Favouritism. *European Journal of Social Psychology* 9, 2 (1979), 187–204. <https://doi.org/10.1002/ejsp.2420090207>
- [85] Dirk van der Linden, Pauline Anthonysamy, Bashar Nuseibeh, Thein Tun, Marian Petre, Mark Levine, John Towse, and Awais Rashid. 2020. Schrödinger's Security: Opening the Box on App Developers' Security Rationale. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 149–160. <https://doi.org/10.1145/3377811.3380394>
- [86] Veracode. 2020. *State of Software Security*. Technical Report 11. Veracode.
- [87] Peter Wason. 1960. On the Failure to Eliminate Hypotheses in a Conceptual Task. *Quarterly Journal of Experimental Psychology* 12, 3 (July 1960), 129–140. <https://doi.org/10.1080/17470216008416717>
- [88] Neil D. Weinstein. 1980. Unrealistic Optimism about Future Life Events. *Journal of Personality and Social Psychology*, 39, 5 (1980), 806–820. <https://doi.org/10.1037/0022-3514.39.5.806>
- [89] Charles Weir, Ingolf Becker, and Lynne Blair. 2021. A Passion for Security: Intervening to Help Software Developers. In *International Conference on Software Engineering: Software Engineering in Practice*. IEEE, Madrid, Spain, 21–30. <https://doi.org/10.1109/ICSE-SEIP52600.2021.00011>
- [90] Robert Wood and Albert Bandura. 1989. Social Cognitive Theory of Organizational Management. *Academy of Management Review* 14, 3 (July 1989), 361–384. <https://doi.org/10.5465/amr.1989.4279067>
- [91] Marvin Wyrich, Andreas Preikschat, Daniel Graziotin, and Stefan Wagner. 2021. The Mind Is a Powerful Place: How Showing Code Comprehensibility Metrics Influences Code Understanding. *arXiv:2012.09590 [cs]* (Feb. 2021). [arXiv:cs/2012.09590](https://arxiv.org/abs/2012.09590)
- [92] Shundan Xiao, Jim Witschey, and Emerson Murphy-Hill. 2014. Social Influences on Secure Development Tool Adoption: Why Security Tools Spread. In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing*. Association for Computing Machinery, Baltimore, MD, USA, 1095–1106. <https://doi.org/10.1145/2531602.2531722>
- [93] Jing Xie, Heather Richter Lipford, and Bill Chu. 2011. Why Do Programmers Make Security Errors?. In *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 161–164. <https://doi.org/10.1109/VLHCC.2011.6070393>
- [94] Iyad Zayour, Imad Moukadem, and Issam Moghrabi. 2013. Complexity Is in the Brain of the Beholder: A Psychological Perspective on Software Engineering's Ultimate Challenge. *Journal of Software* 8, 5 (May 2013), 1079–1085. <https://doi.org/10.4304/jsw.8.5.1079-1085>

## 7 APPENDIX

Primary Code	Secondary Code	Description	Example
Cognition	decision making	Reference to the need to make decisions in a conscious manner	“any gambles I would take would be decisions to stray to conventional UI/UX methodology or incorporate different new agile practices and ways of working”
Cognition	cognitive bias	Reference to cognitive biases	“i perform tests for the applications that i make, i always try to handle all the security issues that i have come to be accustomed with before going forward with the application deployment”
Development Stages	training	Reference to training	“Treating user security as a priority at the beginning of a software project is a good way to ensure that there is some measure of protection. Another policy would be to require ongoing security training and education for the whole software development team”
Development Stages	time	Mentions of time as a factor in development	“major example of this would be when you are dealing with time constraints. Most companies have only so much time to create new programs, or else they risk financial issues. So oftentimes a development teams has to make the decision of whether to rush a program to the finish line or if they should spend extra time making sure everything works fine”
Development Stages	ethics	Mentions of ethical or moral ideas in line with data handling or software development	“there are also more malicious risks in this area; some less morally-conscious programmers may put their customers/users at risk by taking a gamble on the security side of things”

Development Stages	testing	software testing phase	“Sometimes to improve security we should allow intruders to hack the system, in meanwhile we need to track their activity. In this way, we can have an idea of security flaws. Above technique can be achieved through white hat hackers, white hat hackers need to be recruited and allowed to security test the system. It is quite intuitive that hacker will then know about the security flaws but at the same time we need to trust the person to get the better idea of our own security flaws and to avoid major security flaws. ”
Development Stages	planning/research	Any actions denoting a planned approach to development	“The risk of potentially going wrong is with us every time, but It can be reduced by taking prior steps, it starts from requirement gathering and then development methodologies”
Domain	education	a formal knowledge acquisition that is separate from training in the development workplace	“there is always the chance that I would get counter hacked or just plan hacked. I could lose the equipment in a flash, or I could get screwed over by a larger company. At the least, this would all just be learning experience. ”
Domain	finance	mentions of financial consequences from actions, budgets or consideration of money in their development cycle	“Risk of financial losses is the one if the deadline was not met by our team. This will affect our team since it will inject stress to the team members knowing that financial losses will occur if deadline was not met”
Domain	law	Reference to legal matters	“Data handled by an application could be mishandled in other ways, causing: sensitive data from being made unintentionally available to unauthorised users, or data to be corrupted. If data protection regulations are broken, I would have broken the law, and my reputation (and wallet) will suffer as a result”

Emotion	positive emotion	Self-experienced positive emotion	“I am super happy with this projects as a professional because I helped my client to save millions of dollars. As a result, my client is going to pay out bonuses to their employees and I continued to be awarded with new project enhancements to work with”
Emotion	negative emotion	Self-experienced negative emotion	“I often feel overwhelmed by the fear of facing damage to my professional reputation. At the same time I am also at risk of having all my hard work in vain because of functional failure of the web application”
Functionality	priority	reference to functionality being the priority within the work	“some individuals(Junior Dev) who are working on features make decisions to use the open-source packages/dependencies for faster development. I think this gamble is not worth it if security is a major concern. ”
Functionality	failure	Reference of failing functionality	“always the chance that something minute has gone wrong and whatever was made will not run properly; I find that smaller problems are often the trickiest to deal with, because it’s harder to find what went wrong”
Functionality	general	Reference to general functionality within the software	“when i have to take data from client and to send data to apis, i have check the data type also.. like if there is string type needed in api and i am sending number, so i have to validate that at the user end”
Interpersonal	team	Reference to team work or team mates	“As a team, a huge gamble I found is releasing early alphas for free testing, it depends on your clients, if they are not tech savey, they might not understand what a bug is, or features that are planned might be cut off and they might assume that they might never come or will take longer to developer than realistic”
Interpersonal	third party (people)	reference to third party people, such as contractors for pen-testing	“we make our research to consider the most important security rules, and incase needed we will be hiring security experts to provide us with the necessary information and tools to protect our solution”

Interpersonal	communication with peer	Communication between peers, managers or those who act within the development area	“Another gamble worth taking is consulting with multiple developers and having your code checked and seen if it could be improved by anybody. Vice versa a gamble not worth having is not doing that and having potential trouble in the future, it's just not worth it”
Interpersonal	communication with client	Communication with clients, those requesting or using software. Clients/Users are considered as similar in this context	“in the life cycle of software developments there's a stage where you receive the product requirements from the clients. So what in most cases, these products requirements are not are not really exhaustive. So there will surely be chunks left out that you would have to fill in yourself. Filling this in would be, um, can be called the gamble because it might go wrong. It might go right”
Interpersonal	(third party) product	Mentions of third party products or services	“Laravel seemed to take care of a lot of security concerns and having read a bunch of articles online, i knew what to do and what not to do. ”
Optimism	optimism	Indications of optimism bias, such as that they could never be insecure	“I'm not afraid of functional failure because by the time the app is finished and launched I'll have to perform all the tests needed just to be sure that after launching, everything will go smoothly”
Responsibility	Self	Responsibility for actions or consequences falls to an individual, specifically a software developer	“anything that involves a breach of customer data or authority can have serious follow on repercussions. The key issues to me are the ethical dimension to the end user, by losing the data your compromising some part of their life that you wouldn't want to. This has a moral and emotional personal impact. Outside of this there's a reduction of professional standing for being responsible for such a breach”

Responsibility	Other (external to company)	Responsibility is passed to someone outside of the development sphere, e.g. clients or users. NOT managers or other teams	“I would allow users the right to decide whether they want to add extra layers of protection such as two-factor authentication for their account, because some users prefer the convenience of not having to authenticate their account with each login”
Responsibility	Other (person internal to company)	Reference of responsibility belong to someone else within the development cycle (such as peers or colleagues)	“the IT Security team must ensure that every payment is secure and bank accounts of the clients are not in danger”
Responsibility	Other (object or service internal to company)	Responsibility that is passed to internally used objects - this relates to APIs, frameworks or other tools that are considered to be part of the software development domain	“I usually get different third party programs/services to check over my software to make sure that it is safe and that there isn't really any problems or issues with it that could lead to a vulnerability. My reasoning is that in my opinion, it's definitely easier to do it this way and it is typically faster than doing it urself”
Risk	general	Mention of actions, behaviours or consequences that confer risk that are out of the control of the developers. The existence of risk without mention of mitigation or appetite	“The main risk is functional failure because if the client requirements are not met then the reason for the application is not complete and might force them to either terminate the arrangement or even require financial returns”
Risk	Self risk	Consequences pose a risk to the individual who started the action	“We are working only with the enterprise companies so there is a huge risk of working with such clients because you have to deliver 100% exact thing which they have in their mind. and if you are fail to do this your reputation , money , time is lost”

Risk	Mitigation/reduction	Reference to efforts made to reduce risk within the workflow.	“if any third-party tools are being used or any other service is used to deploy to server first research about the service provider, their history reviewed. after being clear about the security of the system then only that service is used”
Risk	Risk to others	Consequences pose a risk to others outside of the development sphere, such as clients	“When the product is finished I launch series of tests depending on the functionality of the app. I sometimes will try to use pentest tools to check some parts of the app or even consult with more experience pentesters. As app developers, we are sometimes responsible for some intimate parts of human life (of the users). So by not protecting the app properly, we may easily ruin someone's life, and that's a huge burden to carry”
Risk	Appetite	Mentions of risk taking behaviour	“we don't take any special processes to identify risks or security vulnerabilities. I'm not sure what the exact reasoning behind these decisions are but I think we are more focused on developing and maintaining the software”
Risk	Avoidance	Reference to gambles, risks or uncertainty being untenable. Absolutism regarding there must be an absence of risk	“Any risk that reduces security would never be worth attempting unless there is little to no data or information that would result in possible problems if taken”
Security	general	Reference of security without specific mention of priority or what kind	“security of the endpoints exposed were taken care by either basic authentication or oauth authentication depending on the business needs. Some data was also being store on the cloud and the fields involving personal data was gdpr tagged”

Security	priority	Security referred to as a priority	“You must first of all protect your code give access to only who’s working on that project then think of CORS and who can use server APIS then think of authentication then think about failure on logic ,then may test code against SQL injecting and attacks and test logic too you have test many times before launch app”
Security	bad actor	Active attempts to penetrate software from bad actors.	“Also exposing my customers to risk, by threat actors either utilising vulnerabilities in the application to target them, or by fully compromising the solution and then initiating attacks against them from a server stand point”
Security	Secondary (not priority)	Mention or reference to security being less than important, particularly under time pressure and being given less importance than before	“The important of usability was way ahead of security and i had a strict schedule for release - so i had minimum of time focusing on security, i mainly relied on Laravel to do the work for me (which is limited i guess)”
Security	bad code	Mention of code that is weak in security, but not through bad actors, but rather poor implementation	“There were some APIs that are not hidden away in a VPC because I didn’t have the time to configure them”
Security	privacy	Mentions of data privacy	“The recent software that we released on our customer is a Passport Management System. This system manages citizen data and security is very important on the development. The team need to make sure that no citizen data will be exposed from Enrolment to the Delivery of the citizen passport. To make sure that this is properly implemented, security checks were performed all through out the project duration. ”



## 7.1 Statement of Continuous Thesis Summary

*“Developers are responsible for the software they implement... Getting developers to care about security is essential in order to produce software which protects its users.” - Naiakshina et al., 2017*

This chapter used the same dataset as Chapter 6 but explored the rich text responses through a thematic analysis. This chapter reports three themes: Responsibility, Optimism, and Risk. These are highlighted as potential ways to interpret developers’ perspectives on developer actions and behaviours and not as definitive ways of understanding how freelance developers perceive risk.

Chapters 6 and 7 utilised different analytic approaches, with Chapter 6 adopting a quantitative, more confirmatory approach and Chapter 7 following a qualitative exploration of the data, and together they offer complementary analyses. There was a risk that using a mixed-methods approach would have yielded divergent findings, complicating the research in this area, but this was not seen. Concepts such as risk awareness were detected through keywords in Chapter 6 and the thematic analysis which highlighted the different ways that risk was mentioned. These two chapters demonstrate that a mixed-methods analysis provides a more exhaustive investigation of risk perceptions.

Social identity theory (Abrams & Hogg, 1990) is relevant to secure software development, but it has only been linked to software development a handful of times (e.g., Backevik et al., 2019; Rauf et al., 2021). In an exploration of social identity in robotics software engineering, Gavidia-Calderon et al. (2023) found that ethics is often deprioritised, but conforming with a group identity increases personal responsibility, meaning that engineers are more likely to act ethically.

With the current data, risky software behaviours declined when developers either felt a personal risk or perceived a shared identity with software users, evidencing the role of social identities in modifying behaviours. The social identity approach was identified as a

potential fundamental theory in phase 1, and this chapter provides support for its relevance.

This chapter also strengthens the overall thesis contribution by demonstrating that the secure behaviour of developers cannot be fully understood without accounting for social context. In contrast to the previous chapter focused on cognition, this chapter shows how group affiliation, specifically identification with users or the development community, can motivate ethical and responsible coding practices. In doing so, it brings together domains such as responsibility and communication with psychological theories that explain how and why these skills manifest in behaviour.

In previous chapters, I have focused on exploring the *perceptions* of people working within software engineering, with less opportunity to study more direct behaviours and coding choices observed in software engineering. In the following and final research chapter, I examine how dual processing theory applies to code comprehension tasks. Code comprehension is a core facet of software development, as developers are frequently expected to work with existing code. I take an existing hypothesis that security vulnerabilities are typically handled through system 1 processing (Cappos et al., 2014), and I apply as-yet unused (in software engineering) measures for dual processing theory to understand the relationship between the ability to detect insecure code and the propensity to engage in reflective styles of cognition.

### ***7.1.1 Contribution to Thesis Argument and Forward Trajectory***

This chapter advances the thesis argument by introducing social identity theory as a critical and relevant lens for understanding secure software behaviours. While previous chapters emphasised cognition and reasoning (such as Chapter 6), this chapter shows that social context and group identification also play a key role. Specifically, developers demonstrated greater responsibility and reduced risky behaviours when they identified with end users or felt personally accountable, which is an insight that adds a socially-grounded dimension to the psychological profile of secure software engineering.

This supports the thesis's broader claim that soft skills like responsibility, ethics, and communication are deeply influenced by both cognitive styles and social affiliations.

Importantly, this chapter continues to build the integrative case that secure software development must be understood not merely through technical competence or isolated traits, but through the interaction of psychological constructs, group dynamics, and socially constructed values. As such, this chapter moves the thesis toward a more complete behavioural model of software engineering; one that accounts for how developers think, feel, and align themselves socially when making decisions.

To this point, the thesis has progressed from identifying soft skills and their development, to exploring the cognitive and social psychological dimensions that underpin those skills in security-relevant contexts. Together, Chapters 6 and 7 show that reflective thinking and group-based responsibility are not only conceptually important but behaviourally impactful in shaping how software engineers engage with risk.

The next and final empirical chapter further extends the application of dual processing theory, this time in the context of code comprehension tasks. Shifting from self-reported behaviours to more direct measures of secure coding performance, it explores whether the propensity for reflective thinking predicts the ability to detect security vulnerabilities in code. This chapter builds on prior findings by assessing the observable outcomes of reflective cognition in practical coding scenarios closing the loop between perception, psychology, and practice that has been evidenced in the previous chapters.

## 8 Software Vulnerabilities as Cognitive Blindspots; assessing the suitability of a dual processing theory of decision making for secure coding

Ivory, M., Towse, J., Sturdee, M., Levine, M., & Nuseibeh, B. (*in review*). Software Vulnerabilities as Cognitive Blindspots; assessing the suitability of a dual processing theory of decision making for secure coding. *Transactions on Software Engineering and Methodology*. <https://doi.org/10.31234/osf.io/v7fqb>

## Software Vulnerabilities as Cognitive Blindspots; assessing the suitability of a dual processing theory of decision making for secure coding

MATTHEW IVORY, Lancaster University, Great Britain

JOHN TOWSE, Lancaster University, Great Britain

MIRIAM STURDEE, University of St Andrews, Scotland

MARK LEVINE, Lancaster University, Great Britain

BASHAR NUSEIBEH, Lero, Republic of Ireland and Open University, United Kingdom

Security vulnerabilities are present in many software systems, putting those who entrust software with their data in harm's way. Many vulnerabilities are avoidable since they are not new and are well-described. Despite this awareness, they remain widespread. One hypothesis for their persistence is that they represent software blindspots, problems that are implicit in the mental models of developers and thus escape attention (Brun et al., 2023; Oliveira et al. 2018). Our current understanding of how cognitive influences secure coding is limited, and we address this by extending the hypothesis by suggesting differences in decision making approaches alter the ability to detect vulnerabilities. Through an empirical study and power analysis, we show the potential value of dual processing theory, where individuals make decisions using one of two cognitive systems: a default system reliant on heuristics and intuitive mechanisms, and a more deliberate and computational interventionist system. This preregistered study replicates key predictions from previous blindspot research, extends the analysis towards cognition, and models effect sizes of variables that might impact software security. We complement this analysis with data simulations to expose the sampling scale of empirical studies that would be necessary for highly powered work in this domain.

CCS Concepts: • **Security and privacy** → **Social aspects of security and privacy**; • **Human-centered computing** → *Empirical studies in HCI*;

Additional Key Words and Phrases: security, cognitive psychology, dual processing theory, code comprehension, blindspots

### ACM Reference Format:

Matthew Ivory, John Towse, Miriam Sturdee, Mark Levine, and Bashar Nuseibeh. 2024. Software Vulnerabilities as Cognitive Blindspots; assessing the suitability of a dual processing theory of decision making for secure coding. 1, 1 (February 2024), 25 pages. <https://doi.org/10.475/123.4>

---

Authors' addresses: Matthew Ivory, Lancaster University, Department of Psychology, Lancaster, Lancashire, LA1 4YW, Great Britain, [matthew.ivory@lancaster.ac.uk](mailto:matthew.ivory@lancaster.ac.uk); John Towse, Lancaster University, Department of Psychology, Lancaster, Lancashire, LA1 4YW, Great Britain, [j.towse@lancaster.ac.uk](mailto:j.towse@lancaster.ac.uk); Miriam Sturdee, University of St Andrews, School of Computer Science, North Haugh, St Andrews, LA1 4YW, Scotland, [ms535@st-andrews.ac.uk](mailto:ms535@st-andrews.ac.uk); Mark Levine, Lancaster University, Department of Psychology, Lancaster, Lancashire, LA1 4YW, Great Britain, [mark.levine@lancaster.ac.uk](mailto:mark.levine@lancaster.ac.uk); Bashar Nuseibeh, Lero, Ireland, Castletroy, Co. Limerick, LA1 4YW, Republic of Ireland, Open University, United Kingdom, [Bashar.Nuseibeh@open.ac.uk](mailto:Bashar.Nuseibeh@open.ac.uk).

---

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

© 2024 Association for Computing Machinery.

Manuscript submitted to ACM

## 1 INTRODUCTION

Software vulnerabilities are a pervasive issue within software engineering, and they can have serious real-world consequences, such as impact on urgent medical care [71] or psychological harm [55]. Many of the vulnerabilities detected in modern software are not new. Well-known vulnerabilities are detected in over three-quarters of software, indicating that despite their notoriety, they remain unhandled despite the tools available to software developers. To address this, we offer support to the hypothesis that vulnerabilities exist as cognitive blindspots in decision making during secure coding.

Secure coding is the practice that ensures software “does not contain known vulnerabilities” [63], and vulnerabilities are defined as unexpected logic flows which create compromising software behaviours, allowing for unintended access to information or functionality [33]. It is observed that technical expertise and experience are poor indicators of predispositions towards secure coding [10], as is security knowledge [50]. Acknowledging that technical expertise has minimal effect on producing secure code, we should explore more psychological factors to explain how individuals can identify vulnerabilities during code reviews.

One psychologically-motivated explanation focuses on software engineers’ cognitive capabilities and the cognitive salience of critical choices. We explore the role of human decision making and cognitive psychology to understand individual differences in secure coding predispositions better. By treating software engineers as a diverse population displaying individual cognitive differences, and who must satisfice [70], we can better understand how cognition influences vulnerability detection. One theory is that security vulnerabilities are systematically missed by developers because they occupy blindspots in their attention and information processing.

In a software engineering context, a blindspot is code where the expected behaviour of a function diverges from the intended behaviour. It has been suggested that vulnerabilities in software code exist because of these blindspots [13]. Oliveira et al. [51] tested the hypothesis with measures of working memory and cognitive processing speed. They asked engineers to solve short Java programming puzzles that contained either a blindspot in the form of insecure API use or no blindspot. They found puzzles without a blindspot were solved significantly more often than puzzles with a blindspot, supporting the general hypothesis that vulnerabilities are often missed, but neither working memory nor processing speed affected accuracy. Brun et al. [10] replicated the study using Python, confirming the previous finding and demonstrating this phenomenon as language agnostic. They saw limited effect on blindspot detection with the same measures of cognition, only finding a significant effect on long-term memory capacity. This body of research explored cognition but did not provide information on *why* cognitive blindspots exist in security contexts. We propose to address this issue by applying the dual processing theory of decision making [21] towards the same code comprehension tasks used.

Dual processing theory posits individuals make decisions using two distinct systems, a default intuitive system that relies upon heuristics, and a second system that uses all available information to make reasoned choices [22]. In taking the position that decision making processes are not homogeneous and that individual differences are present in general populations [23], we seek to test whether this heterogeneity influences secure coding. Dual processing theories of decision making are relatively absent within software engineering research despite having a proven influence on performance in general judgments [77]. Only one known study has explored dual processing theory in software engineers that found a complex relationship between

Software Vulnerabilities as Cognitive Blindspots; assessing the suitability of a dual processing theory of decision making for secure coding 3

risk sensitivity and system 2 engagement [33]. To date, software tasks have remained unexplored, which is addressed in this paper.

In this study, we offer a partial replication of previous findings by Cappos et al. [13], Oliveira et al. [51], and Brun et al. [10] that blindspot presence results in reduced comprehension and that technical expertise has little effect in explaining blindspot detection. We extend their hypothesis by applying a dual processing theory of decision making. We use a modest sample of 37 participants to test the potential of dual processing theory in software research, recognising that supporting evidence is sparse, and so we use the data to conduct power analyses to identify the necessary samples required for cognitive psychology in secure coding research. This study offers a greater connection between software engineering and the individual differences observed in decision making competencies.

We successfully replicate findings that blindspots are difficult to detect, and that detection cannot be explained by technical knowledge or experience. We offer mixed results for the application of dual processing, but this was primarily due to data insensitivity (highlighting the need for larger samples), and we do not reject dual processing theory’s value for explaining secure coding.

The paper is structured as follows. In section 2, we essential psychological background content relevant to the theoretical underpinnings of this study. Section 3 details the hypotheses, relevant methodological decisions, measures, and procedures for research transparency. We report our results in section 4, which are discussed in terms of their theoretical support and implications for software engineering practice and research in section 5. To contextualise our study within the wider software engineering literature, we explore related work in section 6. Section 7 summarises the study’s contributions.

## 2 BACKGROUND

This section covers the essential psychological content that contextualises the research. This paper defines *cognitive blindspots* as units of information that are systematically overlooked, typically through unconscious or intuitive approaches to making judgments, resulting in non-optimal decisions. Blindspots are due to the human disposition to deploy decision making *heuristics* (mental short-cuts). Where heuristics lead to fallacies or erroneous thinking, they create biases. These are systematic, flawed judgments that deviate from ideal performance [20].

Biases are present throughout the whole software development process [62], and their influence can result in non-optimal decisions being made. In the example of tool selection, developers tend to use familiar tools [68], even if they are not appropriate for specific tasks [5]. No single tool can detect all software vulnerabilities meaning that a familiarity bias can result in non-optimal decisions [19].

Human working memory is sharply limited [7], creating the potential for overload. Heuristics help manage the cognitive complexity of the decision space by rendering the problem down to a simplified form [9]. However, because the mental representations of the task are now partial, incomplete, or non-exhaustive [36], important facets can become obscured to the mind’s eye, creating cognitive blindspots. From a psychological perspective, this means that vulnerabilities that are known “in principle” to a developer are hidden from view during the software task.

The presence of heuristics, biases, and attentional blindness all point towards the presence of more than one system of cognitive information processing, because we recognise different approaches to decision making,

where choices can be made intuitively and unconsciously, or we can exert effort and make more careful, reasoned choices. These differences can be explained by the dual processing theory.

The default-interventionist psychological model of dual processing theory proposes two cognitive systems [22]: the default and *intuitive* system 1 and a more intentional and *rational* system 2 that is only engaged when sufficiently cued [16]. System 1 processing is the default style of decision making, driven by heuristics to reduce complex judgments into simpler cognitive operations [37]. In contrast, system 2 processing is more deliberate but computationally demanding and is typically deployed when individuals seek an optimal solution by using all available information. System 2 is the interventionist mode, and it only overrides System 1 when an individual consciously or unconsciously sees a greater need for accuracy. System 1 is liable to generate simpler, less complete mental models than System 2 [35], and blindspots can reside in the created gaps.

### 3 METHODOLOGY

This section presents the preregistered hypotheses that motivated the following research design choices. The sample and recruitment strategy are reported, followed by the materials and study design.

We present this study as the groundwork required for deploying psychological measures in software engineering. We utilise a modest sample of 37 participants to assess the suitability and practicality of deploying dual processing theory in secure software engineering research. We recognise that the sample is not sufficient to translate into meaningful, practical implications (which is avoided in the discussion), but the sample is used to simulate data and conduct power analyses to determine the suitable effect sizes for future research. Choosing not to collect large numbers of participants for this study ensured that resources were managed suitably, which is necessary when working with software engineers who require a high compensation rate to participate in research.

#### 3.1 Hypotheses

The study was preregistered, lodging the design and hypotheses in advance, which can be found here: [doi.org/10.17605/OSF.IO/CE78G](https://doi.org/10.17605/OSF.IO/CE78G). Hypothesis 1 is a replication of Brun et al. [10] used to test the effect of blindspots in Python code. Hypothesis 2 applies cognitive reflection measures to assess the propensity for dual processing. Hypothesis 3 uses optimism bias as a proxy for a tendency towards system 1 processing. Hypothesis 4 and 5 use scales of rational and intuitive decision making to test the strength of an individual's tendency towards either cognitive system. Hypothesis 6 replicates the hypothesis from Brun et al. relating to the self-reported measures of puzzle interaction. Hypotheses 7 and 8 are based on previous findings that experience and declarative knowledge do not moderate vulnerability detection.

1. Python puzzles containing API blindspots will be more difficult to solve than scenarios without blindspots correctly.
2. Developers with higher levels of cognitive reflection will solve scenarios with blindspots more effectively than developers with lower cognitive reflection.
3. Developers demonstrating realistic levels of optimism will solve scenarios with blindspots more effectively than developers with higher levels of optimism.



4. Developers with higher levels of rational decision making will solve scenarios with blindspots more effectively than developers with lower levels of rational decision making.
5. Developers with lower levels of an intuitive decision making style will solve scenarios with blindspots more effectively than developers with higher levels of intuitive decision making.
6. Developers' perceived ratings of puzzle difficulty, effort, familiarity, and confidence will affect their ability to solve scenarios containing blindspots.
7. Developers with more experience and familiarity with programming and Python will show no difference in their ability to solve scenarios with API blindspots.
8. Developers with more cybersecurity knowledge and experience will show no differences in their ability to solve scenarios with API blindspots.

### 3.2 Participants

For this study, a sample of 37 participants was used, which is suitably powered to assess the replication in hypothesis 1 and offers adequate data to simulate further data to estimate the required scale of empirical work to address psychological questions in software engineering. Participants were recruited from Upwork.com, a freelancing website. Adverts invited freelancers to submit proposals for a “Code Review”. Five adverts were placed to promote visibility (when one stopped receiving proposals, it was archived, and an identical advert uploaded). The collection process took 21 days. As part of the submission, potential participants confirmed they were over 18, had at least one year’s experience with Python that was not limited to a university course, had experience with API functions, and asked to describe a recent Python project briefly. The advertising materials and information sheet deceived participants about the study’s true intention to avoid potential biasing. The materials described the study as interested in general code comprehension and its association with cognitive function. The Faculty-level ethics committee approved the deception. Participants received \$28 as compensation for their time and effort.

In total, 126 proposals were submitted, and 89 were rejected. Rejection criteria were not formalised before data collection to ensure that only high-quality participants were selected. Examples of rejected proposals included suspicion of duplicate accounts (with two or more different applicants giving almost identical answers and no prior Upwork experience), use of AI-generated content for screening questions (determined via online checkers or manual inspection), a failure to mention appropriate experience, a lack of English fluency (assessed from responses and further communication), or failure to convince the primary researcher that they fulfilled the criteria properly. In cases where it was unclear whether participants were to be approved, further communication was established to determine whether a proposal would be accepted. Upwork profiles, job history, and linked profiles (e.g., GitHub) were also used to determine eligibility. Due to this rigorous selection process, no attrition was seen for accepted participants.

The mean age of the 37 participants was 29.11 (SD = 9.55), ranging from 19 to 71. One participant preferred not to report gender, 35 reported male, and one self-described as “freelancer”. This overwhelming male-dominated sample was not targeted nor intended. Participant ethnicity is reported in Table 1, with the vast majority being White or Asian. Two participants had no education beyond high school level, two had some university experience, 18 had an undergraduate degree, 3 had some postgraduate university education, and 12 had postgraduate degrees. The average experience in general programming was 6.05 years (SD = 4.67), and Python experience was less at 4.39 years (SD = 2.46).

Table 1. Demographic breakdown of participant ethnicity

Ethnicity	Count
Asian (Indian, Pakistani, Bangladeshi, Chinese, any other Asian background)	14
Mixed two or more ethnic groups	2
Other (Arab or any others)	3
Prefer not to say	2
White	16

### 3.3 Materials

This section describes the psychological measures and code comprehension puzzles used for assessing blindspot detection. This study utilised an observational survey design where participants completed various measures without assignment to different experimental groups.

Measures of dual processing theory include the cognitive reflection test [23] and the rational and intuitive subscales of the general decision making style scale [67], both of which are described below. Cognitive reflection has been identified to play a role in explaining risk awareness around security in populations of software developers [33], and the rational decision making subscale has been identified as a predictor of good users' behaviours [28]. These psychological measures have prior use in security research and offer an opportunity to interpret complex behaviours.

**3.3.1 Cognitive Reflection Test.** The Cognitive Reflection Test (CRT; Frederick [23]) is a three-question test designed for measuring cognitive reflection, an individual's tendency to engage in system 2 processing during decision making. The questions offer intuitive but incorrect responses, and only through system 2 engagement is the correct answer identified. The most well-known CRT question is, "A bat and a ball cost \$1.10 in total. The bat costs \$1.00 more than the ball. How much does the ball cost?". The intuitive and incorrect response of 10 cents is the most common answer [23], instead of the correct answer of five cents, implying that system 1 processing is the default processing style. To the authors' knowledge, CRT has only been applied once before with software developers where an interaction between cognitive reflection and optimistic perceptions of software security was detected [33].

We used two cognitive reflection tests, CRT and CRT-2. The CRT was initially developed by Frederick [23], and the CRT-2 by Thomson and Oppenheimer [76]. The second version was designed to reduce the numerical nature of the original test. In the present study, the wording was altered, so attempts to find answers via online searches would be more difficult, but the question design remained unaltered. Participants were also asked whether they had seen the questions before to measure whether they may have been responding based on previous experience; however, this is not a significant concern as research has shown CRT performance is stable over time [74]. Response times were also recorded.

**3.3.2 OWASP Vulnerability Test.** The OWASP Vulnerability Test (OVT; Ivory et al. [33]) was used as a domain-specific measure of optimism bias for software engineers. This measure consists of two sections, and the first asks participants to estimate the likelihood of the *average developer* producing software containing one of the top five OWASP vulnerabilities (injection flaws, broken authentication, sensitive data exposure, XML external entity flaws, and broken access control). They are presented with five questions, one for each vulnerability, in a randomised order. Following a separation task, participants are asked about the

Manuscript submitted to ACM

Software Vulnerabilities as Cognitive Blindspots; assessing the suitability of a dual processing theory of decision making for secure coding

7

Table 2. The presentation order and attributes of the Python puzzles presented to participants. The puzzle ID relates to the original ID given in Brun et al. (2023) and cyclomatic scores are a measure of code complexity with higher scores indicating more difficult code to comprehend or modify

Puzzle	ID	Vulnerability Type	Blindspot Type	Cyclomatic Score
1	P02	Injection	Validation Missing	5
2	P09	Injection	Validation Missing	3
3	PX06	Injection		2
4	P21	Overflow	Function Misuse	2
5	P30	File	TOCTTOU	3
6	P36	File	Validation Missing	4
7	PX15	SSL		2
8	P31	File	Missing Verification	3

same vulnerabilities but to report the likelihood of *themselves* incorporating these vulnerabilities. This test measures comparative optimism, as previously used in other research [31].

**3.3.3 General Decision Making Style Scale.** The General Decision Making Style Scale (GDMS) measures five styles of decision making [67]: rational, avoidant, dependent, intuitive, and spontaneous. These subscales assess an individual’s decision making approach. The 25-question scale is presented in a randomised matrix with a five-point Likert response (strongly disagree – strongly agree). For the hypotheses, only the rational and intuitive subscales were of interest, as they are theoretically linked to dual processing theory, with the intuitive scale mapping against system 1 processing and higher levels of rational decision making mapping onto system 2.

**3.3.4 Puzzles.** The Python puzzles were developed by Brun et al. [10]. They consist of 10-21 lines of code with an accompanying scenario description providing context to the puzzle. An example of a scenario and puzzle is shown in Figure 1. Participants read the scenario and code and answered a free-text question about the behaviour once the code was executed. They are then asked a multiple-choice question of expected behaviour given specific inputs. Following this, they were asked to rate their confidence in solving the puzzle from 1-10, the percentage of others they would expect to solve the puzzle (from 1-100), perceived difficulty (from 1-10), familiarity with the functions (from 1-10), and scenario clarity (from 1-10). To aid the deception, they were asked whether any parts of the puzzle were confusing and how, whether there were unfamiliar functions, and to list these. They were asked what resources they used to review the code and, finally, to report the fatigue they experienced in completing the puzzle (from 1-10). Puzzles were scored as being correct or incorrect based on the multiple-choice question.

We presented participants with eight puzzles, six with blindspots and two without blindspots. Five of the six blindspot puzzles focused on input/output vulnerabilities, including networking activity and reading and writing to and from streams, files, and internal memory buffers. The sixth was a string manipulation vulnerability of user input. Puzzles were presented in two blocks, allowing participants to take a break in between. Each block had three blindspot puzzles and one non-blindspot puzzle and were balanced in terms of cyclomatic complexity. Table 2 shows the puzzle order, type of vulnerability and cyclomatic complexity. The complete set of puzzles used, their order, and associated questions can be found in the online repository here: <https://osf.io/2r4zx>.

```
01 import os
02
03 REPOSITORY_FOLDER = "/var/repository/"
04
05 def download(filename):
06     path = os.path.join(REPOSITORY_FOLDER, filename)
07
08     if os.path.exists(path):
09         return open(path).read()
10     else:
11         return None
```

What will the download function do when executed?

Which of the following is correct if users are allowed to enter any string value as filename?

The function fails only when the given string value as filename is invalid.

The function fails only when the file does not exist in the repository folder.

The function is able to read any (readable) file on the system.

The function is able to read only (readable) files in the repository folder.

None of the above

Fig. 1. An example of a puzzle containing a blindspot. All puzzles were formatted similarly, with the context given first, followed by an image of the code, and then asked to describe the code's behaviour. Each puzzle was prefaced with the scenario as context. The context would explain the general setup and use of the code section and provide examples, if necessary, of its use case. It also noted that readers should assume all necessary permissions are given for execution.

### 3.4 Procedure

Following participant approval participants were sent an Upwork contract and an attached information sheet. Participants were not told of the specific goal of examining blindspots in puzzles until data collection was complete. See Figure 2 for the study pipeline.

Participants were sent an online Qualtrics link and could only participate if they provided full informed consent. Participants completed the eight Python puzzles. They then completed demographic questions of age, Manuscript submitted to ACM

Software Vulnerabilities as Cognitive Blindspots; assessing the suitability of a dual processing theory of decision making for secure coding

9

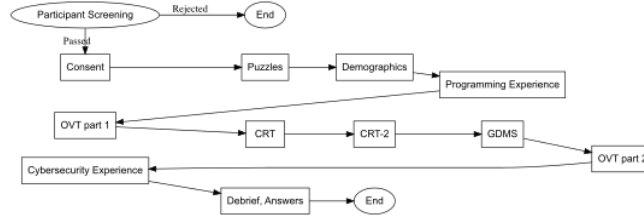


Fig. 2. Study pipeline that participants experienced. All participants experienced the same survey flow with randomisation in some sections.

gender, ethnicity, employment, and education. They were then asked to report general and Python-specific programming experience.

After this, the cognitive measures were presented, randomised within each measure, with the first OVT section, then the CRT measures, the GDMS questions, and finally, the second part of the OVT measure. Once participants completed the cognitive measures, they were asked questions on cybersecurity, reporting how much of their knowledge was self-taught or formally taught and the frequency with which they were required to employ it. They were then given a debrief explaining the study's true purpose, which was followed by the same puzzles with explanations of the vulnerabilities to ensure participants were aware of their potential blindspots. This concluded the study.

### 3.5 Analysis

Analysis was implemented through R 4.2.2, and data, analysis scripts, and instructions for reproducing the results can be found here: <https://osf.io/2r4zx>.

Two model types were used: mixed-effect ordinal logistic regression (MOLR) and general linear regression models. Where independent variables could vary across puzzles within participants (e.g., perceived confidence of solving different puzzles), MOLR models with a dependent variable of puzzle accuracy and random effect of puzzle were used to control for differences between participants and handle the nested data structure. Where independent variables did not vary, such as test scores, linear regression models were used with a dependent variable of total puzzle accuracy. The GDMS responses were subject to confirmatory factor analysis to compute single scores for the rational and intuitive subscales. These models are the same as those used by Brun et al. [10]. As with the analysis by Brun and colleagues, Bayes factors were used to assess models that had non-significant effects to determine whether the absence of an effect is due to data insensitivity (reflected in a value between .33 and 1) or a preference to accept the null hypothesis (a value under .33).

Following this, we conducted power analyses by simulating new data to understand the minimum number of participants for future confirmatory work and further explore the cognitive measures that are significant predictors with larger samples.

Statistical power is the probability of rejecting the null hypothesis in favour of an alternative hypothesis where a true effect does exist [39]. In lay terms, statistical power asks, "If I run my experiment 100 times, how many times will I find a significant result?". Power is a valuable tool for ensuring that research provides meaningful results. A standard threshold for suitable power in experimental psychology is 80% [11], and it

is modulated by the number of observations within the dataset used. As a result, experimental research in software engineering must ensure that adequate samples are used. When using mixed-effect regression models, data simulation is one method for determining statistical power for varying participant levels [39].

In our research, we ask an as-yet unanswered question: What are the plausible effect sizes and required sample sizes to address psychological questions of software engineering? To do so, we frame the current experiment as *groundwork*, providing the conceptual foundations for dual processing theory in software engineering and highlighting the role of statistical power for meaningful research. Collecting a modest sample provides the data and modelling opportunity to estimate the required scale of empirical work to address psychological questions in software engineering. The motivation for answering this question stems from the recognition that in psychology, many studies are not sufficiently powered, which has led to many findings not replicating [43].

The data collected was used to simulate 740 unique and new observations. These were used to explore the required participant numbers for suitable power. Original data were transformed by applying random noise to independent variables to simulate new data. Noise was sampled from a range of one standard deviation and randomly added or subtracted from the existing values. Values higher or lower than the possible variable limits were set to the maximum and minimum values, respectively. Simulated data reflected the general patterns observed by transforming independent variables and not dependent variables (i.e., accuracy) in this way. This process was performed 20 times to create a novel simulated dataset with 740 observations.

To calculate statistical power, 1,000 iterations of each model tested the hypotheses using a random subset of the 740 observations for each iteration. Iterations were conducted for sample sizes from 10 to 740 participants. Model coefficients and  $p$  values were extracted for all model terms. From this, power was calculated for each sample size as the percentage of the 1,000 models that provided significant terms at the .05 level.  $H_1$  was not included in the calculations because 37 participants provided ample power per the preregistration.

Power calculations were conducted simultaneously using a High-End Computing Cluster due to their computationally intensive nature. The calculations were performed using R 3.6.0, and scripts, data, and associated files for reproducing these calculations are found at <https://osf.io/2r4zx/>.

The two methods of assessing data suitability and their effects, Bayes factors and data simulation, are used to provide more information on the measures used and the value of dual processing theory. The power analysis is in addition to the preregistered plan, as it is often conducted in the planning stages of a research project. We include it here to motivate further research to ensure suitable sample sizes are used. It offers the opportunity to use a larger sample derived from the existing data and determine potential effects. The methods complement each other as Bayes factors rely upon the models created from the existing data to determine how well the models perform, and the simulated sample offers a dataset larger than is reasonable to collect in typical research.

## 4 RESULTS

The main hypotheses were tested using the data collected from the 37 participants, and then all the hypotheses bar  $H_1$  were subject to power analyses.  $H_1$  was predetermined to be suitably powered using the data provided by Brun et al. [10]. We report on the analysis of original data and power simulations in the same subsections.

Table 3. Percentage of correctly solved answers compared to the sample from Frederick (2005)

Score	Frederick (2005)	Present Study
0	33%	0%
1	28%	5.41%
2	23%	16.21%
3	17%	78.38%

#### 4.1 H<sub>1</sub>: The Effect of Blindspot on Accuracy

To address the hypothesis that Python puzzles with an API blindspot will be more challenging to solve than those without, a MOLR model was used, regressing the presence of blindspots against accuracy with a random effect of participant. The data did not support the model, so the random effect was removed, resulting in the simpler model, which was supported by the Bayesian Inference Criterion (BIC). The revised model had a BIC value of 368.12 compared to the original 373.81.

The revised model was found to be significant, with the blindspot presence having a coefficient  $\beta$  of -1.26,  $p < .001$ , giving an odds-ratio of .28, indicating that if a puzzle possesses a blindspot, the odds a participant solves the puzzle incorrectly is 3.53 times more likely than solving it correctly. Participants in the present study were more than twice as likely to solve puzzles without blindspots than reported in Brun et al. [10].

#### 4.2 H<sub>2</sub>: The Effect of Cognitive Reflection

Comparing the CRT scores against the original findings by Frederick [23] in Table 3, our responses are very different to the original findings, with a large majority successfully answering all questions. It is unclear why participants scored so highly on measures that do not typically elicit high performance, but it may be due to the use of generative AI language models providing answers. Testing this potential cause with Chat-GPT found that six of the seven questions were correctly solved. As such, our findings should be treated cautiously as they may not reflect the intended measure.

To address H<sub>2</sub>, that higher levels of cognitive reflection will associate with solving scenarios with blindspots more effectively, a linear regression using both CRT and CRT-2 was deployed. Model examination suggested removing the CRT term, leaving only CRT-2 as a predictor. BIC values indicated that the model using only CRT-2 was preferable. The model using solely CRT-2 was non-significant,  $p = .077$ . The Bayes factor for the model using both CRT variables compared to the null is .21, indicating that null hypothesis acceptance is more likely than data insensitivity, but for the model with just CRT-2, the Bayes factor was .93, suggesting that data insensitivity is far more likely than the null hypothesis.

#### 4.3 H<sub>3</sub>: The Effect of Optimism and Confidence

To test the effect seen in Ivory et al. [33], that engineers are likely to see themselves as less likely to be susceptible to including vulnerabilities in their work, the mean average of the OVT task was 125.84 (SD = 17.88), with a minimum score of 95 and a maximum of 155. A score of 100 would indicate individuals scoring themselves as equally likely as the general engineering population, but the higher average score indicates that the optimistic self-belief is persistent across samples. A one-sample t-test confirms that the OVT scores significantly differ from an average score of 100,  $t(36) = 8.79$ ,  $p < .001$ , with a large effect size  $d = 1.441$ .

We hypothesised that developers who demonstrate realistic levels of optimism will solve scenarios with blindspots more effectively than developers with higher levels of optimism, acting as a measure of system 1 processing. MOLR for accuracy scores with OVT, self-confidence, confidence in others, and random effects yielded just a significant effect of self-confidence. This indicated a less complex model was appropriate using just self-confidence and random effects, as the difference between the two models was not significant,  $p = .792$ .

The model with only self-confidence was significant, with a coefficient of .21,  $p = .014$ , equivalent to an odds ratio increase of 1.23, meaning that the odds of correctly solving a puzzle increase by 1.23 times for each unit increase in confidence. This finding indicates that self-confidence is an indicator of correctly solving puzzles containing blindspots. Testing the non-significant terms in the model, a Bayes factor of .01 was reported indicating that the null hypothesis is more likely.

#### 4.4 H<sub>4</sub>: The Effect of Rational Decision Making

H<sub>4</sub> hypothesised that developers with higher GDMS rational scores would solve scenarios with blindspots more effectively. This was tested using linear regression, and the model was not significant,  $F(1, 32) = 1.61$ ,  $p = .214$ , indicating that rational decision making does not affect the detection of vulnerabilities. The Bayes factor was .39, indicating data insensitivity.

#### 4.5 H<sub>5</sub>: The Effect of the Intuitive Style of Decision Making

H<sub>5</sub> expected that lower GDMS intuitive scores would associate with solving puzzles with blindspots more accurately. Linear regression did not support this,  $F(1, 32) = .90$ ,  $p = .351$ . The Bayes factor was .27, indicating a preference towards the null.

#### 4.6 H<sub>6</sub>: The Effect of Puzzle Attributes

H<sub>6</sub> tested the non-directional statement that developers' perceived ratings of puzzle difficulty, effort, familiarity, and confidence would affect their ability to solve scenarios containing blindspots. The MOLR model was found to be non-significant with no significant terms. The Bayes factor reported was .002, indicating a strong preference for the null hypothesis, suggesting that perceived ratings did not influence puzzle-solving ability.

#### 4.7 H<sub>7</sub>: The Effect of Expertise and Experience

H<sub>7</sub> stated that experience and programming ability would not affect puzzle solving, based upon previous findings, and the hypothesis sought to replicate the finding by Brun et al. [10] and Oliveira et al. [51]. Linear regression with total puzzles solved as the dependent variable, with programming experience, Python experience, and general programming familiarity as predictors, was not significant,  $F(3,30) = 1.83$ ,  $p = .163$ . The Bayes factor was .09, suggesting that no effect was more likely than data insensitivity. This supports our hypothesis and confirms findings by Brun et al. [10].

#### 4.8 H<sub>8</sub>: The Effect of Security Knowledge

H<sub>8</sub> hypothesised that cybersecurity knowledge and experience would have no effect on blindspot detection. Linear regression using cybersecurity experience and level of formal cybersecurity knowledge as predictors

Manuscript submitted to ACM



Software Vulnerabilities as Cognitive Blindspots; assessing the suitability of a dual processing theory of decision making for secure coding 13

did not yield a significant effect,  $F(5,28) = .41$ ,  $p = .840$ , and a Bayes factor of  $< .01$  indicates a strong chance of no effect being present, supporting our hypothesis.

#### 4.9 Findings Summary

H<sub>1</sub> sought to replicate findings from Brun et al. [10] that API blindspots in code would be harder to solve than puzzles without, and we found a significant effect supporting this hypothesis. H<sub>2</sub> tested dual processing through cognitive reflection and found no significant difference for vulnerability detection, with contrasting results from the two measures, failing to support the hypothesis; CRT responses were seen to be inconsistent with expected responses, suggesting compromised data, but CRT-2 alone offered a Bayes factor of .93 indicating the non-significant difference was due to data insensitivity and not an absence of an effect. H<sub>3</sub> tested whether a measure of optimism bias could explain detection ability, but Bayes factors indicated that no effect likely exists, failing to support the hypothesis. H<sub>4</sub> and H<sub>5</sub> used measures of rational and intuitive decision making to further explore dual processing, finding no significant differences, likely due to data insensitivity. H<sub>6</sub> is a replication from Brun et al.; no difference was seen for puzzle attributes. H<sub>7</sub> and H<sub>8</sub> were confirmatory tests of the findings in Brun et al., and no significant effects were seen for technical or security expertise, supporting our hypotheses.

#### 4.10 Simulated Samples for Power Analysis

Power calculations were derived from model simulations. Table 4 shows the required sample sizes needed to achieve a statistical power of .80 and the expected effect (the average coefficient within the regression model) achieved with appropriate power. Measures where no effect was detected, even in large samples included: hypothesis 3, where the effect for OVT and confidence tends towards zero, as was also seen for perceived effort in hypothesis 6. For hypothesis 8, the sample size is predicted to be over twice as large as the simulated sample, assuming a linear relationship between sample size and power. For all other measures used, they achieved statistical power within 740 participants, with seven of the remaining ten measures achieving power in 203 participants or less.

H<sub>2</sub>: The large sample associated with the CRT suggests any effect is small and unlikely to have any significant impact, whereas CRT-2 is likely to be more effective as a measure, with a .37 increase in accuracy for every correct response to CRT-2.

H<sub>3</sub>: A unit increase in perceived self-confidence was associated with a coefficient of .16.

H<sub>4</sub> and H<sub>5</sub>: For the GDMS scale, a unit increase in rational thinking results in a .26 increase in overall accuracy, and a unit increase in intuitive thinking results in a .17 decrease in the total puzzles accurately solved.

H<sub>6</sub>: For perceived difficulty, a unit increase in rating was associated with a coefficient of -.04, resulting in a minor decrease in accuracy. For familiarity with functions, a .07 coefficient was reported, relating to an increase in accuracy, and for self-confidence in a model with the terms of difficulty and familiarity, a coefficient of .13 was reported.

H<sub>7</sub>: For Python experience, a small coefficient of .10 is seen, suggesting that for a unit increase in reported Python experience, overall accuracy increases by .10, and for technical proficiency, a -.12 decrease is seen in overall accuracy.

Table 4. Potential effect sizes and samples reported from the simulation and power analysis. \* = this duplicated measure is the result of it being included in a second model from hypothesis 3, \*\* = the values for cybersecurity are based upon linear regression predictions and assume a linear relationship between power, coefficient, and sample size.

Hypothesis	Measure	Coefficient Size	Required Sample
2	CRT	-0.18	698
2	CRT-2	0.37	93
3	OVT		
3	Confidence-others		
3	Confidence-self	0.16	63
4	Rational	0.26	179
5	Intuitive	-0.17	371
6	Difficulty	-0.04	694
6	Effort		
6	Familiarity	0.07	186
6	Confidence-self*	0.13	88
7	Experience	0.10	144
7	Technical Proficiency	-0.12	203
8	Cybersecurity Experience**	-0.05	1517

H<sub>8</sub>: For cybersecurity experience, an effect of -.05 was detected, indicating a minimal decrease in overall accuracy for each unit increase in experience.

## 5 DISCUSSION

We successfully replicated the finding that blindspot puzzles were challenging. This is true even for those that cause well-known vulnerabilities, such as code injection or missing input validation. The effect we obtained was statistically reliable and had a larger effect size than in the Brun et al. [10] paper from which the materials were derived. We also replicated that technical expertise is not a predictor of vulnerability detection. It also extended that study in two important respects - the potential mediating factors of psychological variables and the modelling of effect and sampling requirements for relevant research studies.

### 5.1 Theoretical Support

We support the hypothesis proposed by Cappos et al. [13] by reporting on measures of dual processing systems in decision making. The power analysis indicates that dual processing theory may be associated with vulnerability detection. We saw no association for optimism bias susceptibility. For non-cognitive measures, self-confidence in solving puzzles was associated with higher accuracy, as was familiarity with code functions. The self-perceived difficulty was negatively associated. In contrast to Brun et al. [10], we saw a small positive effect for Python experience and a small negative effect for general proficiency.

The successful replication of Brun et al. [10] and Oliveira et al. [51] is valuable, but it generates further unanswered questions, specifically “What is it about blindspots that render software engineers less capable of identifying them?”. From the cognitive science perspective, blindspots can be explained through decision making styles, and the implications that intuitive versus rational approaches influence the mental representation of the software problem. An intuitive approach (system 1 thinking) increases the likelihood that the coding problem is not fully represented in their mental model, thereby creating the blindspot

itself. More analytic and algorithmic thinking is expected to identify the puzzle issues and form a more complete mental model that allows them to work toward a solution. Encouraging developers to create mental representations of software security can help them to draw inferences around its success [34].

We find this explanatory framework very appealing and plausible. However, the evidence from the current data is clearly somewhat mixed. The prediction from this above argument is that participants with higher CRT scores would solve more blindspot puzzles (by virtue of deploying more system 2 thinking). The data did not support this unequivocally. However, this may be due, at least in part, to the assessment of CRT performance, which may have been confounded by AI use. Despite the rigorous review of potential participants, data quality suffered in the CRT responses. It is unclear why a test that typically experiences floor effects did not present any zero-score responses, with over 78% answering all questions correctly. Participants may have used external resources like Chat-GPT to provide answers. The survey flow did not state that participants could not use resources at this point in the survey. If this was the case and participants were not being tested on their reflective abilities, no genuine conclusions can be drawn. As a result, the CRT data may be contaminated and not provide valuable findings. Future work should ensure participants know they cannot use resources to answer questions designed to measure cognition.

This study's primary theoretical outcome is its support for the proposed paradigm of security vulnerabilities occupying blindspots in our cognition [13]. By applying the dual processing theory [21], we used system 1 and system 2 processing measures to explore the potential effects these have on detecting security vulnerabilities. When interpreting results via dual processing theory, it is of little surprise that factors of technical expertise or cybersecurity had little impact. One's tendency to engage system 2 processing is a separate cognitive process to declarative knowledge, and the cue required to suppress system 1 is not linked to general intelligence [75]. The finding that higher levels of rational decision making styles tend towards increased detection, and higher levels of intuitive decision making tend towards decreased detection aligns with dual processing theory, highlighting that developers who tend towards more rational, system 2 processing styles of decision making are more likely to spot vulnerabilities.

We used two methods for assessing the data beyond the statistical modelling of the collected data: Bayes factors, and data simulation. We used Bayes factors to explore the likelihood of null results being a result of data insensitivity or that no effect is present. We also used simulated data to identify effects collected from a larger sample. These methods offer a greater insight into the data, mitigate the limitation of using a small sample size, and provide an enhanced understanding of the reported non-significant results. Agreement between the two methods was found for the two measures that have an effect: CRT-2 and rational decision making. Intuitive decision making has a weak Bayes factor of .27, and the evidence points towards a potentially small effect existing. For measures of function familiarity and self-confidence, the sample sizes identified via the power analysis (186 for function familiarity and 88 for self-confidence) suggest these are also worth exploring in further research with larger samples.

The OVT was used to measure domain-specific optimism bias and to collect information on the likelihood or strength of biases used during system 1 processing. Whilst an effect for overly optimistic views was seen for the sample in general, it did not associate with detecting blindspots in any significant pattern. This may be because, despite being domain-specific for software engineers, it is not specific enough when considering Python code. It may also be that whilst it is a measure of optimism bias, it does not adequately capture system 1 processing.

Many of the effects we identify are relatively small, indicating that solely employing engineers based on our findings would not solve security issues. The effects are more subtle and are indicative of the complexity of software engineering. Reducing these complex behaviours into easy-to-capture aspects of cognition would not be appropriate for determining who is assigned security tasks. However, they are an appropriate step forward in understanding how to support software engineers in identifying vulnerabilities.

## 5.2 Implications

The primary outcome of this research is that it confirms that further work using this paradigm and dual processing theory would benefit the software engineering community. Linking the theory to the paradigm with preliminary results also provides opportunities to explore targeted interventions for improving system 2 engagement when interacting with software code. Prompting for security has been shown to have a positive effect [30], but these changes are often short-lasting [80].

API developers should consider their role in supporting software developers, particularly with functions that have security implications, such as cryptography or file handling. Documentation usability has long been criticised as driving developers to look for information outside of official sources [26], resulting in weakened software [3]. API developers should ensure documentation is clear, providing secure examples [59] and not treating developers as security experts [4], as well as noting when functions are not secure by default or where potential vulnerabilities may occur.

Our replication of the finding that programming experience, technical knowledge, or cybersecurity experience have no significant effect on blindspot detection can be used to help educate developers about the dangers of thinking that these factors will help them produce high-quality, secure code. By targeting these ideas and delivering the message that intuition is not a positive trait to possess during security-essential tasks, developers can be made aware of the need to engage in reflective and critical thinking styles during security phases.

Companies that can do so should employ or delegate security testing to specific teams or individuals whose primary role is security-focused. Typically, developers are seen to prioritise functionality over security when tasked with both [6], and an absence of clearly defined roles for security can lead to a diminished sense of responsibility for ensuring software is secure [32]. Identifying specific security roles can reduce the presence of conflicting tasks, allowing for enhanced security focus, which may increase reflective thinking around security. Not all software projects can employ additional staff as security testers, but even assigning protected time where security is the sole focus can help boost deliberative thinking about security.

The indication that decision making styles may affect blindspot detection (as seen through the data simulation and power analysis) has implications for those involved in writing or reviewing code. The notion that increased rationality and decreased decision making are associated with greater detection of blindspots suggests that interventions should account for these differences. The findings *do not* suggest that those who report lower rational decision making styles are ill-suited for security tasks, as dual processing theory highlights a difference in the cue strength that causes system 2 to intervene during the decision making process.

Research from domains where decision making is critical, such as medical and clinical diagnostics, has previously explored potential interventions for enhancing rationality. These interventions include cognitive forcing strategies [15], which promote the development of internal models through learning metacognitive

Software Vulnerabilities as Cognitive Blindspots; assessing the suitability of a dual processing theory of decision making for secure coding 17

processes, recognising biases and where they may occur, before applying mental or physical checks to ensure developers are accounting for these biases. For software engineering, an example of a cognitive forcing strategy would be to provide materials and workshops to teach engineers about decision making theories and heuristics and biases. Providing context for specific biases, such as confirmation bias during testing stages [66], and the importance of using metacognitive awareness of planning non-confirmatory tests to identify edge cases and vulnerabilities more easily can increase awareness over decision making approaches. Cognitive forcing strategies are naturally individual and account for individual differences in default decision making styles. Those typically more intuitive may favour physical checks such as checklists that encourage rational thinking. In contrast, more nuanced prompts may suit those predisposed to more rational styles.

Diverse perspectives may also be beneficial in engaging more reflective thinking styles. A catalyst for engaging system 2 processing can be peer communication [69], as it allows for greater exploration of potential viewpoints and reduces potential biases. Aligning with different social identities, such as those shared by software users, can enhance feelings of responsibility [32], which can also result in decisions being taken that account for others [38]. By acknowledging these views, developers may look at software code differently, allowing them to identify security vulnerabilities that would otherwise be missed.

### 5.3 Research Implications

The power simulations suggest cognitive reflection has a potential link to blindspot detection. This implication is less evident than expected from the data, as participants scored higher on both tests than previous research would suggest. The simulation found a negative association for CRT and a positive association for CRT-2. These findings are incongruent as they are intended to measure the same cognitive dimension. As a result, it is not easy to draw meaningful conclusions from the findings without being too selective as to the interpretation. If, as previously suggested, AI was used to answer the questions, then the data and the models offer no real insight. As such, we decline to provide any strong discussion over these findings.

Appropriately powered studies have been an issue in psychology research as a history of inadequately powered research has rendered many published results inaccurate [8], with only around 40% of results being reproducible [53]. This issue is not limited to psychology research but has been explored in security research, with a similar absence of well-powered studies being detected [54]. We emphasise the importance of conducting a priori power analyses and demonstrate methods for doing so (small samples followed by data simulation).

The study and data simulation exposes methods for software engineering research to ensure that empirical research is suitably powered to identify true effects. Previous research that applies psychologically motivated ideas towards security (such as cognition, working memory, or prompting) has used sample sizes between 40 and 138 participants. In the present findings, we report requiring samples between 93 and 179 to detect the larger psychological effects. From this, we encourage future work in this domain to use simulations or previous findings to determine the minimum sample sizes needed. Alternatively, refining the measures and experimental conditions to increase the effect sizes through controlling the environment can reduce the sample size required.

It is also recognised that the sharing of data by Brun et al. [10] is a positive contribution to the software engineering research community, as they provide validated materials, enabling further research to compare findings more directly [78]. Software research typically deploys non-standardised tasks that make comparisons

with other research more difficult. Sharing data and materials helps to standardise software security research, making findings more comparable across studies.

#### 5.4 Limitations

One of this study's main limitations is the sample size of 37 participants. With a modest and knowingly underpowered small sample, the results should be treated cautiously, and this is recognised and handled in two ways. The first is through the conservative discussion of the results and their implications, focusing more on the support the findings provide for the "vulnerabilities as blindspots" paradigm [13]. By deliberately keeping the discussion from translating findings to practical application at this stage, we recognise the limits of the research. The second is the inclusion of the simulation of 740 novel observations. With the simulated data, statistical power was calculated for future research, and combined with the provision of all research materials, this provides a clear path for confirmatory studies.

The sample was collected online using Upwork, an online freelancing platform criticised for potentially including inexperienced or low-quality participants [48]. So, to ensure data quality, we used multiple resources to verify participants' experience as programmers, and they were asked screening questions about this experience and prompted for further details if initial responses were not satisfactory. Their freelance profiles were also checked for previous work and experience; GitHub or other linked public sites were checked or requested if missing, along with any other resources available to check suitability. Finally, applicants suspected of responding using AI-generated content were declined, along with those who could only provide information limited to academic settings, as student participants were not used. As a result, only 29.37% of applicants were accepted.

#### 5.5 Future Work

The obvious and initial development would be to carry out a suitably powered confirmatory study. For most cognitive measures collected, a minimum sample size of 203 participants would be required to achieve a statistical power level of .8 (giving an 80% chance of detecting a true effect if it exists). A sample of around 100 would be sufficient to capture the stronger effects measured.

Other developments that warrant further exploration are to remove measures that likely hold no predictive value and explore other measures in their place, such as the Need for Cognition scale [12], a measure of an individual's tendency to engage in cognition during decision making.

Additional opportunities exist to test the effectiveness of psychological interventions for reducing software engineers' blindspots when producing or interacting with software code. Previous research has shown the benefit of short-term interventions such as nudging [49], but longer-term interventions should be explored or developed to promote the persistence of security behaviours, such as cognitive forcing.

### 6 RELATED WORK

This study builds upon a previous programme that posits software vulnerabilities are often missed by engineers as the vulnerabilities occupy cognitive blindspots [13]. Support for this idea came from Oliveira et al. [52], who tested the effect of priming engineers to review data for security. Following this, other studies explored cognitive aspects of blindspot detection using Java [51] and Python [10]. Both studies supported the paradigm, particularly for the underlying hypothesis that security vulnerabilities are difficult for engineers to

Manuscript submitted to ACM

Software Vulnerabilities as Cognitive Blindspots; assessing the suitability of a dual processing theory of decision making for secure coding 19

detect. Our research extends the paradigm by applying dual processing theory [21] and measures of system 1 and system 2 processing to the hypothesis.

The remainder of this section provides context for our research regarding cognition research in software engineering, API security, and the use of dual processing theory in other applied domains.

## 6.1 Cognition and Software Engineering

Within software security, an increased focus has been placed on the cognition and behaviours of those involved in software creation [24]. Security is perceived as a reactive, event-driven process [41] with little priority over functionality [40]. The reduced perceived importance of security and its temporal association with specific events (as opposed to procedural, proactive workflows) means that it is less likely to be at the forefront of an engineer’s mind during decision making, aligning with the dual processing theory.

Using prompts to encourage secure coding is of some benefit. Prompting can be the cue required to engage system 2 processing and override or suppress system 1. Simple interventions of requesting security have demonstrated their power in improving security with a broad range of populations, including students [50], freelancers [49], and professional developers [48]. Prompting can also work more subtly, such as asking engineers to write design specifications before writing secure code [30].

The cognitive load demanded by software development is significant and diverse, with requirements including functionality, performance, usability, and security, amongst others [63]. Cognitive load research in secure software development is limited, but manually conducted reviews and tests would likely overload cognitive capacities, resulting in increased presence of software vulnerabilities [19].

Biases within software security have been explored previously and aid in explaining many common issues [44]. One commonly reported bias is optimism bias, which is found in many forms, such as underestimating time requirements for projects [45], downplaying security threats [42], or reduced concern over software security [33]. Other biases include confirmation bias, where individuals seek information that confirms internal hypotheses rather than disprove them [73]. Confirmation bias is seen as an issue in software testing [66], as engineers may be less motivated or capable of applying non-confirmatory tests on edge cases, where vulnerabilities often lie. Biases result from system 1 processing, and their noted prevalence in software engineering research highlights just how common this processing is.

Research into cognition and psychology surrounding software engineering has explored ideas relevant to understanding security behaviours. We contribute to this existing research by applying a theory of decision making that aligns with work by others [10, 13, 51], highlighting its potential to explain behaviours observed surrounding software vulnerabilities.

## 6.2 API Usability and Security

API usability research explores how their design affects a user’s performance and how it impacts software security [46]. API misuse is a significant issue within software development, with nearly 90% of Android applications possessing at least one cryptographic API fault [18]. This section highlights three issues found in usability research about security.

One of the core issues identified as reducing API security usability is an absence of documentation. This absence affects an engineer’s awareness of the security of API functions. It has been highlighted that where documentation is difficult to read or too low-level in detail, engineers must read a lot of documentation to

understand basic principles and ensure their API use is secure [47]. Others also found that the absence of easy-to-use documentation resulted in functionally correct but insecure software [27]. While simpler APIs are more secure than complex APIs, good documentation can moderate API complexity [1]. API documentation should be written in the most precise and simplest terms possible, making default behaviour (whether secure or insecure) explicit to ensure API users are not required to become experts before use [58]. Despite this, API developers assume user expertise [14], which is at odds with the finding that many developers use APIs for functionality over security [29], and APIs that do not reflect this behaviour, perhaps by requiring users to specify security settings, are more likely to be used insecurely. In the opposite direction, developers expect APIs to be secure by default [57], reflecting their lack of expertise in the API and their desire to get their software created with minimal effort.

The API creator's intent affects how it is expected to be used [64], and misunderstandings can result in the API being used incorrectly. How an API is designed makes a difference in how it can be used, and the absence of high-level information makes the intentions harder to grasp. Usability can be reflected in the examples provided for function use in documentation. Where a mismatch between an example and a user's desire is seen, this creates friction during the learning and usage process. This lack of actionable or easily-used examples can cause issues for engineers, who seek answers elsewhere instead of relying upon official documentation, such as online forums, including StackOverflow [79]. Security solutions from online forums can be functionally correct yet insecure, particularly compared to official documentation, which is more secure but typically more challenging [2].

Our research explored how cognition and dual processing theory affect the ability to comprehend API use within existing code. The research above highlights the issues developers face, with the clear and actionable goal of producing simpler documentation, as it reduces extrinsic cognitive load and offers clear pathways for engineers. We complement this by highlighting how individual differences amongst software engineers can affect the ability to use this information and detect blindspots.

### 6.3 The Application of Dual Processing Theory

We present the first study into secure software development that explicitly uses dual processing theory to interpret the results. Within software engineering, the theory's potential has been discussed [see 60, 61, 65], but it has not been empirically tested.

Previous research has applied the theory to other domains where decision making is critical, such as in medical applications [17], suggesting that symptoms may be overtreated in situations of uncertainty due to a reliance on system 1 processing. The overtreatment may result from clinicians wishing to reduce potential negative emotions from a decision. In nursing, dual processing theory affects decisions in clinical environments [56].

Dual processing theory has been applied in academic contexts with students exploring experimental models [72]. When evaluating incorrect models, presenting students with additional information can provide a sufficient cuing mechanism for engaging system 2. Financial literacy is also moderated by intuitive versus rational decision making, where intuitive, heuristic-based decisions reduce a person's financial performance in stock market simulations [25]. Dual processing theory applies to various domains where decision making is critical, and it stands to reason that software security is no different. Our findings lend support to this position.



## 7 CONCLUSIONS

We report on a study that assessed the viability of using the dual processing theory of decision making [21] to explain why software vulnerabilities are often missed during code review. This research advances the paradigm that security vulnerabilities often exist as cognitive blindspots [13]. System 1 and 2 processing measures were used alongside Python puzzles containing blindspots.

Our findings suggest cognitive reflection and rational decision making are linked to better performance, whereas intuitive decision making is negatively associated. We support previous findings that technical experience and expertise do not affect blindspot detection. The results are discussed for the support they offer the paradigm and potential ways the findings could be utilised in practical settings once validated with confirmatory research. This study provides the foundations for further work in providing software engineers with psychology-based interventions that are not restricted to programming languages, environments, or IDEs but are grounded in their cognitive competencies.

## REFERENCES

- [1] Yasemin Acar, M. Backes, S. Fahl, S. Garfinkel, D. Kim, M. L. Mazurek, and C. Stransky. 2017. Comparing the Usability of Cryptographic APIs. In *2017 IEEE Symposium on Security and Privacy (SP)*. 154–171. <https://doi.org/10.1109/SP.2017.52>
- [2] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. 2016. You Get Where You're Looking for: The Impact of Information Sources on Code Security. In *2016 IEEE Symposium on Security and Privacy (SP)*. 289–305. <https://doi.org/10.1109/sp.2016.25>
- [3] Yasemin Acar, Christian Stransky, Dominik Wermke, Charles Weir, Michelle L. Mazurek, and Sascha Fahl. 2017. Developers Need Support, Too: A Survey of Security Advice for Software Developers. In *2017 IEEE Cybersecurity Development (SecDev)*. IEEE, Cambridge, MA, USA, 22–26. <https://doi.org/10.1109/SecDev.2017.17>
- [4] Md Rayhan Amin and Tanmay Bhowmik. 2021. Information on Potential Vulnerabilities for New Requirements: Does It Help Writing Secure Code?. In *2021 IEEE 29th International Requirements Engineering Conference (RE)*. 408–413. <https://doi.org/10.1109/RE51729.2021.00046>
- [5] Vaibhav Anu, Kazi Zakia Sultana, and Bharath K. Samanthula. 2020. A Human Error Based Approach to Understanding Programmer-Induced Software Vulnerabilities. In *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. 49–54. <https://doi.org/10.1109/ISSREW51248.2020.00036>
- [6] Hala Assal and Sonia Chiasson. 2019. 'Think Secure from the Beginning'; A Survey with Software Developers. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems - CHI '19*. 1–13. <https://doi.org/10.1145/3290605.3300519>
- [7] Alan D. Baddeley. 1986. *Working Memory*. Clarendon Press/Oxford University Press, New York, NY, US. xi, 289 pages.
- [8] Monya Baker. 2016. 1,500 Scientists Lift the Lid on Reproducibility. *Nature* 533, 7604 (May 2016), 452–454. <https://doi.org/10.1038/533452a>
- [9] Denise R. Beike and Steven J. Sherman. 1994. Social Inference; Inductions, Deductions, and Analogies. In *Handbook of Social Cognition* (2nd ed. ed.), Robert S. Wyer and Thomas K. Srull (Eds.). L. Erlbaum Associates, Hillsdale, N.J.
- [10] Yuriy Brun, Tian Lin, Jessie Elise Somerville, Elisha M. Myers, and Natalie C. Ebner. 2023. Blindspots in Python and Java APIs Result in Vulnerable Code. *ACM Transactions on Software Engineering and Methodology* (April 2023). <https://doi.org/10.1145/3571850>
- [11] Marc Brysbaert and Michaël Stevens. 2018. Power Analysis and Effect Size in Mixed Effects Models: A Tutorial. 1, 1 (Jan. 2018), 9. <https://doi.org/10.5334/joc.10>
- [12] John T. Cacioppo and Richard E. Petty. 1982. The Need for Cognition. *Journal of Personality and Social Psychology* 42, 1 (1982), 116–131. <https://doi.org/10.1037/0022-3514.42.1.116>
- [13] Justin Cappos, Yanyan Zhuang, Daniela Seabra Oliveira, Marissa Rosenthal, and Kuo-Chuan Yeh. 2014. Vulnerabilities as Blind Spots in Developer's Heuristic-Based Decision-Making Processes. In *Proceedings of the 2014 Workshop on New Security Paradigms Workshop - NSPW '14*. ACM Press, Victoria, British Columbia, Canada, 53–62. <https://doi.org/10.1145/2683467.2683472>

- [14] Partha Das Chowdhury, Joseph Hallett, Nikhil Patnaik, Mohammad Tahaei, and Awais Rashid. 2021. Developers Are Neither Enemies Nor Users: They Are Collaborators. In *2021 IEEE Secure Development Conference (SecDev)*. 47–55. <https://doi.org/10.1109/SecDev51306.2021.00023>
- [15] Pat Croskerry. 2003. Cognitive Forcing Strategies in Clinical Decisionmaking. *Annals of Emergency Medicine* 41, 1 (Jan. 2003), 110–120. <https://doi.org/10.1067/mem.2003.22>
- [16] Kaja Damjanović, Vera Novković, Irena Pavlović, Sandra Ilić, and Slobodan Pantelić. 2019. A Cue for Rational Reasoning: Introducing a Reference Point in Cognitive Reflection Tasks. *Europe's Journal of Psychology* 15, 1 (Feb. 2019), 25–40. <https://doi.org/10.5964/ejop.v15i1.1701>
- [17] Benjamin Djulbegovic, Iztok Hozo, Jason Beckstead, Athanasios Tsalatsanis, and Stephen G. Pauker. 2012. Dual Processing Model of Medical Decision-Making. *BMC Medical Informatics and Decision Making* 12, 1 (Sept. 2012), 94. <https://doi.org/10.1186/1472-6947-12-94>
- [18] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. 2013. An Empirical Study of Cryptographic Misuse in Android Applications. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security - CCS '13*. ACM Press, Berlin, Germany, 73–84. <https://doi.org/10.1145/2508859.2516693>
- [19] Sarah Elder, Nusrat Zahan, Rui Shu, Monica Metro, Valeri Kozarev, Tim Menzies, and Laurie Williams. 2022. Do I Really Need All This Work to Find Vulnerabilities? *Empirical Software Engineering* 27, 6 (Aug. 2022), 154. <https://doi.org/10.1007/s10664-022-10179-6>
- [20] Jonathan St. B. T. Evans. 1984. Heuristic and Analytic Processes in Reasoning. *British Journal of Psychology* 75, 4 (1984), 451–468. <https://doi.org/10.1111/j.2044-8295.1984.tb01915.x>
- [21] Jonathan St. B. T. Evans. 2003. In Two Minds: Dual-Process Accounts of Reasoning. *Trends in Cognitive Sciences* 7, 10 (Oct. 2003), 454–459. <https://doi.org/10.1016/j.tics.2003.08.012>
- [22] Jonathan St. B. T. Evans and Keith E. Stanovich. 2013. Dual-Process Theories of Higher Cognition: Advancing the Debate. *Perspectives on Psychological Science* 8, 3 (May 2013), 223–241. <https://doi.org/10.1177/1745691612460685>
- [23] Shane Frederick. 2005. Cognitive Reflection and Decision Making. *Journal of Economic perspectives* 19, 4 (2005), 25–42. <https://doi.org/10.1257/089533005775196732>
- [24] Steven Furnell. 2021. The Cybersecurity Workforce and Skills. *Computers & Security* 100 (Jan. 2021), 102080. <https://doi.org/10.1016/j.cose.2020.102080>
- [25] Markus Glaser and Torsten Walther. 2014. Run, Walk, or Buy? Financial Literacy, Dual-Process Theory, and Investment Behavior. <https://doi.org/10.2139/ssrn.2167270>
- [26] Peter Leo Gorski, Yasemin Acar, Luigi Lo Iacono, and Sascha Fahl. 2020. Listen to Developers! A Participatory Design Study on Security Warnings for Cryptographic APIs. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–13. <https://doi.org/10.1145/3313831.3376142>
- [27] Peter Leo Gorski, Luigi Lo Iacono, Dominik Wermke, Christian Stransky, Sebastian Möller, Yasemin Acar, and Sascha Fahl. 2018. Developers Deserve Security Warnings, Too: On the Effect of Integrated Security Advice on Cryptographic API Misuse. *Fourteenth Symposium on Usable Privacy and Security* (2018), 265–281.
- [28] Margaret Gratian, Sruthi Bandi, Michel Cukier, Josiah Dykstra, and Amy Ginther. 2018. Correlating Human Traits and Cyber Security Behavior Intentions. *Computers & Security* 73 (March 2018), 345–358. <https://doi.org/10.1016/j.cose.2017.11.015>
- [29] Matthew Green and Matthew Smith. 2016. Developers Are Not the Enemy!: The Need for Usable Security APIs. *IEEE Security Privacy* 14, 5 (Sept. 2016), 40–46. <https://doi.org/10.1109/MSP.2016.111>
- [30] Joseph Hallett, Nikhil Patnaik, Benjamin Shreeve, and Awais Rashid. 2021. “Do This! Do That!, And Nothing Will Happen” Do Specifications Lead to Securely Stored Passwords?. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 486–498. <https://doi.org/10.1109/ICSE43902.2021.00053>
- [31] Jerome D. Hoover and Alice F. Healy. 2019. The Bat-and-Ball Problem: Stronger Evidence in Support of a Conscious Error Process. *Decision* 6 (2019), 369–380. <https://doi.org/10.1037/dec0000107>
- [32] Matthew Ivory, Miriam Sturdee, John Towse, Mark Levine, and Bashar Nuseibeh. 2023 (in review). Can You Hear the ROAR of Software Security? How Responsibility, Optimism And Risk Shape Developers' Security Perceptions. *Empirical Software Engineering* (2023 (in review)). <https://doi.org/10.31234/osf.io/pexvz>
- [33] Matthew Ivory, John Towse, Miriam Sturdee, Mark Levine, and Bashar Nuseibeh. 2023. Recognizing the Known Unknowns; the Interaction Between Reflective Thinking and Optimism for Uncertainty Among Software Developer's Security Perceptions. *Technology, Mind, and Behavior* 4, 3: Winter 2023 (Dec. 2023). <https://doi.org/10.1037/tmb0000122>
- [34] Philip Nicholas Johnson-Laird. 1983. *Mental Models: Towards a Cognitive Science of Language, Inference, and Consciousness*. Harvard University Press.
- [35] Philip N. Johnson-Laird. 2010. Mental Models and Human Reasoning. *Proceedings of the National Academy of Sciences* 107, 43 (Oct. 2010), 18243–18250. <https://doi.org/10.1073/pnas.1012933107>

- [36] Daniel Kahneman and Shane Frederick. 2002. Representativeness Revisited: Attribute Substitution in Intuitive Judgment. In *Heuristics and Biases* (1 ed.), Thomas Gilovich, Dale Griffin, and Daniel Kahneman (Eds.). Cambridge University Press, 49–81. <https://doi.org/10.1017/CBO9780511808098.004>
- [37] Daniel Kahneman, Paul Slovic, and Amos Tversky (Eds.). 1974. *Judgment under Uncertainty: Heuristics and Biases* (1st ed.). Cambridge University Press, Cambridge, United Kingdom.
- [38] Roderick M. Kramer, Pamela Pommerenke, and Elizabeth Newton. 1993. The Social Context of Negotiation: Effects of Social Identity and Interpersonal Accountability on Negotiator Decision Making. *Journal of Conflict Resolution* 37, 4 (Dec. 1993), 633–654. <https://doi.org/10.1177/0022002793037004003>
- [39] Levi Kumle, Melissa L.-H. Võ, and Dejan Draschkow. 2021. Estimating Power in (Generalized) Linear Mixed Models: An Open Introduction and Tutorial in R. *Behavior Research Methods* 53, 6 (Dec. 2021), 2528–2543. <https://doi.org/10.3758/s13428-021-01546-0>
- [40] Tamara Lopez, Helen Sharp, Thein Tun, Arosha Bandara, Mark Levine, and Bashar Nuseibeh. 2019. Talking About Security with Professional Developers. In *2019 IEEE/ACM Joint 7th International Workshop on Conducting Empirical Studies in Industry (CESI) and 6th International Workshop on Software Engineering Research and Industrial Practice (SER IP)*. 34–40. <https://doi.org/10.1109/CESSER-IP.2019.00014>
- [41] Tamara Lopez, Helen Sharp, Thein Tun, Arosha K. Bandara, Mark Levine, and Bashar Nuseibeh. 2019. "Hopefully We Are Mostly Secure": Views on Secure Code in Professional Practice. In *12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, Montreal, QB, Canada, 61–68. <https://doi.org/10.1109/chase.2019.00023>
- [42] André Loske, Thomas Widjaja, and Peter Buxmann. 2013. Cloud Computing Providers' Unrealistic Optimism Regarding IT Security Risks: A Threat to Users?. In *ICIS 2013 Proceedings*. ICIS, Milano, Italy.
- [43] Scott E. Maxwell, Michael Y. Lau, and George S. Howard. 2015. Is Psychology Suffering from a Replication Crisis? What Does "Failure to Replicate" Really Mean? *American Psychologist* 70, 6 (2015), 487–498. <https://doi.org/10.1037/a0039400>
- [44] Rahul Mohanani, Iftaah Salman, Burak Turhan, Pilar Rodríguez, and Paul Ralph. 2020. Cognitive Biases in Software Engineering: A Systematic Mapping Study. *IEEE Transactions on Software Engineering* 46, 12 (Dec. 2020), 1318–1339. <https://doi.org/10.1109/TSE.2018.2877759>
- [45] Kjetil Molokken and Magne Jørgensen. 2005. Expert Estimation of Web-Development Projects: Are Software Professionals in Technical Roles More Optimistic Than Those in Non-Technical Roles? *Empirical Software Engineering* 10, 1 (Jan. 2005), 7–30. <https://doi.org/10.1023/B:EMSE.0000048321.46871.2e>
- [46] Brad A. Myers and Jeffrey Stylos. 2016. Improving API Usability. *Commun. ACM* 59, 6 (May 2016), 62–69. <https://doi.org/10.1145/2896587>
- [47] Sarah Nadi, Stefan Krüger, Mira Mezini, and Eric Bodden. 2016. Jumping through Hoops: Why Do Java Developers Struggle with Cryptography APIs?. In *Proceedings of the 38th International Conference on Software Engineering (ICSE '16)*. Association for Computing Machinery, New York, NY, USA, 935–946. <https://doi.org/10.1145/2884781.2884790>
- [48] Alena Naiakshina, Anastasia Danilova, Eva Gerlitz, and Matthew Smith. 2020. On Conducting Security Developer Studies with CS Students: Examining a Password-Storage Study with CS Students, Freelancers, and Company Developers. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–13. <https://doi.org/10.1145/3313831.3376791>
- [49] Alena Naiakshina, Anastasia Danilova, Eva Gerlitz, Emanuel von Zeischwitz, and Matthew Smith. 2019. "If You Want, I Can Store the Encrypted Password": A Password-Storage Field Study with Freelance Developers. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–12. <https://doi.org/10.1145/3290605.3300370>
- [50] Alena Naiakshina, Anastasia Danilova, Christian Tiefenau, Marco Herzog, Sergej Dechand, and Matthew Smith. 2017. Why Do Developers Get Password Storage Wrong? A Qualitative Usability Study. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. Association for Computing Machinery, New York, NY, USA, 311–328. <https://doi.org/10.1145/3133956.3134082>
- [51] Daniela Seabra Oliveira, Tian Lin, Muhammad Sajidur Rahman, Rad Akefirad, Donovan Ellis, Eliany Perez, Rahul Bobhate, Lois A. DeLong, Justin Cappos, and Yuriy Brun. 2018. {API} Blindspots: Why Experienced Developers Write Vulnerable Code. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*. 315–328.
- [52] Daniela Seabra Oliveira, Marissa Rosenthal, Nicole Morin, Kuo-Chuan Yeh, Justin Cappos, and Yanyan Zhuang. 2014. It's the Psychology Stupid. In *Proceedings of the 30th Annual Computer Security Applications Conference*. Association for Computing Machinery, New Orleans, LA, USA, 296–305. <https://doi.org/10.1145/2664243.2664254>
- [53] Open Science Collaboration. 2015. Estimating the Reproducibility of Psychological Science. *Science* 349, 6251 (Aug. 2015), aac4716. <https://doi.org/10.1126/science.aac4716>

- [54] Anna-Marie Orloff, Christian Tiefenau, and Matthew Smith. 2023. {SoK}: I Have the (Developer) Power! Sample Size Estimation for Fisher's Exact, {Chi-Squared}, {McNemar's}, Wilcoxon {Rank-Sum}, Wilcoxon {Signed-Rank} and t-Tests in {Developer-Centered} Usable Security. In *Nineteenth Symposium on Usable Privacy and Security (SOUPS 2023)*. 341–359.
- [55] Alexa Palassis, Craig P. Speelman, and Julie Ann Pooley. 2021. An Exploration of the Psychological Impact of Hacking Victimization. *SAGE Open* 11, 4 (Oct. 2021), 21582440211061556. <https://doi.org/10.1177/21582440211061556>
- [56] John Paley, Helen Cheyne, Len Dalglish, Edward A. S. Duncan, and Catherine A. Niven. 2007. Nursing's Ways of Knowing and Dual Process Theories of Cognition. *Journal of Advanced Nursing* 60, 6 (2007), 692–701. <https://doi.org/10.1111/j.1365-2648.2007.04478.x>
- [57] Hernan Palombo, Armin Ziaie Tabari, Daniel Lende, Jay Ligatti, and Xinming Ou. 2020. An Ethnographic Understanding of Software ({In}Security) and a {Co-Creation} Model to Improve Secure Software Development. In *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*. 205–220.
- [58] Nikhil Patnaik, Andrew C. Dwyer, Joseph Hallett, and Awais Rashid. 2021. Don't Forget Your Classics: Systematizing 45 Years of Ancestry for Security API Usability Recommendations. *arXiv:cs/2105.02031*
- [59] Nikhil Patnaik, Andrew C Dwyer, Joseph Hallett, and Awais Rashid. 2022. SLR: From Saltzer & Schroeder to 2021...: 47 Years of Research on the Development and Validation of Security API Recommendations. *ACM Transactions on Software Engineering and Methodology* (Sept. 2022). <https://doi.org/10.1145/3561383>
- [60] Marian Petre. 2022. Exploring Cognitive Bias 'in the Wild': Technical Perspective. *Commun. ACM* 65, 4 (April 2022), 114–114. <https://doi.org/10.1145/3517215>
- [61] Carianne Pretorius, Maryam Razavian, Katrin Eling, and Fred Langerak. 2018. Towards a Dual Processing Perspective of Software Architecture Decision Making. In *2018 IEEE International Conference on Software Architecture Companion (ICSA-C)*. 48–51. <https://doi.org/10.1109/ICSA-C.2018.00021>
- [62] Paul Ralph. 2013. Possible Core Theories for Software Engineering. In *2013 2nd SEMAT Workshop on a General Theory of Software Engineering (GTSE)*. 35–38. <https://doi.org/10.1109/GTSE.2013.6613868>
- [63] Irum Rauf, Marian Petre, Thein Tun, Tamara Lopez, Paul Lunn, Dirk Van Der Linden, John Towse, Helen Sharp, Mark Levine, Awais Rashid, and Bashar Nuseibeh. 2021. The Case for Adaptive Security Interventions. *ACM Transactions on Software Engineering and Methodology* 31, 1 (Sept. 2021), 9:1–9:52. <https://doi.org/10.1145/3471930>
- [64] M. P. Robillard. 2009. What Makes APIs Hard to Learn? Answers from Developers. *IEEE Software* 26, 6 (Nov. 2009), 27–34. <https://doi.org/10.1109/MS.2009.193>
- [65] Anthony V. Robins. 2022. Dual Process Theories: Computing Cognition in Context. *ACM Transactions on Computing Education* 22, 4 (Sept. 2022), 41:1–41:31. <https://doi.org/10.1145/3487055>
- [66] Iflaah Salman, Burak Turhan, and Sira Vegas. 2019. A Controlled Experiment on Time Pressure and Confirmation Bias in Functional Software Testing. *Empirical Software Engineering* 24, 4 (Aug. 2019), 1727–1761. <https://doi.org/10.1007/s10664-018-9668-8>
- [67] Susanne G. Scott and Reginald A. Bruce. 1995. Decision-Making Style: The Development and Assessment of a New Measure. *Educational and Psychological Measurement* 55, 5 (Oct. 1995), 818–831. <https://doi.org/10.1177/0013164495055005017>
- [68] Agnia Sergeyuk, Sergey Titov, Yaroslav Golubev, and Timofey Bryksin. 2023. Overcoming the Mental Set Effect in Programming Problem Solving. (2023).
- [69] Ben Shreeve, Catarina Gralha, Awais Rashid, João Araujo, and Miguel Goulão. 2022. Making Sense of the Unknown: How Managers Make Cyber Security Decisions. *ACM Transactions on Software Engineering and Methodology* (Aug. 2022). <https://doi.org/10.1145/3548682>
- [70] H. A. Simon. 1956. Rational Choice and the Structure of the Environment. *Psychological Review* 63, 2 (1956), 129–138. <https://doi.org/10.1037/h0042769>
- [71] William Smart. 2018. *Lessons Learned Review of the WannaCry Ransomware Cyber Attack*. Technical Report. National Health Service, London, United Kingdom. 1–42 pages.
- [72] J. Caleb Speirs, MacKenzie R. Stetzer, Beth A. Lindsey, and Mila Kryjevskaja. 2021. Exploring and Supporting Student Reasoning in Physics by Leveraging Dual-Process Theories of Reasoning and Decision Making. *Physical Review Physics Education Research* 17, 2 (Nov. 2021), 020137. <https://doi.org/10.1103/PhysRevPhysEducRes.17.020137>
- [73] Webb Stacy and Jean MacMillan. 1995. Cognitive Bias in Software Engineering. *Commun. ACM* 38, 6 (1995), 57–63. <https://doi.org/10.1145/203241.203256>
- [74] Michael Stagnaro, Gordon Pennycook, and David G Rand. 2018. Performance on the Cognitive Reflection Test Is Stable across Time. *Stagnaro, MN, Pennycook, G., & Rand, DG (2018) Performance on the Cognitive Reflection Test is stable across time. Judgment and Decision Making* 13 (2018), 260–267.
- [75] Keith E. Stanovich and Richard F. West. 2014. The Assessment of Rational Thinking: IQ  $\neq$  RQ. *Teaching of Psychology* 41, 3 (July 2014), 265–271. <https://doi.org/10.1177/0098628314537988>

Software Vulnerabilities as Cognitive Blindspots; assessing the suitability of a dual processing theory of decision making for secure coding 25

- [76] Keela S Thomson and Daniel M Oppenheimer. 2016. Investigating an Alternate Form of the Cognitive Reflection Test. *Judgment and Decision making* 11, 1 (2016), 99.
- [77] Maggie E. Toplak, Richard F. West, and Keith E. Stanovich. 2011. The Cognitive Reflection Test as a Predictor of Performance on Heuristics-and-Biases Tasks. *Mem Cognit* 39, 7 (Oct. 2011), 1275–89. <https://doi.org/10.3758/s13421-011-0104-1> arXiv:21541821
- [78] Andrea Towse, David A. Ellis, and John Towse. 2021. Making Data Meaningful: Guidelines for Good Quality Open Data. *The Journal of Social Psychology* 161, 4 (July 2021), 395–402. <https://doi.org/10.1080/00224545.2021.1938811>
- [79] Dirk van der Linden, Emma Williams, Joseph Hallett, and Awais Rashid. 2020. The Impact of Surface Features on Choice of (in)Secure Answers by Stackoverflow Readers. *IEEE Transactions on Software Engineering* (2020), 1–1. <https://doi.org/10.1109/tse.2020.2981317>
- [80] Merije Van Rookhuijzen, Emely De Vet, and Marieke A. Adriaanse. 2021. The Effects of Nudges: One-Shot Only? Exploring the Temporal Spillover Effects of a Default Nudge. *Frontiers in Psychology* 12 (2021).

## 8.1 Statement of Continuous Thesis Summary

*“We can be blind to the obvious, and blind to our own blindness” - Daniel Kahneman, 2011*

This chapter applies measures of dual processing towards vulnerability detection in Python code to explore the suitability of the theory for explaining the variance observed in differences in vulnerability detection. This chapter tests the hypothesis that software vulnerabilities occupy blindspots in our cognition (Cappos et al., 2014) and extends it by applying measures of decision making to assess whether intuitive and rational thinking styles alter detection success. I replicate previous findings from Brun et al. (2023) that code containing blindspots is more challenging to solve, and I then assess the suitability of cognitive measures for explaining variance in vulnerability detection.

In the modest sample of 37 participants, I confirmed previous findings that blindspots are difficult to detect, and that technical nor security expertise significantly improve vulnerability detection. No significant effects in this sample were detected to support dual processing theory. However, Bayes factors indicate that many of these non-significant findings are due to the low sample size and data insensitivity instead of a preference for the null hypothesis. This modest sample is used to simulate 740 novel observations, and a power analysis is conducted on this dataset, offering potential effect sizes and minimum sample sizes required for investigating psychological dimensions within secure software development. For CRT-2, an effect in the expected direction is observed in a simulated sample of 93 participants. Rational and intuitive decision making styles are also seen to support the general hypothesis for samples of 179 and 371 participants, respectively, and both are in the expected direction to support the dual processing theory. That is, increased rationality was associated with increased detection, and higher intuitive styles were associated with lower detection rates. The findings from the groundwork sample, Bayes factors, and power simulation further evidence that conducting psychological research in the software engineering domain is complex, and potential associations are

likely masked by the environment itself. This chapter offers valuable research-related implications for other software engineering researchers, such as the need for sharing and standardisation of software behaviour measures and the role of power analyses to ensure sufficient research quality. The power analysis indicates that some significant effects are detected in modest samples, and further research should control the experimental setup carefully.

This chapter highlights the potential value of dual processing theory for software engineering research by conducting a groundwork study supported by a simulated dataset to determine likely effect sizes and directions of measures. The findings support the idea that dual processing is a valid theory to interpret security behaviours in software engineering. Increased system 2 use has been correlated with defect detection for functionality (Buffardi, 2023), and it is recognised that software development requires intensive cognitive engagement (Rauf et al., 2021) which emphasises the requirement for system 2 engagement.

### ***8.1.1 Contribution to Thesis Argument***

This chapter contributes to the thesis by empirically investigating the role of dual processing theory in the practical task of detecting security vulnerabilities in code. While previous chapters explored cognitive reflection and social identity through self-reported behaviours and thematic analysis, this chapter provides direct evidence from code comprehension experiments, reinforcing the argument that System 2 (reflective) thinking improves vulnerability detection, whereas intuitive (System 1) thinking may hinder it. The replication of blindspot challenges in vulnerability detection further grounds the theory in software engineering practice.

Despite a modest sample size limiting statistical power, the simulated data and power analyses offer valuable guidance for future research design, highlighting the complexity and demands of psychological research within software engineering contexts. These findings support the claim that cognitive styles, particularly the engagement of reflective

reasoning, are fundamental to secure software development behaviours.

By extending dual processing theory to the concrete domain of code comprehension, this chapter demonstrates that psychological constructs can be meaningfully integrated with software engineering tasks. It exemplifies how a behavioural science approach can offer new interpretations of persistent challenges in secure coding – specifically, that many security vulnerabilities remain undetected not because of technical failure but because of cognitive blindspots. Readers are encouraged to see this work as a bridge between the psychological sciences and empirical software engineering, contributing to a more complete, interdisciplinary account of secure development behaviour.

Together with earlier chapters on cognitive and social dimensions, this chapter strengthens the interdisciplinary framework underpinning the thesis: that security-related software engineering behaviour is best understood through integrating cognitive psychology and social identity theories with traditional software engineering concepts. In doing so, it reinforces a key claim of the thesis: that psychological and social dimensions are not peripheral to software engineering but central to understanding and improving developer behaviour.



## 9 General Discussion

Secure software development is not trivial, and software engineers face many competing factors, of functionality, usability, performance, and, importantly, security (Rauf et al., 2021). Despite a plethora of available support, including tools and documentation, engineers continue to produce insecure software (Weir et al., 2020b). Software is the by-product of human behaviour (John et al., 2005), and it can be hypothesised that individual differences and personal perceptions influence how software security is implemented. Based on these foundations, the thesis was designed – through empirical study – to investigate and illuminate the spaces between secure software development and those who attempt to develop it. This chapter integrates the specific work packages and reflects on the broader interpretations and conclusions.

The core of this thesis comprises five research chapters, covering five research studies that investigated the influence of cognitive and social psychology on secure software engineering behaviours. The thesis used a phased work plan. Phase 1 explored the soft skills relevant to software development. The findings were synthesised to emphasise potential latent psychological theories that could describe or predict variances in soft skill competencies. Moreover, two critical theories were identified, social identity theory (Abrams & Hogg, 1990) and the dual processing theory of decision making (Evans, 2003), as potentially fruitful theories to test because they offer explanations for why insecure behaviours persist. Phase 2 then tested these theories against security perceptions and secure coding behaviours.

Analysis of the literature demonstrated that relatively few empirical projects have applied contemporary psychological concepts to software development scenarios. Adjacent areas to the thesis were identified, such as cognitive bias (Mohanani et al., 2020) and working memory (Brun et al., 2023), with working memory being linked to learning programming (Prat et al., 2020). However, as far as can be established, minimal empirical work addresses dual processing theory or social identity concerning secure coding behaviours.

Starting with soft skills, which are more familiar concepts to software engineers, phase 1 helped inform and shape the choices of psychological theories that are likely to have the most impact on software engineering, as well as providing assurances that the research implications are relevant.

## 9.1 Summary of Results

In the first research paper, Chapter 4, a survey study investigated the valued soft skills of CS graduates to identify where these skills were developed. The CS sample was compared against psychology graduates. The latter group provided a comparison in that nearly 80% of CS students enter software engineering employment (Association of Graduate Careers Advisory, 2022) after their undergraduate degree. In contrast, psychology graduate employability is more diverse and psychology-specific careers typically require further study (Stamms et al., 2016). CS graduates reported that skills of *problem solving*, *analytical thinking*, and *methodical thinking* were developed during education, whereas more workplace-oriented skills, like *leadership* and *teamwork*, were associated with employment. In contrast to psychology graduates, CS graduates placed greater value on team skills. Key skills required for security included *communication*, *responsibility*, *problem solving*, *critical thinking*, *decision making*, *flexibility*, *initiative*, *innovation*, and *organisation*.

Chapter 5 reported on two convergent studies. In study one, I conducted a multi-site interview with CS educators to understand their perceptions of soft skills and how these skills are integrated into core modules. Valued skills embedded within the modules were separated into cognitive, social, and professional skills. Students were perceived to be initially resistant to the idea of soft skills, and changes in this are event-driven, typically through industry exposure. Study two extracted the soft skills embedded within curricula information, which found frequent mentions of *teamwork*, *communication*, *professional skills*, *ethical thinking*, *critical thinking*, and *problem solving*. Soft skills were often found in proximity, indicating they were taught or assessed in the same modules. The

exceptions to this were *problem solving* and *autonomy*, which were closely tied to *programming*, and *software engineering* was linked to *teamwork* and *communication*, highlighting the social nature of how software engineering is taught.

In Chapter 6, security and risks in software engineering were investigated as a function of measures of cognitive reflection, risk aversion, and unrealistic optimism bias. An interaction between cognitive reflection and unrealistic optimism for uncertainty-based language was identified, suggesting developers spoke about risk in complex ways. High optimism and reflection indicated greater uncertainty awareness and more pessimistic views reduced uncertainty as cognitive reflection increased. In general, developers were typically over-optimistic about personal susceptibility towards security vulnerabilities.

Chapter 7 focused on rich text responses from the same data as Chapter 6 and subjected them to a thematic analysis. Three core themes of Responsibility, Optimism, and Risk were identified that emphasised nuanced ways of interpreting security and risk sensitivity. These themes are characterised with respect to social identity theory, highlighting its applicability for software engineering. Responsibility was interpreted in numerous ways, through diffusion, displacement, or acceptance. Similarly, developers assessed risk through direction and a personal appetite for uncertainty.

In Chapter 8, dual processing theory was applied towards a code comprehension task to test the hypothesis that software vulnerabilities exist as cognitive blindspots. This chapter offered the first experimental exploration of dual processing theory in secure software engineering. Previous findings of Brun et al. (2023) were replicated – that blindspots are hard to detect, and that technical and security expertise have little effect on detection rates. Statistical power analyses indicated that dual processing theory has potential for successfully interpreting secure coding behaviours.

## 9.2 Answering Research Questions

Two research questions were posed to structure this thesis:

### 1. What non-technical skills (or soft skills) are valued within computer science and software engineering?

The first question is answered using the phase 1 findings. The non-technical skills valued by both graduates and educators align, and the most valued skills from each study are reported in Table 2. From this, all three studies unanimously identified *communication*, *teamwork*, *critical thinking*, and *problem solving*. These four skills can be grouped into social dimensions (communication & teamwork) and cognitive dimensions (critical thinking & problem solving).

Soft skills represent one dimension distinguishing between good and exceptional software developers (Capretz & Ahmed, 2018), and the thesis findings corroborate and extend previous research that reports the value of *communication* (Ahmed et al., 2012b; Montandon et al., 2021), *teamwork* (Groeneveld et al., 2020b; Jia et al., 2017), *problem solving* (Florea & Stray, 2018; Matturro, 2013), and *critical thinking* (Matturro et al., 2019).

Skill	Chapter 4	Chapter 5a	Chapter 5b
Communication	✓	✓	✓
Critical Thinking	✓	✓	✓
Problem Solving	✓	✓	✓
Teamwork	✓	✓	✓
Ethical Thinking		✓	✓
Conflict Management		✓	
Professional Skills			✓
Reading/Writing		✓	
Responsibility	✓		
Time Management		✓	

**Table 2**

*The skills identified in Chapters 4 and 5 as being important for software learning development. 5a refers to study one, and 5b to study two in Chapter 5. This graph is replicated from the Statement of Continuous Thesis Summary in Chapter 5.*

### 2. Can psychological variables and constructs shape security behaviours within software engineering?

The second question is answered throughout the research chapters, offering diverse findings and implications in sections 9.3 and 9.4. Here, I provide more detailed answers to this question, offering contributions to theoretical, practical, and research implementations for improving secure coding behaviours.

The soft skills identified in phase 1 were synthesised into latent theories of social identity and dual processing. The interaction identified in Chapter 6 suggested a complex relationship between perceptions of security risks and cognition, highlighting that the uncertainty awareness of developers is not linearly tied to measures of cognitive reflection. Chapter 7 explored the same dataset quantitatively and identified the role of social identity influencing responsibility around software security. Responsibility is identified in Chapter 4 as a valued soft skill, and Chapter 7 reinforces this with social identity interpretations integrating responsibility, communication, and group work. In Chapter 8, the relationship between cognition and vulnerability detection was variable; the groundwork sample demonstrated minimal support, but further analyses indicated this was likely due to sample size. These findings suggest that while cognition and social psychology can explain variance in secure coding practices, the relationship is complex and further research requires careful control of possible confounds.

### **9.3 Theoretical Implications**

Calls have been made for more psychology research within software engineering and greater integration of psychology and software engineering research (Acar et al., 2016b; Capretz & Ahmed, 2018), and my thesis answers these calls. The research chapters are preprinted and have all been published or submitted to peer-reviewed venues to increase the visibility of the work conducted in this area. The thesis delivers research findings based on two contemporary psychological theories, responding to the call by testing empirically cognition and social identity within secure software engineering.

### ***9.3.1 Soft Skills and Pedagogy***

The findings from phase 1 offer complementary perspectives on how software engineers are exposed to soft skills during their education. Using standardised definitions for otherwise ephemeral concepts, phase 1 identified the most valued soft skills through a triangulation process. The findings speak to the possibility that the soft skills gap (Akdur, 2021) is affected by the student’s reluctance to embrace the value of soft skills until they experience the need for them, rather than a misalignment between industry and educator expectations. These findings indicate the experiential impact of coming face to face with soft skills in more professional contexts. These event-driven catalysts are seen in other learning domains too, supporting the idea of event-driven development. Exposing secondary school students to practical data collection engages them in the activity and increases their motivation for tasks involving their own data compared to pre-existing data (Martin et al., 2010). Similarly, medical trainees are often exposed to practical experience as early as possible to develop their skills (Sawyer et al., 2015).

### ***9.3.2 Cognitive Psychology Applications***

This thesis is not the first place where dual processing theory has been applied towards software engineering, but much of the literature was published “in parallel” with the thesis. This work has been primarily conceptual rather than empirical (e.g., Petre, 2022; Pretorius et al., 2018; Robins, 2022), with one empirical study correlating cognitive reflection against functional defect detection (Buffardi, 2023). To my knowledge, this thesis is the first empirical application of dual processing theory within secure software development, but the existing work offers support to the notion that it is a suitable theory to apply toward decision making in secure software engineering.

In Chapters 6 and 8, I applied the theory to make quantitative predictions, supported by subsequent analysis. Specifically, Chapter 6 establishes an interaction between optimism bias and cognitive reflection for security perceptions, indicating that the psychological theories that often demonstrate clear effects in general populations are more nuanced in

applied contexts. Similarly, Chapter 8's power analysis evidenced the need for large samples to detect effects related to dual processing, or precise empirical measures to reduce noise. However, in some cases, the data tended towards accepting the null hypothesis (that dual processing has no role in predicting security behaviours), reiterating the uncertainty and nuances of software engineering psychology.

These findings raise the question, "Why is the effect more nuanced than in standard psychology experiments?". One answer is that no direct link between dual processing decision making exists, and secure coding simply *is* very complex. Another answer lies in the research approach, where the designs used so far were not sufficient for isolating decision making states from other influences in secure coding. The thesis cannot fully answer this question, but next steps (as outlined in 9.8) would suggest that the causal nature of dual processing needs to be tested to provide better answers to this question.

The findings align with previous work, where software development requires cognitive effort (Rauf et al., 2021), which demands system 2 engagement and consequently means that it is being deployed in competing contexts such as for functionality as well as security. System 2 engagement correlates with increased defect detection for functionality tests (Buffardi, 2023), supporting the thesis findings for the relevance within secure contexts.

### ***9.3.3 Social Psychology Applications***

Chapter 7 highlights how responsibility for security can be conceptualised through social identity theory. The thesis findings offer support for diverse social identities existing in secure software development and for interpreting how social identity may influence approaches towards responsibility and risk acceptance. Rauf et al. (2022) identified that security perceptions in developer communities are not homogenous, and in explorations of freelance communities, Rauf et al. (2023) identified diverse ways in which these perceptions manifest. Security cultures and motivations for security use range from internal motivators, such as feelings of responsibility, to external requirements, such as

app store policies. It was also seen that some freelancers do not consider security as a concern, citing reasons including that security is only necessary for large projects, or that their processes are already secure enough. This work, published in parallel with the PhD (while Chapter 7 underwent peer review), provides supporting evidence for the findings in Chapter 7. The thesis focuses on examining these security perceptions through the lens of social identity, enabling a greater interpretation of *why* these perceptions may exist within the freelancer community. The thesis leverages pre-existing, well-researched psychological constructs for investigating causal relationships between social groups and security thinking.

The success of group-based work is about more than just social factors of communication and group management. Group diversity, in terms of individual cognitive differences, can be valuable for the balancing of perspectives (Shreeve et al., 2022), and this diversity can offer opportunities for greater engagement in more reflective thinking (Coffeng et al., 2023). Developers are encouraged to seek multiple perspectives while secure coding to increase deliberative thinking.

#### **9.4 Practical Implications**

The practical implications and applications of this thesis are diverse due to the research approach. They also differ in terms of their realistic “delivery schedules”. That is, some are more speculative than others, and some are more fully formed. They include opportunities to improve curricula, promote security behaviours from freelancers, and educate current and developing engineers about the possible dangers of working intuitively and its impact on secure code.

It is worth reiterating a point made in Chapter 8 that the findings from phase 2 *should not* be used as performance indicators or recruitment metrics. Instead, the thesis emphasises the need for multiple perspectives within software development, from ranges of cognitive styles to the strength of social identities. These provide a balancing of perspectives, and if all developers presented homogeneous cognitive states and social ties,



then only technical expertise and experience would predict secure coding, which Chapter 8 indicates is not true.

#### ***9.4.1 Curriculum Design***

Findings from phase 1 offer implications towards curriculum design. Problem solving was strongly linked to graduates' education, suggesting the undergraduate programme delivers this skill well. However, social communication and teamwork skills were reportedly not associated with education. A larger issue exists within education, in that educators need to teach what students prospectively need, and this can change dramatically over the course of an individual's career. What an educator was taught is not necessarily what they should teach. This emphasises the importance of teaching principles rather than declarative information because this supports flexibility in preparation of the future. Chapters 4 and 5 suggest incorporating project-based learning with involvement from industry partners to speed up the recognition of soft skill's value, allowing students a controlled environment to practice these skills. Natural language processing approaches can be utilised to check curriculum material for soft skill distributions and ensure they are embedded across the course.

In Chapter 5, educators emphasised that ethical thinking is often taught in an overt, direct way and forms an integral part of the curriculum, a reflection of the inclusion of ethics in the SE2014 (Ardis et al., 2015), SWEBOK (Bourque & Fairley, 2014), and CyBOK (Rashid et al., 2021). However, Chapter 4 showed that graduates place a low value on the importance of ethics. The reasons for this are unclear, but it may be due to differences in how education and industry understand ethics. Even frameworks such as SWEBOK state that staying abreast of contemporary ethical standards is the individual's responsibility, and the academic equivalent, SE2014, indicates educators should ensure students are familiarised with the ACM/IEEE code of ethics without further scaffolding. To mitigate this, educators and industry partners should work together to identify the necessary ethical components, offering enhanced support for student's ethical awareness

and development.

#### ***9.4.2 Closing the Temporal Gap***

Chapter 5 finds students do not always recognise the value of soft skills, avoiding development opportunities or disengaging with course content on their topic, and this is potentially reflected in the finding that despite teamwork and communication being taught and emphasised within core module content, graduates report it as being developed outside of education. These findings echo the event-driven nature of soft skill development, pointing to the need for educators to provide these catalytic events. Educators should seek to develop activities that require genuine team elements with a strong, practical, and industry-applicable motivation.

#### ***9.4.3 Rewards for Secure Coding***

Chapter 6 suggests that reward systems can be implemented to engage engineers in reflective or critical thinking to boost secure coding. Rewards can increase personal motivation to achieve quality work, which typically involves greater cognitive effort (Kunda, 1990). Reward systems do not need to be financially based when working with freelancers, as public recognition through gamification (such as profile badges) can be used to increase traffic to freelancers recognised for producing secure code. Lopez et al. (2018) explored the use of badges and points on StackOverflow, finding that they are beneficial for increasing engagement. Other methods exist for the improvement of security awareness through gamification, such as the Motivating Jenny project (Lopez et al., 2020b) who presented game-based interventions for raising security awareness in developer teams.

#### ***9.4.4 Increasing accountability***

Chapter 7 highlighted how responsibility is framed and that social identities can lead to a reduced acceptance of personal responsibility. Management should emphasise a top-down security culture and enable freelancers or temporary staff to share these values, leading to greater individual responsibility. Methods for ensuring freelancers adopt similar social

identities are beyond the scope of this thesis but making security a substantial part of an organisation's branding, certainly during recruitment, can reinforce this value as being required for potential employees. Increasing accountability of individuals can be used to enhance the awareness of risk directed to oneself, which would also boost responsibility. Similarly, ensuring developers are cognisant of their users and reducing the social distance between developer and user may also increase personal responsibility for secure coding.

#### ***9.4.5 Educating engineers on decision making styles***

In Chapter 8, I replicated the finding that programming experience and technical knowledge have little effect on software vulnerability detection and found indications that dual processing theory can explain individual differences in vulnerability detection. Educators can use these findings to educate CS students (and current developers) about the dangers of assuming that technical knowledge guarantees secure software. Teaching these populations about how intuitive decision making is not the best approach for writing code, can be used to increase developers' awareness of the need to engage in reflective and critical styles of thinking.

### **9.5 Final Thesis Structure**

As noted in the Chapter 3 statement, there are differences between the intended thesis structure proposed in Chapter 3 and the final thesis structure. Namely, three major differences are worth discussing.

1. The work plan in Chapter 3 mentioned three phases, but the thesis only mentions phases 1 and 2. Phase 3 comprised methods for disseminating research through a) seminars, workshops, and tutorials and b) research publications. As of thesis submission, two chapters are published in peer-reviewed venues, and four are published on preprint servers while under peer review. Phase 3 intended to cover intervention development to raise awareness of psychological dimensions in pedagogy, but it was not possible to incorporate this into the programme due to resource constraints.

2. The intended plan had a student interview study in phase 1 that was not conducted.

The primary reasons for this omission were finite resources and an initial naïvety about the investment that would be required. In addition, the thesis timeline did not align with a schedule to follow a cohort *throughout* the academic calendar and ideally over *multiple* years. This timeline was not possible and would have detracted from the feasibility of what was undertaken. Phase 1 findings indicate that this study is still beneficial to thoroughly test the idea that the skill gap is attributable to student misconceptions.

3. The intended phase 2 plan mentioned three studies, and the final phase 2 work only contains one of these studies, with Chapters 6 and 7 being defined as a “standalone” study. Again, naïvety accounts for the absence of debugging and database creation tasks. That is, there are reasons why some empirical paradigms are currently rare in the research field. The methodological development of stimuli and protocols is neither straightforward nor quick. Nonetheless, the final thesis outcome still provides empirical substance, breadth, and coherence. Additionally, as highlighted in Chapter 8, studying cognition in software engineering requires a large sample size, which would not have been possible to cover with the PhD funding alone.

Taken together, the studies in this thesis demonstrate that secure software development is not solely a technical challenge, but a cognitive and social one. By combining behavioural science and psychological methods with empirical software engineering, this thesis offers an interdisciplinary framework that positions soft skills as central to understanding and improving secure coding behaviours. Across graduate perceptions, staff practice, developer psychology, and experimental code comprehension, the thesis shows that reasoning styles, social identities, and reflective capacities shape how developers perceive, discuss, and respond to risk. This work thus invites readers to view secure software engineering through a psychological and sociotechnical lens, one that complements, rather than competes with, technical expertise.

## 9.6 Reflections on Methodological Coherence and Theoretical Integration

This thesis adopted a socio-cognitive and pragmatic mixed-methods framework, which informed both the design and interpretation of the research studies. Across diverse methodologies there was a consistent theoretical commitment to understanding computing and secure behaviours soft skills as both individually internalised and socially situated psychological phenomena.

By aligning the qualitative studies with a constructionist epistemology, the research was able to explore how participants co-constructed meaning in their specific institutional and disciplinary contexts. Thematic analysis, selected for its flexibility and alignment with this epistemological stance, enabled the identification of shared meaning patterns across groups, which could then be generalised conceptually to inform subsequent studies.

The quantitative studies, framed within an experiential realist epistemology, extended this inquiry by testing models of behaviour and latent psychological constructs. While these paradigms differ epistemologically, they were united by a socio-cognitive theoretical lens that emphasises the interaction between internal beliefs and external structures in shaping outcomes. Importantly, the thesis demonstrates how a methodologically pluralist approach, when guided by a coherent theoretical frame, can produce high quality, valuable findings. Rather than viewing qualitative and quantitative studies as isolated or merely additive, the thesis treated them as complementary.

This integration supports a triangulated understanding of social and cognitive behaviours. An understanding that reflects both the measurable and the meaningful, the cognitive and the contextual. In this way, the thesis provides an example of how socio-cognitive approaches can function as a unifying framework across methodological boundaries, yielding insights that neither approach could achieve alone.

## 9.7 Limitations

Specific research limitations and threats to validity are detailed in the relevant chapters. This section focuses on the broader constraints that affect the thesis.

The studies were conducted online because of the coronavirus pandemic that mandated physical distancing measures and partly because of the populations targeted. The interview study in Chapter 5 could have been in-person, but participants volunteered for online interviews. In other words, online data collection primarily occurred through force of circumstances. There are notable benefits to online data collection, including increased diversity, faster data collection, and the ability to reach dispersed populations. Still, there are downsides, too. Online participants may demonstrate reduced attentiveness compared to in-person paradigms (Newman et al., 2021), likely explained by scenarios where participants complete experiments in parallel with other activities, such as watching television. The juggling of multiple tasks can reduce attention and validity as participants are less motivated to provide detailed information, distorting the effect sizes observed. A weakened effect increases the likelihood that null effects are detected where a true effect should exist. For example, weakened motivation for information provision was a specific issue in Chapters 6 and 7, but controls were set to reduce this behaviour, including timers restricting participants from submitting answers too soon. However, as organisations adopt hybrid or remote working patterns, online studies likely reflect working environments. Particularly with software engineers, where most of their time is conducted online or in front of a computer, engaging in online digital experiments may be similar to typical environments.

It is also harder to control for diversity within online samples (Casler et al., 2013), and within-participant variability is likely higher than in-person studies, meaning effect sizes may be diminished. Despite these potential issues, Casler et al. (2013) did not find evidence for differences between online and in-person studies, suggesting it may not be an issue. However, perhaps a more unique limitation to the thesis than is commonly seen in

general psychological research is related to online participant recruitment. Users of freelance platforms are diverse and self-identifying or, rather, self-branding (Blyth et al., 2022). For researchers, this makes participant screening more involved than other recruitment strategies, as freelancer profiles contain the content they wish to promote. As discovered in Chapter 8, many profiles purporting to be experienced software engineers were inexperienced CS students. This increased the time investment needed for screening, and the potential diversity within the community is far greater than advertised. Despite this, freelance communities are still valuable populations to understand, as solo developers increasingly produce software (van der Linden et al., 2020a), and freelance communities are recognised as a unique population of interest (Rauf et al., 2023).

Another issue with using online participants is that with the introduction of artificial language models (e.g., ChatGPT), participants used these tools to respond to screening questions. These responses look promising, but upon reading, it is clear they possess no real value. This increases time investment. While a limiting factor, it does offer one straightforward method of screening. In software engineering, ChatGPT is a valid tool for assisting software production. However, it was not expected that participants would use it to answer rich text answers, and it can become difficult to disentangle what is written by a participant or AI. This only affected Chapter 8, as all other studies were conducted prior to ChatGPT release, or were interview studies. To reduce the chance of programming-related questions (such as code comprehension) being answered by AI in Chapter 8, I provided code snippets as images rather than text, which minimised participants' ability to copy-paste the code directly into AI tools. Due to the timeline of the thesis, no other studies were affected.

The thesis was intended to comprise high-quality research with suitably powered statistical analyses and comprehensive qualitative information. Adequate power usually means larger sample sizes, which comes with the trade-off of cost. The PhD budget was limited in its purchasing power when acknowledging the higher-than-average cost of

recruiting professional software engineers compared to compensating undergraduate psychology students. Chapter 8 exemplifies this tension, where only 37 participants were recruited. Data simulation was used to carry out a power analysis to address this shortfall. Different populations were targeted to address tensions between powered research and available resources (including graduates, staff, and freelancers) by triangulation of these sources.

## 9.8 Further Work

This thesis sets the stage for further research into the cognitive and social constructs that may inform security behaviours within software engineering. Chapter 8 culminates at a stage that indicates the need for a well-powered confirmatory study that addresses the potential noise in collecting psychological measures in software engineers. Deploying further studies of dual processing theory for secure coding in brownfield tasks (such as reviewing code) and greenfield tasks (such as writing secure code) can boost our understanding of the relationship between cognitive state and secure behaviours. These findings can then be used to develop interventions that leverage individual cognitive differences to reveal causal relationships between cognition and secure coding.

The broader influence of cognition in secure coding warrants further exploration beyond dual processing theory. As this area of research is still in its early stages, it is pertinent to investigate cognition from multiple perspectives to ensure that potential influences are suitably addressed. For example, revisiting the findings reviewed in Chapter 2 that working memory has little influence in secure coding, it is beneficial to understand why this may have occurred despite displaying some effect in bug detection. Working memory has clear links with task success and high-level cognition (Baddeley & Hitch, 1974), so why is it not considered an attribute of secure coding? Speculatively, code-based activities do not require high levels of working memory as overloads are absorbed into tools. The high variability in individual computer environments may accommodate for working memory deficiencies. Further research that controls variables that may attenuate



working memory in secure coding would enhance our understanding of individual cognitive differences in this domain.

The findings from the social identity work in Chapter 7 indicated that the perceived social distance between developers and their users can modulate feelings of responsibility for ensuring software is secure. Incorporating elements of the design used in Rauf et al. (2022), where individuals were primed to think of themselves as members of the developer community or not, the influence of social identity strength can be explored against the vulnerability detection task used in Chapter 8. This work would offer multiple advancements in that using the same materials as Chapter 8 allows for easier comparisons between findings, and previous work recognises that relationships exist between social identities and approaches to secure coding (see Chapter 7 and Rauf et al., 2023 for examples). However, to date, no causal relationships have been identified.

The thesis also emphasises the need for future work to ensure that adequate sample sizes are used, and that study designs should be carefully controlled to ensure that detected effects contain as minimal noise as possible. It is seen throughout the research, such as in Chapter 8, that expected effects were not reported and that this is assumed to be the result of participants using AI to answer psychological measures. Any work carried out in this space should be carefully planned to reduce possible confounds.

These research areas are not the only possible research developments but represent the thesis work's direct progression. The thesis lays the groundwork for these research programmes, evidencing their likelihood of success by providing a better understanding of how secure coding can be achieved and offer the identification of relationships between psychological concepts and secure software engineering currently lacking in secure software engineering. Further work must seek to *observe* these relationships and *confirm* their causal nature.

## 9.9 Conclusion

This thesis highlights that the psychological dimensions of software engineers are complex. Secure software development and the behaviours exhibited within are multifaceted. The development cycle has many competing interests, including functionality, usability, and security (Rauf et al., 2021). This complex environment means that understanding the psychology of individuals is vital but difficult to isolate. In this thesis, I approached the absence of psychological research in software engineering by first understanding the valuable soft skills for software engineering, before translating these into latent psychological theories that were applied to secure software engineering behaviours.

In phase 1, I identified the key soft skills present within CS education, finding that students only learn some of their valued skills during education, despite the efforts of educators to provide opportunities for students to develop these skills. The core skills could be grouped: communication & teamwork, and problem solving & critical thinking. Social identity theory (Haslam, 2012) and the dual processing theory of decision making (Evans, 2003) were identified as suitable methods for interpreting the psychology of software engineering.

In phase 2, through an analysis of developer-held perceptions of security risks, the role of system 2 activation (measured by CRT) was found to have an effect in interaction with optimistic beliefs. The role of social identity was applied to developer perceptions, and how developers handle responsibility and risk can be interpreted through understanding social identities and the representation of ingroup and outgroup behaviours. This research was followed by a replication study that confirmed blindspots in code are difficult to detect, as well as a further development that tested dual processing theory's relationship with vulnerability detection. No significant effects were found within the groundwork sample for dual processing, but power simulations suggest that suitably powered studies would find effects in samples between 100 and 200 participants, depending on which measures are used.

The thesis has implications for software learning development in CS education and for practising software engineers. Phase 1 advocates the integration of meaningful social skill exercises that expose students to the reality of these skills' value, with the aim to close the temporal gap observed and have students recognise the value of soft skills during education, not after. Phase 2 findings suggest that teaching opportunities should be used to highlight aspects of dual processing and how intuitive programming can be detrimental to security. For practising engineers, phase 2 indicates language can offer insight into how developers perceive others in the development cycle, which can be used to identify potential areas where responsibility can be refused or where unacceptable risks are taken. The findings suggest developers should be conscious about how their documentation is written and whether insecure default or potentially opaque descriptions can reduce other developers' ability to produce secure code.

By establishing a role for social identity theory and dual processing theory as the latent dimensions of key soft skills, the research findings offer a small but meaningful contribution to understanding how individuals influence secure software development. This thesis provides multiple answers to the call for increased psychology research in software engineering (Capretz & Ahmed, 2018) and explains why secure coding behaviours are not always observed. It is hoped that the thesis research will inspire others to research the interdisciplinary area of secure software development.

### Consolidated Bibliography

- Abdellaoui, M., Bleichrodt, H., & Paraschiv, C. (2007). Loss Aversion Under Prospect Theory: A Parameter-Free Measurement. *Management Science*, 53(10), 1659–1674. <https://doi.org/10.1287/mnsc.1070.0711>
- Abrams, D., & Hogg, M. (1990). An Introduction to the Social Identity Approach. In *Social identity theory: Constructive and critical advances* (pp. 1–9). Harvester Wheatsheaf. <https://books.google.co.uk/books?id=hJjZAAAAMAAJ>
- Acar, Y., Backes, M., Fahl, S., Garfinkel, S., Kim, D., Mazurek, M. L., & Stransky, C. (2017a). Comparing the Usability of Cryptographic APIs. *2017 IEEE Symposium on Security and Privacy (SP)*, 154–171. <https://doi.org/10.1109/SP.2017.52>
- Acar, Y., Backes, M., Fahl, S., Kim, D., Mazurek, M. L., & Stransky, C. (2016a). You Get Where You're Looking for: The Impact of Information Sources on Code Security. *2016 IEEE Symposium on Security and Privacy (SP)*, 289–305. <https://doi.org/10.1109/sp.2016.25>
- Acar, Y., Backes, M., Fahl, S., Kim, D., Mazurek, M. L., & Stransky, C. (2017b). How Internet Resources Might Be Helping You Develop Faster but Less Securely. *IEEE Security Privacy*, 15(2), 50–60. <https://doi.org/10.1109/MSP.2017.24>
- Acar, Y., Fahl, S., & Mazurek, M. L. (2016b). You are Not Your Developer, Either: A Research Agenda for Usable Security and Privacy Research Beyond End Users. *IEEE Cybersecurity Development*, 3–8. <https://main.sec.uni-hannover.de/publications/conf-secdev-acarfm16/>
- Acar, Y., Stransky, C., Wermke, D., Weir, C., Mazurek, M. L., & Fahl, S. (2017c). Developers Need Support, Too: A Survey of Security Advice for Software Developers. *2017 IEEE Cybersecurity Development (SecDev)*, 22–26. <https://doi.org/10.1109/SecDev.2017.17>
- Adams, A., & Sasse, M. A. (1999). Users are not the enemy. *Communications of the ACM*, 42(12), 40–46. <https://doi.org/10.1145/322796.322806>
- Agbo, F. J., Yigzaw, S. T., Sanusi, I. T., Oyelere, S. S., & Mare, A. H. (2021). Examining

- theoretical and pedagogical foundations of computational thinking in the context of higher education. *2021 IEEE Frontiers in Education Conference (FIE)*, 1–8.  
<https://doi.org/10.1109/FIE49875.2021.9637405>
- Ahmed, F., Capretz, L. F., Bouktif, S., & Campbell, P. (2012a). Soft skills requirements in software development jobs: a cross-cultural empirical study. *Journal of Systems and Information Technology*, *14*(1), 58–81. <https://doi.org/10.1108/13287261211221137>
- Ahmed, F., Capretz, L. F., Bouktif, S., & Campbell, P. (2015). Soft Skills and Software Development: A Reflection from Software Industry. *International Journal of Information Processing and Management*, *4*(3), 171–191.  
<https://doi.org/10.48550/arXiv.1507.06873>
- Ahmed, F., Capretz, L. F., & Campbell, P. (2012b). Evaluating the Demand for Soft Skills in Software Development. *IT Professional*, *14*(1), 44–49.  
<https://doi.org/10.1109/MITP.2012.7>
- Ait Aomar, A. (2023). SkillNER: A (smart) rule based NLP module to extract job skills from text [{{GitHub}} Repository]. In *SkillNER*.  
<https://github.com/AnasAito/SkillNER>.
- Ajila, C., & Abiola, A. (2004). Influence of Rewards on Workers Performance in an Organization. *Journal of Social Sciences*, *8*(1), 7–12.  
<https://doi.org/10.1080/09718923.2004.11892397>
- Akdur, D. (2021). Skills Gaps in the Industry: Opinions of Embedded Software Practitioners. *ACM Transactions on Embedded Computing Systems*, *20*(5), 43:1–43:39.  
<https://doi.org/10.1145/3463340>
- Alnuaimi, O. A., Robert, L. P., & Maruping, L. M. (2010). Team Size, Dispersion, and Social Loafing in Technology-Supported Teams: A Perspective on the Theory of Moral Disengagement. *Journal of Management Information Systems*, *27*(1), 203–230.  
<https://doi.org/10.2753/MIS0742-1222270109>
- Alter, A. L., & Oppenheimer, D. M. (2008). Effects of Fluency on Psychological Distance and Mental Construal (or Why New York Is a Large City, but New York Is a Civilized

- Jungle). *Psychological Science*, 19(2), 161–167.  
<https://doi.org/10.1111/j.1467-9280.2008.02062.x>
- Alter, A. L., Oppenheimer, D. M., Epley, N., & Eyre, R. N. (2007). Overcoming intuition: Metacognitive difficulty activates analytic reasoning. *Journal of Experimental Psychology: General*, 136(4), 569–576. <https://doi.org/10.1037/0096-3445.136.4.569>
- Amaral, M. J., & Monteiro, M. B. (2002). To be without being Seen: Computer-Mediated Communication and Social Identity Management. *Small Group Research*, 33(5), 575–589. <https://doi.org/10.1177/104649602237171>
- Amin, M. R., & Bhowmik, T. (2021). Information on Potential Vulnerabilities for New Requirements: Does It Help Writing Secure Code? *2021 IEEE 29th International Requirements Engineering Conference (RE)*, 408–413.  
<https://doi.org/10.1109/RE51729.2021.00046>
- Andersson, C., & Logofatu, D. (2018). Using cultural heterogeneity to improve soft skills in engineering and computer science education. *2018 IEEE Global Engineering Education Conference (EDUCON)*, 191–195.  
<https://doi.org/10.1109/EDUCON.2018.8363227>
- Anu, V., Sultana, K. Z., & Samanthula, B. K. (2020). A Human Error Based Approach to Understanding Programmer-Induced Software Vulnerabilities. *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, 49–54. <https://doi.org/10.1109/ISSREW51248.2020.00036>
- Anu, V., Walia, G., Hu, W., Carver, J., & Bradshaw, G. (2016). *Effectiveness of Human Error Taxonomy during Requirements Inspection: An Empirical Investigation*. 531–536. <https://doi.org/10.18293/SEKE2016-177>
- Ardis, M., Budgen, D., Hislop, G. W., Offutt, J., Sebern, M., & Visser, W. (2015). SE 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. *Computer*, 48(11), 106–109. <https://doi.org/10.1109/MC.2015.345>
- Armstrong, M. E., Jones, K. S., Namin, A. S., & Newton, D. C. (2020). Knowledge, Skills, and Abilities for Specialized Curricula in Cyber Defense: Results from

- Interviews with Cyber Professionals. *ACM Transactions on Computing Education*, 20(4), 29:1–29:25. <https://doi.org/10.1145/3421254>
- Assal, H., & Chiasson, S. (2018a). Motivations and Amotivations for Software Security. *Fourteenth Symposium on Usable Privacy and Security*, 4.
- Assal, H., & Chiasson, S. (2018b). Security in the Software Development Lifecycle. *Proceedings of the Fourteenth USENIX Conference on Usable Privacy and Security*, 281–296. <https://doi.org/10.5555/3291228.3291251>
- Assal, H., & Chiasson, S. (2019). 'Think secure from the beginning'; A survey with Software Developers. *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems - CHI '19*, 1–13. <https://doi.org/10.1145/3290605.3300519>
- Association of Graduate Careers Advisory. (2022). *What can I do with a computer science degree? | Prospects.ac.uk*. What Can I Do with a Computer Science Degree? <https://www.prospects.ac.uk/careers-advice/what-can-i-do-with-my-degree/computer-science>
- Auhagen, A. E., & Bierhoff, H.-W. (2002). *Responsibility: The Many Faces of a Social Phenomenon*. Taylor & Francis.
- Backevik, A., Tholén, E., & Gren, L. (2019). Social Identity in Software Development. *12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 107–114. <https://doi.org/10.1109/chase.2019.00033>
- Bada, M., Sasse, A. M., & Nurse, J. R. C. (2019). Cyber Security Awareness Campaigns: Why do they fail to change behaviour? *arXiv:1901.02672 [Cs]*. <http://arxiv.org/abs/1901.02672>
- Baddeley, A. D. (1986). *Working memory* (pp. xi, 289). Clarendon Press/Oxford University Press.
- Baddeley, A. D., & Hitch, G. (1974). Working Memory. In G. H. Bower (Ed.), *Psychology of Learning and Motivation* (Vol. 8, pp. 47–89). Academic Press. [https://doi.org/10.1016/S0079-7421\(08\)60452-1](https://doi.org/10.1016/S0079-7421(08)60452-1)
- Baham, C., & Hirschheim, R. (2021). Issues, challenges, and a proposed theoretical core

- of agile software development research. *Information Systems Journal*, 32(1), 103–129.  
<https://doi.org/10.1111/isj.12336>
- Bailin, S., & Siegel, H. (2003). Critical Thinking. In *The Blackwell Guide to the Philosophy of Education* (pp. 181–193). John Wiley & Sons, Ltd.  
<https://doi.org/10.1002/9780470996294.ch11>
- Baker, M. (2016). 1,500 scientists lift the lid on reproducibility. *Nature*, 533(7604), 452–454. <https://doi.org/10.1038/533452a>
- Balaji, K. V. A., & Somashekar, P. (2009). A Comparative Study of Soft Skills Among Engineers. *IUP Journal of Soft Skills*, 3(3/4), 50–57.
- Balliet, D., Wu, J., & De Dreu, C. K. W. (2014). Ingroup favoritism in cooperation: A meta-analysis. *Psychological Bulletin*, 140, 1556–1581.  
<https://doi.org/10.1037/a0037737>
- Bancino, R., & Zevalkink, C. (2007). Soft Skills: The New Curriculum for Hard-Core Technical Professionals. *Techniques: Connecting Education and Careers (J1)*, 82(5), 20–22.
- Bandura, A. (1986). *Social foundations of thought and action: A social cognitive theory* (pp. xiii, 617). Prentice-Hall, Inc.
- Bandura, A. (2002). Selective Moral Disengagement in the Exercise of Moral Agency. *Journal of Moral Education*, 31(2), 101–119.  
<https://doi.org/10.1080/0305724022014322>
- Banyard, V. L., & Miller, K. E. (1998). The powerful potential of qualitative research for community psychology. *American Journal of Community Psychology*, 26(4), 485–505.  
<https://doi.org/10.1023/A:1022136821013>
- Barreto, C. F., & França, C. (2021). Gamification in Software Engineering: A literature Review. *2021 IEEE/ACM 13th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 105–108.  
<https://doi.org/10.1109/CHASE52884.2021.00020>
- Baucus, M. S., & Beck-Dudley, C. L. (2005). Designing Ethical Organizations: Avoiding



- the Long-Term Negative Effects of Rewards and Punishments. *Journal of Business Ethics*, 56(4), 355–370. <https://doi.org/10.1007/s10551-004-1033-8>
- Baum, T., Schneider, K., & Bacchelli, A. (2019). Associating working memory capacity and code change ordering with code review performance. *Empirical Software Engineering*, 24(4), 1762–1798. <https://doi.org/10.1007/s10664-018-9676-8>
- BCS. (2023). University Computing departments met with record applicant numbers as AI hits the mainstream | BCS [Blog]. In *University Computing departments met with record applicant numbers as AI hits the mainstream*. <https://www.bcs.org/articles-opinion-and-research/university-computing-departments-met-with-record-applicant-numbers-as-ai-hits-the-mainstream/>.
- Beike, D. R., & Sherman, S. J. (1994). Social Inference; Inductions, Deductions, and Analogies. In R. S. Wyer & T. K. Srull (Eds.), *Handbook of social cognition* (2nd ed.). L. Erlbaum Associates.
- Bergersen, G. R., & Gustafsson, J.-E. (2011). Programming Skill, Knowledge, and Working Memory Among Professional Software Developers from an Investment Theory Perspective. *Journal of Individual Differences*, 32(4), 201–209. <https://doi.org/10.1027/1614-0001/a000052>
- Bethlehem, J. (2010). Selection Bias in Web Surveys. *International Statistical Review*, 78(2), 161–188. <https://doi.org/10.1111/j.1751-5823.2010.00112.x>
- Białek, M., & Pennycook, G. (2018). The cognitive reflection test is robust to multiple exposures. *Behav Res Methods*, 50(5), 1953–1959. <https://doi.org/10.3758/s13428-017-0963-x>
- Blackwell, A. F., Britton, C., Cox, A., Green, T. R. G., Gurr, C., Kadoda, G., Kutar, M. S., Loomes, M., Nehaniv, C. L., Petre, M., Roast, C., Roe, C., Wong, A., & Young, R. M. (2001). Cognitive Dimensions of Notations: Design Tools for Cognitive Technology. In M. Beynon, C. L. Nehaniv, & K. Dautenhahn (Eds.), *Cognitive Technology: Instruments of Mind* (pp. 325–341). Springer. [https://doi.org/10.1007/3-540-44617-6\\_31](https://doi.org/10.1007/3-540-44617-6_31)

- Blackwell, A. F., Petre, M., & Church, L. (2019). Fifty years of the psychology of programming. *International Journal of Human-Computer Studies*, 131, 52–63.  
<https://doi.org/10.1016/j.ijhcs.2019.06.009>
- Blyth, D. L., Jarrahi, M. H., Lutz, C., & Newlands, G. (2022). Self-branding strategies of online freelancers on Upwork. *New Media & Society*, 14614448221108960.  
<https://doi.org/10.1177/14614448221108960>
- Borowa, K., Zalewski, A., & Kijas, S. (2021). The Influence of Cognitive Biases on Architectural Technical Debt. *2021 IEEE 18th International Conference on Software Architecture (ICSA)*, 115–125. <https://doi.org/10.1109/ICSA51549.2021.00019>
- Bourque, P., & Fairley, R. E. (2014). *Guide to the Software Engineering Body of Knowledge* (No. 3.0; pp. 1–335). IEEE Computer Society.  
<https://ieeecs-media.computer.org/media/education/swebok/swebok-v3.pdf>
- Brachten, F., Brünker, F., Frick, N. R. J., Ross, B., & Stieglitz, S. (2020). On the ability of virtual agents to decrease cognitive load: an experimental study. *Information Systems and e-Business Management*, 18(2), 187–207.  
<https://doi.org/10.1007/s10257-020-00471-7>
- Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3(2), 77–101. <https://doi.org/10.1191/1478088706qp063oa>
- Brewer, M. B. (2017). Intergroup discrimination: Ingroup love or outgroup hate? In *The Cambridge handbook of the psychology of prejudice* (pp. 90–110). Cambridge University Press. <https://doi.org/10.1017/9781316161579.005>
- Brun, Y., Lin, T., Somerville, J. E., Myers, E. M., & Ebner, N. C. (2022). Blindspots in Python and Java APIs Result in Vulnerable Code. *ACM Transactions on Software Engineering and Methodology*. <https://doi.org/10.1145/3571850>
- Brun, Y., Lin, T., Somerville, J. E., Myers, E. M., & Ebner, N. C. (2023). Blindspots in Python and Java APIs Result in Vulnerable Code. *ACM Transactions on Software Engineering and Methodology*. <https://doi.org/10.1145/3571850>
- Brun, Y., Lin, T., Somerville, J. E., Myers, E., & Ebner, N. C. (2021). Blindspots in

- Python and Java APIs Result in Vulnerable Code. *arXiv:2103.06091 [Cs]*.  
<https://arxiv.org/abs/2103.06091>
- Brysbaert, M., & Stevens, M. (2018). *Power Analysis and Effect Size in Mixed Effects Models: A Tutorial*. 1(1), 9. <https://doi.org/10.5334/joc.10>
- Buehler, R., Griffin, D., & Ross, M. (1994). Exploring the "planning fallacy": Why people underestimate their task completion times. *Journal of Personality and Social Psychology*, 67(3), 366. <https://doi.org/10.1037/0022-3514.67.3.366>
- Buffardi, K. (2023). Cognitive Reflection in Software Verification and Testing. 2023 *IEEE/ACM 45th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, 1–10.  
<https://doi.org/10.1109/ICSE-SEET58685.2023.000006>
- Bürkner, P.-C., & Vuorre, M. (2019). Ordinal Regression Models in Psychology: A Tutorial. *Advances in Methods and Practices in Psychological Science*, 2(1), 77–101.  
<https://doi.org/10.1177/2515245918823199>
- Cacioppo, J. T., & Petty, R. E. (1982). The need for cognition. *Journal of Personality and Social Psychology*, 42(1), 116–131. <https://doi.org/10.1037/0022-3514.42.1.116>
- Caeiro-Rodríguez, M., Manso-Vázquez, M., Mikic-Fonte, F. A., Llamas-Nistal, M., Fernández-Iglesias, M. J., Tsalapatas, H., Heidmann, O., De Carvalho, C. V., Jesmin, T., Terasmaa, J., & Sørensen, L. T. (2021). Teaching Soft Skills in Engineering Education: An European Perspective. *IEEE Access*, 9, 29222–29242.  
<https://doi.org/10.1109/ACCESS.2021.3059516>
- Calitz, A., Cullen, M., & Greyling, J. (2015). *South African Alumni Perceptions of the Industry ICT Skills Requirements*.
- Camp, L. J. (2009). Mental models of privacy and security. *IEEE Technology and Society Magazine*, 28(3), 37–46. <https://doi.org/10.1109/MTS.2009.934142>
- Cappos, J., Zhuang, Y., Oliveira, D. S., Rosenthal, M., & Yeh, K.-C. (2014). Vulnerabilities as Blind Spots in Developer's Heuristic-Based Decision-Making Processes. *Proceedings of the 2014 Workshop on New Security Paradigms Workshop -*

- NSPW '14*, 53–62. <https://doi.org/10.1145/2683467.2683472>
- Capretz, L. F., & Ahmed, F. (2018). A Call to Promote Soft Skills in Software Engineering. *Psychology and Cognitive Sciences - Open Journal*, 4(1), e1–e3. <https://doi.org/10.17140/PCSOJ-4-e011>
- Carter, L. (2011). Ideas for adding soft skills education to service learning and capstone courses for computer science students. *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, 517–522. <https://doi.org/10.1145/1953163.1953312>
- Caskurlu, S., Ashby, I., & Exter, M. (2017, June). The Alignment Between Formal Education and Software Design Professionals' Needs in Industry: Faculty Perception. *2017 ASEE Annual Conference & Exposition*.
- Casler, K., Bickel, L., & Hackett, E. (2013). Separate but equal? A comparison of participants and data gathered via Amazon's MTurk, social media, and face-to-face behavioral testing. *Computers in Human Behavior*, 29(6), 2156–2160. <https://doi.org/10.1016/j.chb.2013.05.009>
- Čavojová, V., & Jurkovič, M. (2017). Comparison of experienced vs. novice teachers in cognitive reflection and rationality. *Studia Psychologica*, 59(2), 100–112. <https://doi.org/10.21909/sp.2017.02.733>
- Cerny, B. A., & Kaiser, H. F. (1977). A Study Of A Measure Of Sampling Adequacy For Factor-Analytic Correlation Matrices. *Multivariate Behavioral Research*, 12(1), 43–47. [https://doi.org/10.1207/s15327906mbr1201\\_3](https://doi.org/10.1207/s15327906mbr1201_3)
- Chai, W. J., Abd Hamid, A. I., & Abdullah, J. M. (2018). Working Memory From the Psychological and Neurosciences Perspectives: A Review. *Frontiers in Psychology*, 9. <https://www.frontiersin.org/articles/10.3389/fpsyg.2018.00401>
- Charmaz, K. (2008). Constructionism and the Grounded Theory Method. In J. A. Holstein & J. F. Gubrium (Eds.), *Handbook of Constructionist Research*. Guilford Press.
- Chattopadhyay, S., Nelson, N., Au, A., Morales, N., Sanchez, C., Pandita, R., & Sarma,

- A. (2020). A tale from the trenches: cognitive biases and software development. *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 654–665. <https://doi.org/10.1145/3377811.3380330>
- Chattopadhyay, S., Nelson, N., Au, A., Morales, N., Sanchez, C., Pandita, R., & Sarma, A. (2022). Cognitive biases in software development. *Communications of the ACM*, 65(4), 115–122. <https://doi.org/10.1145/3517217>
- Chattoraj, A. K., & Shabnam, S. (2015). Importance of Soft Skill in Business. *Anusandhanika*, 7(2), 105–110.
- Chng, S., Lu, H. Y., Kumar, A., & Yau, D. (2022). Hacker types, motivations and strategies: A comprehensive framework. *Computers in Human Behavior Reports*, 5, 100–167. <https://doi.org/10.1016/j.chbr.2022.100167>
- Chowdhury, P. D., Hallett, J., Patnaik, N., Tahaei, M., & Rashid, A. (2021). Developers Are Neither Enemies Nor Users: They Are Collaborators. *2021 IEEE Secure Development Conference (SecDev)*, 47–55. <https://doi.org/10.1109/SecDev51306.2021.00023>
- Cicirello, V. A. (2017). Student developed computer science educational tools as software engineering course projects. *Journal of Computing Sciences in Colleges*, 32(3), 55–61.
- Coffeng, T., van Steenberg, E. F., de Vries, F., Steffens, N. K., & Ellemers, N. (2023). Reflective and decisive supervision: The role of participative leadership and team climate in joint decision-making. *Regulation & Governance*, 17(1), 290–309. <https://doi.org/10.1111/rego.12449>
- Cortázar, C., Goñi, I., Ortiz, A., & Nussbaum, M. (2024). Are Professional Skills Learnable? Beliefs and Expectations Among Computing Graduates. *ACM Transactions on Computing Education*. <https://doi.org/10.1145/3641551>
- Craig, M., Conrad, P., Lynch, D., Lee, N., & Anthony, L. (2018). Listening to early career software developers. *Journal of Computing Sciences in Colleges*, 33(4), 138–149.
- Cranor, L. F. (2005). *Security and Usability: Designing Secure Systems That People Can*

Use. O'Reilly Media, Inc.

- Crichton, W., Agrawala, M., & Hanrahan, P. (2021). The Role of Working Memory in Program Tracing. *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 1–13. <https://doi.org/10.1145/3411764.3445257>
- Croskerry, P. (2003). Cognitive forcing strategies in clinical decisionmaking. *Annals of Emergency Medicine*, 41(1), 110–120. <https://doi.org/10.1067/mem.2003.22>
- Damnjanović, K., Novković, V., Pavlović, I., Ilić, S., & Pantelić, S. (2019). A Cue for Rational Reasoning: Introducing a Reference Point in Cognitive Reflection Tasks. *Europe's Journal of Psychology*, 15(1), 25–40. <https://doi.org/10.5964/ejop.v15i1.1701>
- Danilova, A., Naiakshina, A., Rasgauski, A., & Smith, M. (2021). *Code Reviewing as Methodology for Online Security Studies with Developers – A Case Study with Freelancers on Password Storage*. 21.
- Danilova, A., Naiakshina, A., & Smith, M. (2020). One Size Does Not Fit All: A Grounded Theory and Online Survey Study of Developer Preferences for Security Warning Types. *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, 136–148.
- Dash, N., & Gladwin, H. (2007). Evacuation Decision Making and Behavioral Responses: Individual and Household. *Natural Hazards Review*, 8(3), 69–77. [https://doi.org/10.1061/\(ASCE\)1527-6988\(2007\)8:3\(69\)](https://doi.org/10.1061/(ASCE)1527-6988(2007)8:3(69))
- De Neys, W., Rossi, S., & Houdé, O. (2013). Bats, balls, and substitution sensitivity: cognitive misers are no happy fools. *Psychonomic Bulletin & Review*, 20(2), 269–273. <https://doi.org/10.3758/s13423-013-0384-5>
- de Souza, C. R. B., Figueira Filho, F., Miranda, M., Ferreira, R. P., Treude, C., & Singer, L. (2016). The Social Side of Software Platform Ecosystems. *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 3204–3214. <https://doi.org/10.1145/2858036.2858431>
- De Wit, J., Pieters, W., Jansen, S., & Van Gelder, P. (2021). Biases in security risk management: Do security professionals follow prospect theory in their decisions?

- Journal of Integrated Security and Safety Science*.  
<https://doi.org/10.18757/JISSS.2021.1.5700>
- DeLiema, D., Dahn, M., Flood, V., Asuncion, A., Abrahamson, D., Enyedy, N., & Steen, F. (2019). Debugging as a Context for Fostering Reflection on Critical Thinking and Emotion. In *Deeper Learning, Dialogic Learning, and Critical Thinking* (pp. 209–228). Routledge.
- Destefanis, G., Ortu, M., Counsell, S., Swift, S., Tonelli, R., & Marchesi, M. (2017). On the randomness and seasonality of affective metrics for software development. *Proceedings of the Symposium on Applied Computing*, 1266–1271.  
<https://doi.org/10.1145/3019612.3019786>
- Djulgovic, B., Hozo, I., Beckstead, J., Tsalatsanis, A., & Pauker, S. G. (2012). Dual processing model of medical decision-making. *BMC Medical Informatics and Decision Making*, 12(1), 94. <https://doi.org/10.1186/1472-6947-12-94>
- Donthu, N., Kumar, S., Mukherjee, D., Pandey, N., & Lim, W. M. (2021). How to conduct a bibliometric analysis: An overview and guidelines. *Journal of Business Research*, 133, 285–296. <https://doi.org/10.1016/j.jbusres.2021.04.070>
- Dourish, P. (2001). *Where the Action is: The Foundations of Embodied Interaction*. MIT Press.
- Duijf, H. (2021). Should one trust experts? *Synthese*, 199(3), 9289–9312.  
<https://doi.org/10.1007/s11229-021-03203-7>
- Eck, N. van, & Waltman, L. (2009). Software survey: VOSviewer, a computer program for bibliometric mapping. *Scientometrics*, 84(2), 523–538.  
<https://doi.org/10.1007/s11192-009-0146-3>
- Egele, M., Brumley, D., Fratantonio, Y., & Kruegel, C. (2013). An empirical study of cryptographic misuse in android applications. *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security - CCS '13*, 73–84.  
<https://doi.org/10.1145/2508859.2516693>
- Egelman, S., & Peer, E. (2015). Scaling the Security Wall: Developing a Security

- Behavior Intentions Scale (SeBIS). *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, 2873–2882.  
<https://doi.org/10.1145/2702123.2702249>
- Elder, S., Zahan, N., Shu, R., Metro, M., Kozarev, V., Menzies, T., & Williams, L. (2022). Do I really need all this work to find vulnerabilities? *Empirical Software Engineering*, 27(6), 154. <https://doi.org/10.1007/s10664-022-10179-6>
- Ellemers, N., Spears, R., & Doosje, B. (2002). Self and Social Identity. *Annual Review of Psychology*, 53(1), 161–186. <https://doi.org/10.1146/annurev.psych.53.100901.135228>
- English, R., & Hayes, A. (2022). Towards Integrated Graduate Skills for UK Computing Science Students. *Proceedings of the 2022 Conference on United Kingdom & Ireland Computing Education Research*, 1–7. <https://doi.org/10.1145/3555009.3555018>
- Ennis, R. H. (1987). A taxonomy of critical thinking dispositions and abilities. In *Teaching thinking skills: Theory and practice* (pp. 9–26). W H Freeman/Times Books/ Henry Holt & Co.
- Evans, J. St. B. T. (1984). Heuristic and analytic processes in reasoning. *British Journal of Psychology*, 75(4), 451–468. <https://doi.org/10.1111/j.2044-8295.1984.tb01915.x>
- Evans, J. St. B. T. (2003). In two minds: dual-process accounts of reasoning. *Trends in Cognitive Sciences*, 7(10), 454–459. <https://doi.org/10.1016/j.tics.2003.08.012>
- Evans, J. St. B. T. (2010a). *Thinking twice: Two minds in one brain* (pp. viii, 240). Oxford University Press.
- Evans, J. St. B. T. (2010b). Intuition and Reasoning: A Dual-Process Perspective. *Psychological Inquiry*, 21(4), 313–326. <https://doi.org/10.1080/1047840X.2010.521057>
- Evans, J. St. B. T., & Stanovich, K. E. (2013). Dual-Process Theories of Higher Cognition: Advancing the Debate. *Perspectives on Psychological Science*, 8(3), 223–241. <https://doi.org/10.1177/1745691612460685>
- Fagerholm, F., Felderer, M., Fucci, D., Unterkalmsteiner, M., Marculescu, B., Martini, M., Tengberg, L. G. W., Feldt, R., Lehtelä, B., Nagyvárad, B., & Khattak, J. (2022). Cognition in Software Engineering: A Taxonomy and Survey of a Half-Century of



- Research. *ACM Computing Surveys*, 54(11s), 1–36. <https://doi.org/10.1145/3508359>
- Fareri, S., Melluso, N., Chiarello, F., & Fantoni, G. (2021). SkillNER: Mining and Mapping Soft Skills from any Text. *Expert Systems with Applications*, 184, 115544. <https://doi.org/10.1016/j.eswa.2021.115544>
- Feliciano, J. (2015). *Towards a Collaborative Learning Platform: The Use of GitHub in Computer Science and Software Engineering Courses* [Thesis].
- Fiesler, C., Garrett, N., & Beard, N. (2020). What Do We Teach When We Teach Tech Ethics? A Syllabi Analysis. *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 289–295. <https://doi.org/10.1145/3328778.3366825>
- Finch, W. H. (2020). Using Fit Statistic Differences to Determine the Optimal Number of Factors to Retain in an Exploratory Factor Analysis. *Educational and Psychological Measurement*, 80(2), 217–241. <https://doi.org/10.1177/0013164419865769>
- Fischer, F., Bottinger, K., Xiao, H., Stransky, C., Acar, Y., Backes, M., & Fahl, S. (2017). *Stack Overflow Considered Harmful? The Impact of Copy&Paste on Android Application Security*. 121–136. <https://doi.org/10.1109/sp.2017.31>
- Fischer, J. M., & Ravizza, M. (1998). *Responsibility and Control: A Theory of Moral Responsibility*. Cambridge University Press.
- Florea, R., & Stray, V. (2018). Software Tester, We Want to Hire You! an Analysis of the Demand for Soft Skills. In J. Garbajosa, X. Wang, & A. Aguiar (Eds.), *Agile Processes in Software Engineering and Extreme Programming* (pp. 54–67). Springer International Publishing. [https://doi.org/10.1007/978-3-319-91602-6\\_4](https://doi.org/10.1007/978-3-319-91602-6_4)
- Frederick, S. (2005). Cognitive reflection and decision making. *Journal of Economic Perspectives*, 19(4), 25–42. <https://doi.org/10.1257/089533005775196732>
- Frik, A., Kim, J., Sanchez, J. R., & Ma, J. (2022). Users' Expectations About and Use of Smartphone Privacy and Security Settings. *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 1–24. <https://doi.org/10.1145/3491102.3517504>
- Furnell, S. (2021). The cybersecurity workforce and skills. *Computers & Security*, 100,

102080. <https://doi.org/10.1016/j.cose.2020.102080>
- Furnell, S., & Bishop, M. (2020). Addressing cyber security skills: The spectrum, not the silo. *Computer Fraud & Security*, 2020(2), 6–11.  
[https://doi.org/10.1016/S1361-3723\(20\)30017-8](https://doi.org/10.1016/S1361-3723(20)30017-8)
- Fussell, S. R., & Krauss, R. M. (1992). Coordination of knowledge in communication: Effects of speakers' assumptions about what others know. *Journal of Personality and Social Psychology*, 62(3), 378–391. <https://doi.org/10.1037/0022-3514.62.3.378>
- Galhotra, S., Brun, Y., & Meliou, A. (2017). Fairness Testing: Testing Software for Discrimination. *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 498–510. <https://doi.org/10.1145/3106237.3106277>
- Gallagher, K. P., Kaiser, K. M., Simon, J. C., Beath, C. M., & Goles, T. (2010). The requisite variety of skills for IT professionals. *Communications of the ACM*, 53(6), 144–148. <https://doi.org/10.1145/1743546.1743584>
- Galster, M., Mitrovic, A., Malinen, S., & Holland, J. (2022). What Soft Skills Does the Software Industry \*Really\* Want? An Exploratory Study of Software Positions in New Zealand. *Proceedings of the 16th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, 272–282.  
<https://doi.org/10.1145/3544902.3546247>
- Garousi, V., Giray, G., Tuzun, E., Catal, C., & Felderer, M. (2020). Closing the Gap Between Software Engineering Education and Industrial Needs. *IEEE Software*, 37(2), 68–77. <https://doi.org/10.1109/MS.2018.2880823>
- Gavidia-Calderon, C., Bennaceur, A., Lopez, T., Kordoni, A., Levine, M., & Nuseibeh, B. (2023). Meet your Maker: A Social Identity Analysis of Robotics Software Engineering. *Proceedings of the First International Symposium on Trustworthy Autonomous Systems*, 1–5. <https://doi.org/10.1145/3597512.3600206>
- Gibbs, G. (2007). *Analyzing Qualitative Data*. SAGE Publications, Ltd.  
<https://doi.org/10.4135/9781849208574>
- Gibert, A., Tozer, W. C., & Westoby, M. (2017). Teamwork, Soft Skills, and Research

- Training. *Trends in Ecology & Evolution*, 32(2), 81–84.  
<https://doi.org/10.1016/j.tree.2016.11.004>
- Gigerenzer, G. (2002). *Adaptive Thinking: Rationality in the Real World*. Oxford University Press, USA.
- Gigerenzer, G. (2008). Why Heuristics Work. *Perspectives on Psychological Science*, 3(1), 20–29. <https://doi.org/10.1111/j.1745-6916.2008.00058.x>
- Gigerenzer, G. (2015). *Simply Rational: Decision Making in the Real World*. Oxford University Press.
- Gigerenzer, G., & Gaissmaier, W. (2011). Heuristic Decision Making. *Annual Review of Psychology*, 62(1), 451–482. <https://doi.org/10.1146/annurev-psych-120709-145346>
- Gigerenzer, G., & Todd, P. M. (1999). *Simple heuristics that make us smart* (pp. xv, 416). Oxford University Press.
- Gillard, S. (2009). Soft Skills and Technical Expertise of Effective Project Managers. *Issue in Informing Science and Information Technology*, 6(1), 723–729.
- Giraffa, L. M. M., Moraes, M. C., & Uden, L. (2014). Teaching Object-Oriented Programming in First-Year Undergraduate Courses Supported By Virtual Classrooms. In L. Uden, Y.-H. Tao, H.-C. Yang, & I.-H. Ting (Eds.), *The 2nd International Workshop on Learning Technology for Education in Cloud* (pp. 15–26). Springer Netherlands. [https://doi.org/10.1007/978-94-007-7308-0\\_2](https://doi.org/10.1007/978-94-007-7308-0_2)
- Glaser, M., & Walther, T. (2014). *Run, Walk, or Buy? Financial Literacy, Dual-Process Theory, and Investment Behavior* ({{SSRN Scholarly Paper}} No. 2167270). <https://doi.org/10.2139/ssrn.2167270>
- Goldhaber, G. M. (1974). Organizational communication. (*No Title*). <https://cir.nii.ac.jp/crid/1130282272132859776>
- Gold-Veerkamp, C. (2019, January). A Software Engineer’s Competencies: Undergraduate Preconceptions in Contrast to Teaching Intentions. *Proceedings of the 52nd Hawaii International Conference on System Sciences*. <https://doi.org/10.24251/HICSS.2019.937>

- Gonçales, L. J., Farias, K., & da Silva, B. C. (2021). Measuring the cognitive load of software developers: An extended Systematic Mapping Study. *Information and Software Technology*, 136, 106563. <https://doi.org/10.1016/j.infsof.2021.106563>
- Gonçalves, P. W., Fregnan, E., Baum, T., Schneider, K., & Bacchelli, A. (2020). Do Explicit Review Strategies Improve Code Review Performance? *Proceedings of the 17th International Conference on Mining Software Repositories*, 606–610. <https://doi.org/10.1145/3379597.3387509>
- González-Morales, D., Antonio, L. M. M. de, & García, J. L. R. (2011). Teaching “soft” skills in Software Engineering. *2011 IEEE Global Engineering Education Conference (EDUCON)*, 630–637. <https://doi.org/10.1109/EDUCON.2011.5773204>
- Gorski, P. L., Acar, Y., Lo Iacono, L., & Fahl, S. (2020). Listen to Developers! A Participatory Design Study on Security Warnings for Cryptographic APIs. *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 1–13. <https://doi.org/10.1145/3313831.3376142>
- Gorski, P. L., Lo Iacono, L., Wermke, D., Stransky, C., Möller, S., Acar, Y., & Fahl, S. (2018). Developers Deserve Security Warnings, Too: On the Effect of Integrated Security Advice on Cryptographic API Misuse. *Fourteenth Symposium on Usable Privacy and Security*, 265–281.
- Gotterbarn, D. (2001). Informatics and professional responsibility. *Science and Engineering Ethics*, 7(2), 221–230. <https://doi.org/10.1007/s11948-001-0043-5>
- Graff, M., & Wyk, K. R. V. (2003). *Secure Coding: Principles and Practices*. O'Reilly Media, Inc.
- Graham, C. M., & Lu, Y. (2022). Skills Expectations in Cybersecurity: Semantic Network Analysis of Job Advertisements. *Journal of Computer Information Systems*, 0(0), 1–13. <https://doi.org/10.1080/08874417.2022.2115954>
- Gratian, M., Bandi, S., Cukier, M., Dykstra, J., & Ginther, A. (2018). Correlating human traits and cyber security behavior intentions. *Computers & Security*, 73, 345–358. <https://doi.org/10.1016/j.cose.2017.11.015>

- Green, M., & Smith, M. (2016). Developers are Not the Enemy!: The Need for Usable Security APIs. *IEEE Security Privacy*, 14(5), 40–46.  
<https://doi.org/10.1109/MSP.2016.111>
- Greenacre, M. (2017). *Correspondence Analysis in Practice*. CRC Press.
- Groeneveld, W., Becker, B. A., & Vennekens, J. (2020a). Soft Skills: What do Computing Program Syllabi Reveal About Non-Technical Expectations of Undergraduate Students? *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, 287–293. <https://doi.org/10.1145/3341525.3387396>
- Groeneveld, W., Jacobs, H., Vennekens, J., & Aerts, K. (2020b). Non-cognitive Abilities of Exceptional Software Engineers: A Delphi Study. *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 1096–1102.  
<https://doi.org/10.1145/3328778.3366811>
- Groeneveld, W., Vennekens, J., & Aerts, K. (2019). *Software Engineering Education Beyond the Technical: A Systematic Literature Review* (No. arXiv:1910.09865). arXiv.  
<https://doi.org/10.48550/arXiv.1910.09865>
- Grosz, B. J., Grant, D. G., Vredenburgh, K., Behrends, J., Hu, L., Simmons, A., & Waldo, J. (2019). Embedded EthiCS: Integrating ethics across CS education. *Communications of the ACM*, 62(8), 54–61. <https://doi.org/10.1145/3330794>
- Guerra-Báez, S. P. (2019). A panoramic review of soft skills training in university students. *Psicologia Escolar e Educacional*, 23, 1–10.  
<https://doi.org/10.1590/2175-35392019016464>
- Gutmann, P. (2002). Lessons Learned in Implementing and Deploying Crypto Software. *In Proc. USENIX Security Symp*, 315–325.
- Hagoort, P. (2023). The language marker hypothesis. *Cognition*, 230, 105252.  
<https://doi.org/10.1016/j.cognition.2022.105252>
- Hallett, J., Patnaik, N., Shreeve, B., & Rashid, A. (2021). “Do this! Do that!, and Nothing will Happen” Do Specifications Lead to Securely Stored Passwords? *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 486–498.

<https://doi.org/10.1109/ICSE43902.2021.00053>

Hamari, J. (2017). Do badges increase user activity? A field experiment on the effects of gamification. *Computers in Human Behavior*, 71, 469–478.

<https://doi.org/10.1016/j.chb.2015.03.036>

Haney, J. M., & Lutters, W. G. (2017). Skills and Characteristics of Successful Cybersecurity Advocates. *Thirteenth Symposium on Usable Privacy and Security*.

Haney, J. M., Theofanos, M. F., Acar, Y., & Prettyman, S. S. (2018). "We make it a big deal in the company": Security Mindsets in Organizations that Develop Cryptographic Products. *USENIX Conference on Usable Privacy and Security*, 17.

<https://doi.org/10.5555/3291228.3291257>

Haslam, S. A. (2012). The Social Identity Approach. In *Psychology in Organizations: The Social Identity Approach* (2nd ed., pp. 17–39). SAGE Publications Ltd.

<https://doi.org/10.4135/9781446278819>

Haslam, S. A., & Parkinson, B. (2005). Pulling together or pulling apart?: Towards organic pluralism in social psychology. *The Psychologist*, 18(9), 550–554.

Haslam, S. A., Powell, C., & Turner, J. (2000). Social Identity, Self-categorization, and Work Motivation: Rethinking the Contribution of the Group to Positive and Sustainable Organisational Outcomes. *Applied Psychology*, 49(3), 319–339.

<https://doi.org/10.1111/1464-0597.00018>

Hazzan, O., & Har-Shai, G. (2013). Teaching computer science soft skills as soft concepts.

*Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, 59–64. <https://doi.org/10.1145/2445196.2445219>

Helgesson, D. (2023). *Exploring cognitive waste and cognitive load in software development - a grounded theory*.

Hertel, G., Niedner, S., & Herrmann, S. (2003). Motivation of software developers in Open Source projects: An Internet-based survey of contributors to the Linux kernel. *Research Policy*, 32(7), 1159–1177. [https://doi.org/10.1016/S0048-7333\(03\)00047-7](https://doi.org/10.1016/S0048-7333(03)00047-7)

Hewner, M. (2014). How CS undergraduates make course choices. *Proceedings of the*

- Tenth Annual Conference on International Computing Education Research*, 115–122.  
<https://doi.org/10.1145/2632320.2632345>
- Hewner, M. (2013). Undergraduate conceptions of the field of computer science.  
*Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research*, 107–114. <https://doi.org/10.1145/2493394.2493414>
- Hjeij, M., & Vilks, A. (2023). A brief history of heuristics: how did research on heuristics evolve? *Humanities and Social Sciences Communications*, 10(1), 1–15.  
<https://doi.org/10.1057/s41599-023-01542-z>
- Hoffmann, A. O. I., Post, T., & Pennings, J. M. E. (2015). How Investor Perceptions Drive Actual Trading and Risk-Taking Behavior. *Journal of Behavioral Finance*, 16(1), 94–103. <https://doi.org/10.1080/15427560.2015.1000332>
- Hogg, M., & Vaughan, G. (2017). *Social Psychology*. Pearson Education, Limited.
- Hoover, J. D., & Healy, A. F. (2019). The bat-and-ball problem: Stronger evidence in support of a conscious error process. *Decision*, 6, 369–380.  
<https://doi.org/10.1037/dec0000107>
- Hoover, J. D., & Healy, A. F. (2021). The bat-and-ball problem: a word-problem debiasing approach. *Thinking & Reasoning*, 27(4), 567–598.  
<https://doi.org/10.1080/13546783.2021.1878473>
- Huang, J., Borges, N., Bugiel, S., & Backes, M. (2019). Up-To-Crash: Evaluating Third-Party Library Updatability on Android. *2019 IEEE European Symposium on Security and Privacy (EuroS P)*, 15–30. <https://doi.org/10.1109/EuroSP.2019.00012>
- Ifinedo, P. (2012). Understanding information systems security policy compliance: An integration of the theory of planned behavior and the protection motivation theory. *Computers & Security*, 31(1), 83–95. <https://doi.org/10.1016/j.cose.2011.10.007>
- Iniesto, F., Sargent, J., Rienties, B., Llorens, A., Adam, A., Herodotou, C., Ferguson, R., & Muccini, H. (2021). When industry meets Education 4.0: What do Computer Science companies need from Higher Education? *Ninth International Conference on Technological Ecosystems for Enhancing Multiculturality (TEEM'21)*, 367–372.

<https://doi.org/10.1145/3486011.3486475>

Itani, M., & Srour, I. (2016). Engineering Students' Perceptions of Soft Skills, Industry Expectations, and Career Aspirations. *Journal of Professional Issues in Engineering Education and Practice*, 142(1), 04015005.

[https://doi.org/10.1061/\(ASCE\)EI.1943-5541.0000247](https://doi.org/10.1061/(ASCE)EI.1943-5541.0000247)

Ivins, J., Von Kinsky, B. R., Cooper, D., & Robey, M. (2006). Software Engineers and Engineering: A Survey of Undergraduate Preconceptions. *Proceedings. Frontiers in Education. 36th Annual Conference*, 6–11. <https://doi.org/10.1109/FIE.2006.322364>

Ivory, M. (2022). The Soft Skills of Software Learning Development: the Psychological Dimensions of Computing and Security Behaviours. *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering 2022*, 317–322.

<https://doi.org/10.1145/3530019.3535344>

Ivory, M., Sturdee, M., Towse, J., Levine, M., & Nuseibeh, B. (2023 (in review)). Can you hear the ROAR of software security? How Responsibility, Optimism And Risk shape developers' security perceptions. *Empirical Software Engineering*.

<https://doi.org/10.31234/osf.io/pexvz>

Ivory, M., Towse, J., Sturdee, M., Levine, M., & Nuseibeh, B. (2023a). What's in an undergraduate Computer Science Degree; Alumni perceptions about soft skills in careers. *Transactions on Computing Education*.

Ivory, M., Towse, J., Sturdee, M., Levine, M., & Nuseibeh, B. (2023b). *Software Insecurity as Cognitive Blindspots; Security Vulnerabilities through a Cognitive Psychology Lens*. <https://doi.org/10.17605/OSF.IO/CE78G>

Ivory, M., Towse, J., Sturdee, M., Levine, M., & Nuseibeh, B. (2023c). Recognizing the Known Unknowns; the Interaction Between Reflective Thinking and Optimism for Uncertainty Among Software Developer's Security Perceptions. *Technology, Mind, and Behavior*, 4(3: Winter 2023). <https://doi.org/10.1037/tmb0000122>

Jackendoff, R. S. (2009). *Language, Consciousness, Culture: Essays on Mental Structure*. MIT Press.



- Janssen, E. M., Meulendijks, W., Mainhard, T., Verkoeijen, P. P. J. L., Heijltjes, A. E. G., van Peppen, L. M., & van Gog, T. (2019). Identifying characteristics associated with higher education teachers' Cognitive Reflection Test performance and their attitudes towards teaching critical thinking. *Teaching and Teacher Education*, *84*, 139–149. <https://doi.org/10.1016/j.tate.2019.05.008>
- Jarrold, C., & Towse, J. (2006). Individual differences in working memory. *Neuroscience*, *139*(1), 39–50. <https://doi.org/10.1016/j.neuroscience.2005.07.002>
- Jia, J., Chen, Z., & Du, X. (2017). Understanding Soft Skills Requirements for Mobile Applications Developers. *2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, *1*, 108–115. <https://doi.org/10.1109/CSE-EUC.2017.29>
- John, M., Maurer, F., & Tessem, B. (2005). Human and social factors of software engineering: workshop summary. *ACM SIGSOFT Software Engineering Notes*, *30*(4), 1–6. <https://doi.org/10.1145/1082983.1083000>
- Johnson, R., & Waterfield, J. (2004). Making words count: the value of qualitative research. *Physiotherapy Research International: The Journal for Researchers and Clinicians in Physical Therapy*, *9*(3), 121–131. <https://doi.org/10.1002/pri.312>
- Johnson-Laird, P. N. (1983). *Mental Models: Towards a Cognitive Science of Language, Inference, and Consciousness*. Harvard University Press.
- Johnson-Laird, P. N. (2010). Mental models and human reasoning. *Proceedings of the National Academy of Sciences*, *107*(43), 18243–18250. <https://doi.org/10.1073/pnas.1012933107>
- Jones, H. S., Towse, J., Race, N., & Harrison, T. (2019). Email fraud: The search for psychological predictors of susceptibility. *PLOS ONE*, *14*(1), e0209684. <https://doi.org/10.1371/journal.pone.0209684>
- Jones, K. S., Namin, A. S., & Armstrong, M. E. (2018). The Core Cyber-Defense Knowledge, Skills, and Abilities That Cybersecurity Students Should Learn in School: Results from Interviews with Cybersecurity Professionals. *ACM Transactions on*

- Computing Education*, 18(3), 11:1–11:12. <https://doi.org/10.1145/3152893>
- Jones, M., Baldi, C., Phillips, C., & Waikar, A. (2016). The hard truth about soft skills: What recruiters look for in business graduates. *College Student Journal*, 50(3), 422–429.
- Jones, T., Baxter, M., & Khanduja, V. (2013). A quick guide to survey research. *Annals of The Royal College of Surgeons of England*, 95(1), 5–7. <https://doi.org/10.1308/003588413X13511609956372>
- Joseph, D., Ang, S., Chang, R. H. L., & Slaughter, S. A. (2010). Practical intelligence in IT: Assessing soft skills of IT professionals. *Communications of the ACM*, 53(2), 149–154. <https://doi.org/10.1145/1646353.1646391>
- Kahneman, D. (2011). *Thinking, Fast and Slow*. Farrar, Straus and Giroux.
- Kahneman, D., & Frederick, S. (2002). Representativeness Revisited: Attribute Substitution in Intuitive Judgment. In T. Gilovich, D. Griffin, & D. Kahneman (Eds.), *Heuristics and Biases* (1st ed., pp. 49–81). Cambridge University Press. <https://doi.org/10.1017/CBO9780511808098.004>
- Kahneman, D., & Klein, G. (2009). Conditions for intuitive expertise: A failure to disagree. *American Psychologist*, 64, 515–526. <https://doi.org/10.1037/a0016755>
- Kahneman, D., Slovic, P., & Tversky, A. (Eds.). (1974). *Judgment under uncertainty: Heuristics and biases* (1st ed.). Cambridge University Press.
- Kahneman, D., & Tversky, A. (1979). Prospect Theory: An Analysis of Decision under Risk. *Econometrica*, 47(2), 263–291. <https://doi.org/10.2307/1914185>
- Kahneman, D., & Tversky, A. (1982). Intuitive prediction: Biases and corrective procedures. In A. Tversky, D. Kahneman, & P. Slovic (Eds.), *Judgment under Uncertainty: Heuristics and Biases* (pp. 414–421). Cambridge University Press. <https://doi.org/10.1017/CBO9780511809477.031>
- Karimi, H., & Pina, A. (2021). Strategically Addressing the Soft Skills Gap Among STEM Undergraduates. *Journal of Research in STEM Education*, 7(1), 21–46. <https://doi.org/10.51355/jstem.2021.99>

- Keil, M., Wallace, L., Turk, D., Dixon-Randall, G., & Nulden, U. (2000). An Investigation of Risk Perception and Risk Propensity on the Decision to Continue a Software Development Project. *The Journal of Systems and Software*, 53, 145–157.
- Kelle, U. (2006). Combining qualitative and quantitative methods in research practice: purposes and advantages. *Qualitative Research in Psychology*, 3(4), 293–311.  
<https://doi.org/10.1177/1478088706070839>
- Kina, K., Tsunoda, M., Hata, H., Tamada, H., & Igaki, H. (2016). Analyzing the Decision Criteria of Software Developers Based on Prospect Theory. *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 1, 644–648. <https://doi.org/10.1109/SANER.2016.115>
- Kirlappos, I., Beaument, A., & Sasse, M. A. (2013). “Comply or Die” Is Dead: Long Live Security-Aware Principal Agents. In D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, A. A. Adams, M. Brenner, & M. Smith (Eds.), *Financial Cryptography and Data Security* (Vol. 7862, pp. 70–82). Springer Berlin Heidelberg.  
[https://doi.org/10.1007/978-3-642-41320-9\\_5](https://doi.org/10.1007/978-3-642-41320-9_5)
- Kl nder, J., Schneider, K., Kortum, F., Straube, J., Handke, L., & Kauffeld, S. (2016). Communication in Teams - An Expression of Social Conflicts. In C. Bogdan, J. Gulliksen, S. Sauer, P. Forbrig, M. Winckler, C. Johnson, P. Palanque, R. Bernhaupt, & F. Kis (Eds.), *Human-Centered and Error-Resilient Systems Development* (pp. 111–129). Springer International Publishing.  
[https://doi.org/10.1007/978-3-319-44902-9\\_8](https://doi.org/10.1007/978-3-319-44902-9_8)
- Knauff, M., & Wolf, A. G. (2010). Complex cognition: The science of human reasoning, problem-solving, and decision-making. *Cognitive Processing*, 11(2), 99–102.  
<https://doi.org/10.1007/s10339-010-0362-z>
- Knutas, A., Hynninen, T., & Hujala, M. (2021). To Get Good Student Ratings should you only Teach Programming Courses? Investigation and Implications of Student

- Evaluations of Teaching in a Software Engineering Context. *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, 253–260.  
<https://doi.org/10.1109/ICSE-SEET52601.2021.00035>
- Kraiger, K. (2001). Industrial–Organizational Psychology: Science and Practice. In N. J. Smelser & P. B. Baltes (Eds.), *International Encyclopedia of the Social & Behavioral Sciences* (pp. 7367–7371). Pergamon.  
<https://doi.org/10.1016/B0-08-043076-7/01393-0>
- Kramer, R. M., Pommerenke, P., & Newton, E. (1993). The Social Context of Negotiation: Effects of Social Identity and Interpersonal Accountability on Negotiator Decision Making. *Journal of Conflict Resolution*, 37(4), 633–654.  
<https://doi.org/10.1177/0022002793037004003>
- Kula, R. G., German, D. M., Ouni, A., Ishio, T., & Inoue, K. (2018). Do developers update their library dependencies?: An empirical study on the impact of security advisories on library migration. *Empirical Software Engineering*, 23(1), 384–417.  
<https://doi.org/10.1007/s10664-017-9521-5>
- Kumle, L., Võ, M. L.-H., & Draschkow, D. (2021). Estimating power in (generalized) linear mixed models: An open introduction and tutorial in R. *Behavior Research Methods*, 53(6), 2528–2543. <https://doi.org/10.3758/s13428-021-01546-0>
- Kunda, Z. (1990). The case for motivated reasoning. *Psychological Bulletin*, 108, 480–498.  
<https://doi.org/10.1037/0033-2909.108.3.480>
- Lapinski, M. K., & Rimal, R. N. (2005). An Explication of Social Norms. *Communication Theory*, 15(2), 127–147. <https://doi.org/10.1111/j.1468-2885.2005.tb00329.x>
- Lembke, S., & Wilson, M. G. (1998). Putting the “Team” into Teamwork: Alternative Theoretical Contributions for Contemporary Management Practice. *Human Relations*, 51(7), 927–944. <https://doi.org/10.1177/001872679805100704>
- Lenberg, P., Feldt, R., & Wallgren, L. G. (2015). Behavioral software engineering: A definition and systematic literature review. *Journal of Systems and Software*, 107,

- 15–37. <https://doi.org/10.1016/j.jss.2015.04.084>
- Lenberg, P., Feldt, R., & Wallgren, L.-G. (2014). Towards a behavioral software engineering. *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering*, 48–55.  
<https://doi.org/10.1145/2593702.2593711>
- Leung, L. (2015). Validity, reliability, and generalizability in qualitative research. *Journal of Family Medicine and Primary Care*, 4(3), 324–327.  
<https://doi.org/10.4103/2249-4863.161306>
- Levine, M., Prosser, A., Evans, D., & Reicher, S. (2005). Identity and emergency intervention: How social group membership and inclusiveness of group boundaries shape helping behavior. *Personality & Social Psychology Bulletin*, 31(4), 443–453.  
<https://doi.org/10.1177/0146167204271651>
- Levy, J. S. (1992). An Introduction to Prospect Theory. *Political Psychology*, 13(2), 171–186. <http://www.jstor.org/stable/3791677>
- Lewis, C., Jackson, M. H., & Waite, W. M. (2010). Student and faculty attitudes and beliefs about computer science. *Communications of the ACM*, 53(5), 78–85.  
<https://doi.org/10.1145/1735223.1735244>
- Liberali, J. M., Reyna, V. F., Furlan, S., Stein, L. M., & Pardo, S. T. (2012). Individual Differences in Numeracy and Cognitive Reflection, with Implications for Biases and Fallacies in Probability Judgment. *Journal of Behavioral Decision Making*, 25(4), 361–381. <https://doi.org/10.1002/bdm.752>
- Licorish, S. A., Galster, M., Kapitsaki, G. M., & Tahir, A. (2022). Understanding students' software development projects: Effort, performance, satisfaction, skills and their relation to the adequacy of outcomes developed. *Journal of Systems and Software*, 186, 111156. <https://doi.org/10.1016/j.jss.2021.111156>
- Liebenberg, J., Huisman, M., & Mentz, E. (2014). Knowledge and Skills Requirements for Software Developer Students. *International Journal of Social, Behavioral, Educational, Economic, Business and Industrial Engineering*, 8(8), 6.

- Lincoln, Y. S., & Guba, E. G. (1985). *Naturalistic Inquiry*. SAGE.
- Lingard, R., & Barkataki, S. (2011). Teaching teamwork in engineering and computer science. *2011 Frontiers in Education Conference (FIE)*, F1C-1-F1C-5.  
<https://doi.org/10.1109/FIE.2011.6143000>
- Lopez, T., Sharp, H., Tun, T., Bandara, A. K., Levine, M., & Nuseibeh, B. (2019a). "Hopefully We Are Mostly Secure": Views on Secure Code in Professional Practice. *12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 61–68. <https://doi.org/10.1109/chase.2019.00023>
- Lopez, T., Sharp, H., Tun, T., Bandara, A. K., Levine, M., & Nuseibeh, B. (2022). Security Responses in Software Development. *ACM Transactions on Software Engineering and Methodology*. <https://doi.org/10.1145/3563211>
- Lopez, T., Sharp, H., Tun, T., Bandara, A., Levine, M., & Nuseibeh, B. (2019b). Talking About Security with Professional Developers. *2019 IEEE/ACM Joint 7th International Workshop on Conducting Empirical Studies in Industry (CESI) and 6th International Workshop on Software Engineering Research and Industrial Practice (SER IP)*, 34–40. <https://doi.org/10.1109/CESSER-IP.2019.00014>
- Lopez, T., Tun, T. T., Bandara, A. K., Levine, M., Nuseibeh, B., & Sharp, H. (2020a). Taking the Middle Path: Learning About Security Through Online Social Interaction. *IEEE Software*, 37(1), 25–30. <https://doi.org/10.1109/MS.2019.2945300>
- Lopez, T., Tun, T. T., Bandara, A., Levine, M., Nuseibeh, B., & Sharp, H. (2018). An investigation of security conversations in stack overflow: perceptions of security and community involvement. *Proceedings of the 1st International Workshop on Security Awareness from Design to Deployment*, 26–32.  
<https://doi.org/10.1145/3194707.3194713>
- Lopez, T., Tun, T., Bandara, A., Mark, L., Nuseibeh, B., & Sharp, H. (2019c). An Anatomy of Security Conversations in Stack Overflow. *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*, 31–40. <https://doi.org/10.1109/ICSE-SEIS.2019.00012>

- Lopez, T., Weir, C., Cooper, H., Tun, T., Bandara, A., Levine, M., Nuseibeh, B., & Sharp, H. (2020b). *Motivating Jenny Developer Security Toolkit*.  
<https://doi.org/10.21954/ou.rd.c.4957223.v1>
- Loser, K.-U., & Degeling, M. (2014). Security and Privacy as Hygiene Factors of Developer Behavior in Small and Agile Teams. *ICT and Society*, 255–265.  
[https://doi.org/10.1007/978-3-662-44208-1\\_21](https://doi.org/10.1007/978-3-662-44208-1_21)
- Loske, A., Widjaja, T., & Buxmann, P. (2013, December). Cloud Computing Providers' Unrealistic Optimism regarding IT Security Risks: A Threat to Users? *ICIS 2013 Proceedings*.
- Lowe, T. (2019a). Debugging: The key to unlocking the mind of a novice programmer? *2019 IEEE Frontiers in Education Conference (FIE)*, 1–9.  
<https://doi.org/10.1109/FIE43999.2019.9028699>
- Lowe, T. (2019b). Explaining Novice Programmer's Struggles, in Two Parts: Revisiting the ITiCSE 2004 working group's study using dual process theory. *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, 30–36. <https://doi.org/10.1145/3304221.3319775>
- MacCallum, R. C. (2009). Factor Analysis. In *The SAGE Handbook of Quantitative Methods in Psychology* (pp. 123–147). SAGE Publications Ltd.  
<https://doi.org/10.4135/9780857020994>
- Manikas, K., & Hansen, K. M. (2013). Software ecosystems – A systematic literature review. *Journal of Systems and Software*, 86(5), 1294–1306.  
<https://doi.org/10.1016/j.jss.2012.12.026>
- Manzoor, F., Wei, L., & Asif, M. (2021). Intrinsic Rewards and Employee's Performance With the Mediating Mechanism of Employee's Motivation. *Frontiers in Psychology*, 12.  
<https://www.frontiersin.org/articles/10.3389/fpsyg.2021.563070>
- Marquis, D. G. (1962). Individual Responsibility and Group Decisions Involving Risk. *Industrial Management Review*, 3(2), 8.
- Martin, S., Stanton Fraser, D., Fraser, M., Woodgate, D., & Crellin, D. (2010, October).

*The impact of hand held mobile technologies upon children's motivation and learning: 9th World Conference on Mobile and Contextual Learning.*

- Matturro, G. (2013). Soft skills in software engineering: A study of its demand by software companies in Uruguay. *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 133–136.  
<https://doi.org/10.1109/CHASE.2013.6614749>
- Matturro, G., Raschetti, F., & Fontán, C. (2019). A Systematic Mapping Study on Soft Skills in Software Engineering. *Journal of Universal Computer Science*, 25(1), 26.
- Maxwell, S. E., Lau, M. Y., & Howard, G. S. (2015). Is psychology suffering from a replication crisis? What does “failure to replicate” really mean? *American Psychologist*, 70(6), 487–498. <https://doi.org/10.1037/a0039400>
- McAuliffe, M., & Triandafyllidou, A. (2021). *World Migration Report 2022*. international organisation for migration.
- Mitchell, A., Balasubramaniam, D., & Fletcher, J. (2022). Incorporating Ethics in Software Engineering: Challenges and Opportunities. *2022 29th Asia-Pacific Software Engineering Conference (APSEC)*, 90–98.  
<https://doi.org/10.1109/APSEC57359.2022.00021>
- Mohanani, R., Salman, I., Turhan, B., Rodríguez, P., & Ralph, P. (2020). Cognitive Biases in Software Engineering: A Systematic Mapping Study. *IEEE Transactions on Software Engineering*, 46(12), 1318–1339. <https://doi.org/10.1109/TSE.2018.2877759>
- Mølokken, K., & Jørgensen, M. (2005). Expert Estimation of Web-Development Projects: Are Software Professionals in Technical Roles More Optimistic Than Those in Non-Technical Roles? *Empirical Software Engineering*, 10(1), 7–30.  
<https://doi.org/10.1023/B:EMSE.0000048321.46871.2e>
- Molokken-Ostfold, K., & Jorgensen, M. (2005). A comparison of software project overruns - flexible versus sequential development models. *IEEE Transactions on Software Engineering*, 31(9), 754–766. <https://doi.org/10.1109/TSE.2005.96>
- Molokken-Østfold, K., & Jørgensen, M. (2004). Group Processes in Software Effort



- Estimation. *Empirical Software Engineering*, 9(4), 315–334.  
<https://doi.org/10.1023/B:EMSE.0000039882.39206.5a>
- Montandon, J. E., Politowski, C., Silva, L. L., Valente, M. T., Petrillo, F., & Guéhéneuc, Y.-G. (2021). What skills do IT companies look for in new developers? A study with Stack Overflow jobs. *Information and Software Technology*, 129, 106429.  
<https://doi.org/10.1016/j.infsof.2020.106429>
- Moritz, B., Siemsen, E., & Kremer, M. (2014). Judgmental Forecasting: Cognitive Reflection and Decision Speed. *Production and Operations Management*, 23(7), 1146–1160. <https://doi.org/https://doi.org/10.1111/poms.12105>
- Mougouei, D., Perera, H., Hussain, W., Shams, R., & Whittle, J. (2018). Operationalizing human values in software: a research roadmap. *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 780–784.  
<https://doi.org/10.1145/3236024.3264843>
- Mtsweni, E., Hörne, T., & Poll, J. (2016). *Soft Skills for Software Project Team Members*.  
<https://doi.org/10.7763/IJCTE.2016.V8.1035>
- Mustafa, S., Zhang, W., & Naveed, M. M. (2023). What motivates online community contributors to contribute consistently? A case study on Stackoverflow netizens. *Current Psychology*, 42(13), 10468–10481. <https://doi.org/10.1007/s12144-022-03307-4>
- Myers, B. A., & Stylos, J. (2016). Improving API usability. *Communications of the ACM*, 59(6), 62–69. <https://doi.org/10.1145/2896587>
- Mynatt, C., & Sherman, S. J. (1975). Responsibility attribution in groups and individuals: A direct test of the diffusion of responsibility hypothesis. *Journal of Personality and Social Psychology*, 32(6), 1111–1118.  
<https://doi.org/10.1037/0022-3514.32.6.1111>
- Nadi, S., Krüger, S., Mezini, M., & Bodden, E. (2016). Jumping through hoops: Why do Java developers struggle with cryptography APIs? *Proceedings of the 38th International Conference on Software Engineering*, 935–946.

<https://doi.org/10.1145/2884781.2884790>

Nagaria, B., & Hall, T. (2020). Reducing Software Developer Human Errors by Improving Situation Awareness. *IEEE Software*, 37(6), 32–37.

<https://doi.org/10.1109/MS.2020.3014223>

Naiakshina, A. (2020). *Don't Blame Developers! Examining a Password-Storage Study Conducted with Students, Freelancers, and Company Developers*.

Naiakshina, A., Danilova, A., Gerlitz, E., & Smith, M. (2020). On Conducting Security Developer Studies with CS Students: Examining a Password-Storage Study with CS Students, Freelancers, and Company Developers. *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 1–13.

<https://doi.org/10.1145/3313831.3376791>

Naiakshina, A., Danilova, A., Gerlitz, E., von Zezschwitz, E., & Smith, M. (2019). "If you want, I can store the encrypted password": A Password-Storage Field Study with Freelance Developers. *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 1–12. <https://doi.org/10.1145/3290605.3300370>

Naiakshina, A., Danilova, A., Tiefenau, C., Herzog, M., Dechand, S., & Smith, M. (2017). Why Do Developers Get Password Storage Wrong? A Qualitative Usability Study. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 311–328. <https://doi.org/10.1145/3133956.3134082>

Naiakshina, A., Danilova, A., Tiefenau, C., & Smith, M. (2018). *Deception Task Design in Developer Password Studies: Exploring a Student Sample*. 297–313. <https://www.usenix.org/conference/soups2018/presentation/naiakshina>

Nazar, N., Hu, Y., & Jiang, H. (2016). Summarizing Software Artifacts: A Literature Review. *Journal of Computer Science and Technology*, 31(5), 883–909. <https://doi.org/10.1007/s11390-016-1671-1>

NCSC. (2021). *More Master's degrees at UK universities recognised by cyber security experts*. <https://www.ncsc.gov.uk/news/more-university-degrees-certified>.

Newell, A., & Simon, H. A. (1972). *Human problem solving* (Vol. 104). Prentice-hall

- Englewood Cliffs, NJ. [http://www.sci.brooklyn.cuny.edu/~kopec/cis718/fall\\_2005/2/Rafique\\_2\\_humanthinking.doc](http://www.sci.brooklyn.cuny.edu/~kopec/cis718/fall_2005/2/Rafique_2_humanthinking.doc)
- Newman, A., Bavik, Y. L., Mount, M., & Shao, B. (2021). Data Collection via Online Platforms: Challenges and Recommendations for Future Research. *Applied Psychology*, 70(3), 1380–1402. <https://doi.org/10.1111/apps.12302>
- Nguyen, D. C., Wermke, D., Acar, Y., Backes, M., Weir, C., & Fahl, S. (2017). A Stitch in Time: Supporting Android Developers in Writing Secure Code. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 1065–1077. <https://doi.org/10.1145/3133956.3133977>
- Nichols, S. P. (1997). Professional responsibility: The role of the engineer in society. *Science and Engineering Ethics*, 3(3), 327–337. <https://doi.org/10.1007/s11948-997-0039-x>
- Nicholson, J., Coventry, L., & Briggs, P. (2018). Introducing the Cybersurvival Task: Assessing and Addressing Staff Beliefs about Effective Cyber Protection. *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, 443–457.
- Nosek, B. A., Beck, E. D., Campbell, L., Flake, J. K., Hardwicke, T. E., Mellor, D. T., van 't Veer, A. E., & Vazire, S. (2019). Preregistration Is Hard, And Worthwhile. *Trends in Cognitive Sciences*, 23(10), 815–818. <https://doi.org/10.1016/j.tics.2019.07.009>
- Nosek, B. A., Ebersole, C. R., DeHaven, A. C., & Mellor, D. T. (2018). The preregistration revolution. *Proceedings of the National Academy of Sciences*, 115(11), 2600–2606. <https://doi.org/10.1073/pnas.1708274114>
- Oliveira, D. S., Lin, T., Rahman, M. S., Akefirad, R., Ellis, D., Perez, E., Bobhate, R., DeLong, L. A., Cappos, J., & Brun, Y. (2018). {API} Blindspots: Why Experienced Developers Write Vulnerable Code. *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, 315–328.
- Oliveira, D. S., Rosenthal, M., Morin, N., Yeh, K.-C., Cappos, J., & Zhuang, Y. (2014). It's the psychology stupid. *Proceedings of the 30th Annual Computer Security*

- Applications Conference*, 296–305. <https://doi.org/10.1145/2664243.2664254>
- Open Science Collaboration. (2015). Estimating the reproducibility of psychological science. *Science*, *349*(6251), aac4716. <https://doi.org/10.1126/science.aac4716>
- Ortloff, A.-M., Tiefenau, C., & Smith, M. (2023). {SoK}: I Have the (Developer) Power! Sample Size Estimation for Fisher’s Exact, {Chi-Squared}, {McNemar’s}, Wilcoxon {Rank-Sum}, Wilcoxon {Signed-Rank} and t-tests in {Developer-Centered} Usable Security. *Nineteenth Symposium on Usable Privacy and Security (SOUPS 2023)*, 341–359.
- Osmani, M., Weerakkody, V., Hindi, N. M., Al-Esmail, R., Eldabi, T., Kapoor, K., & Irani, Z. (2015). Identifying the trends and impact of graduate attributes on employability: A literature review. *Tertiary Education and Management*, *21*(4), 367–379. <https://doi.org/10.1080/13583883.2015.1114139>
- Pahlke, J., Strasser, S., & Vieider, F. M. (2012). Risk-taking for others under accountability. *Economics Letters*, *114*(1), 102–105. <https://doi.org/10.1016/j.econlet.2011.09.037>
- Pahlke, J., Strasser, S., & Vieider, F. M. (2015). Responsibility effects in decision making under risk. *Journal of Risk and Uncertainty*, *51*(2), 125–146. <https://doi.org/10.1007/s11166-015-9223-6>
- Palassis, A., Speelman, C. P., & Pooley, J. A. (2021). An Exploration of the Psychological Impact of Hacking Victimization. *SAGE Open*, *11*(4), 21582440211061556. <https://doi.org/10.1177/21582440211061556>
- Paley, J., Cheyne, H., Dalglish, L., Duncan, E. A. S., & Niven, C. A. (2007). Nursing’s ways of knowing and dual process theories of cognition. *Journal of Advanced Nursing*, *60*(6), 692–701. <https://doi.org/10.1111/j.1365-2648.2007.04478.x>
- Palmer, S., & Hall, W. (2011). An evaluation of a project-based learning initiative in engineering education. *European Journal of Engineering Education*, *36*(4), 357–365. <https://doi.org/10.1080/03043797.2011.593095>
- Palombo, H., Tabari, A. Z., Lende, D., Ligatti, J., & Ou, X. (2020). An Ethnographic

- Understanding of Software ({In}Security} and a {Co-Creation} Model to Improve Secure Software Development. *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*, 205–220.
- Parsons, K., McCormac, A., Butavicius, M., Pattinson, M., & Jerram, C. (2014). Determining employee awareness using the Human Aspects of Information Security Questionnaire (HAIS-Q). *Computers & Security*, 42, 165–176.  
<https://doi.org/10.1016/j.cose.2013.12.003>
- Patel, D., Patel, H., Sultana, K. Z., & Anu, V. (2023). Programmer Cognition Failures as the Root Cause of Software Vulnerabilities: A Preliminary Review. *2023 Intermountain Engineering, Technology and Computing (IETC)*, 242–246.  
<https://doi.org/10.1109/IETC57902.2023.10152150>
- Patnaik, N., Dwyer, A. C., Hallett, J., & Rashid, A. (2021). *Don't forget your classics: Systematizing 45 years of Ancestry for Security API Usability Recommendations* (No. arXiv:2105.02031). arXiv. <https://arxiv.org/abs/2105.02031>
- Patnaik, N., Dwyer, A. C., Hallett, J., & Rashid, A. (2022). SLR: From Saltzer & Schroeder to 2021...: 47 years of research on the development and validation of Security API recommendations. *ACM Transactions on Software Engineering and Methodology*. <https://doi.org/10.1145/3561383>
- Patrick, A. S., Long, A. C., & Flinn, S. (2003). HCI and security systems. *CHI '03 Extended Abstracts on Human Factors in Computing Systems*, 1056–1057.  
<https://doi.org/10.1145/765891.766146>
- Pattinson, M., Butavicius, M., Parsons, K., McCormac, A., & Calic, D. (2015). Factors that Influence Information Security Behavior: An Australian Web-Based Study. In T. Tryfonas & I. Askoxylakis (Eds.), *Human Aspects of Information Security, Privacy, and Trust* (pp. 231–241). Springer International Publishing.  
[https://doi.org/10.1007/978-3-319-20376-8\\_21](https://doi.org/10.1007/978-3-319-20376-8_21)
- Pennycook, G., McPhetres, J., Zhang, Y., Lu, J. G., & Rand, D. G. (2020). Fighting COVID-19 Misinformation on Social Media: Experimental Evidence for a Scalable

- Accuracy-Nudge Intervention. *Psychological Science*, 31(7), 770–780.  
<https://doi.org/10.1177/0956797620939054>
- Pennycook, G., & Ross, R. M. (2016). Commentary: Cognitive reflection vs. calculation in decision making. *Frontiers in Psychology*, 7.  
<https://doi.org/10.3389/fpsyg.2016.00009>
- Petre, M. (2009). Insights from expert software design practice. *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, 233–242.  
<https://doi.org/10.1145/1595696.1595731>
- Petre, M. (2022). Exploring cognitive bias “in the wild”: technical perspective. *Communications of the ACM*, 65(4), 114–114. <https://doi.org/10.1145/3517215>
- Pieczul, O., Foley, S., & Zurko, M. E. (2017). Developer-centered security and the symmetry of ignorance. *NSPW 2017: Proceedings of the 2017 New Security Paradigms Workshop*, 46–56. <https://doi.org/https://doi.org/10.1145/3171533.3171539>
- Pikkarainen, M., Haikara, J., Salo, O., Abrahamsson, P., & Still, J. (2008). The impact of agile practices on communication in software development. *Empirical Software Engineering*, 13(3), 303–337. <https://doi.org/10.1007/s10664-008-9065-9>
- Poller, A., Kocksch, L., Kinder-Kurlanda, K., & Epp, F. A. (2016). First-time Security Audits as a Turning Point? Challenges for Security Practices in an Industry Software Development Team. *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, 1288–1294.  
<https://doi.org/10.1145/2851581.2892392>
- Poller, A., Kocksch, L., Türpe, S., Epp, F. A., & Kinder-Kurlanda, K. (2017). Can Security Become a Routine?: A Study of Organizational Change in an Agile Software Development Group. *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, 2489–2503.  
<https://doi.org/10.1145/2998181.2998191>
- Polman, E., & Wu, K. (2020). Decision making for others involving risk: A review and

- meta-analysis. *Journal of Economic Psychology*, 77, 102184.  
<https://doi.org/10.1016/j.joep.2019.06.007>
- Postmes, T., Tanis, M., & de Wit, B. (2001). Communication and Commitment in Organizations: A Social Identity Approach. *Group Processes & Intergroup Relations*, 4(3), 227–246. <https://doi.org/10.1177/1368430201004003004>
- Prat, C. S., Madhyastha, T. M., Mottarella, M. J., & Kuo, C. H. (2020). Relating Natural Language Aptitude to Individual Differences in Learning Programming Languages. *Sci Rep*, 10(1), 3817. <https://doi.org/10.1038/s41598-020-60661-8>
- Pretorius, C., Razavian, M., Eling, K., & Langerak, F. (2018). Towards a Dual Processing Perspective of Software Architecture Decision Making. *2018 IEEE International Conference on Software Architecture Companion (ICSA-C)*, 48–51. <https://doi.org/10.1109/ICSA-C.2018.00021>
- Pretz, J., Naples, A., & Sternberg, R. (2003). Recognising, Defining, and Representing Problems. In J. E. Davidson & R. J. Sternberg (Eds.), *The Psychology of Problem Solving* (pp. 3–30). Cambridge University Press.
- Rabelo, D., Lopes, A., Mendes, W., de Souza, C., Gama, K., Monteiro, D., & Pinto, G. (2022). The Role of Non-Technical Skills in the Software Development Market. *Proceedings of the XXXVI Brazilian Symposium on Software Engineering*, 31–40. <https://doi.org/10.1145/3555228.3555254>
- Rahman, A., Farhana, E., & Imtiaz, N. (2019). Snakes in Paradise?: Insecure Python-Related Coding Practices in Stack Overflow. *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, 200–204. <https://doi.org/10.1109/MSR.2019.00040>
- Rahman, S. (2017). The Advantages and Disadvantages of Using Qualitative and Quantitative Approaches and Methods in Language “Testing and Assessment” Research: A Literature Review. *Journal of Education and Learning*, 6(1), 102–112. <https://eric.ed.gov/?id=EJ1120221>
- Ralph, P. (2013). Possible core theories for software engineering. *2013 2nd SEMAT*

- Workshop on a General Theory of Software Engineering (GTSE)*, 35–38.  
<https://doi.org/10.1109/GTSE.2013.6613868>
- Ralph, P. (2011). Toward a Theory of Debiasing Software Development. In S. Wrycza (Ed.), *Research in Systems Analysis and Design: Models and Methods* (pp. 92–105). Springer. [https://doi.org/10.1007/978-3-642-25676-9\\_8](https://doi.org/10.1007/978-3-642-25676-9_8)
- Ralph, P., & Tempero, E. (2016). Characteristics of decision-making during coding. *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, 1–10. <https://doi.org/10.1145/2915970.2915990>
- Rampin, R., & Rampin, V. (2021). Taguette: Open-source qualitative data analysis. *Journal of Open Source Software*, 6(68), 3522. <https://doi.org/10.21105/joss.03522>
- Rashid, A., Chivers, H., Danezis, G., Lupu, E., Martin, A., & Schneider, S. (2021). *The Cyber Security Body of Knowledge* (Version 1.1.0).
- Rauf, I., Lopez, T., Sharp, H., Petre, M., Tun, T., Levine, M., Towse, J., van der Linden, D., Rashid, A., & Nuseibeh, B. (2022). Influences of developers’ perspectives on their engagement with security in code. *2022 IEEE/ACM 15th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 86–95.  
<https://doi.org/10.1145/3528579.3529180>
- Rauf, I., Petre, M., Tun, T., Lopez, T., Lunn, P., Van Der Linden, D., Towse, J., Sharp, H., Levine, M., Rashid, A., & Nuseibeh, B. (2021). The Case for Adaptive Security Interventions. *ACM Transactions on Software Engineering and Methodology*, 31(1), 9:1–9:52. <https://doi.org/10.1145/3471930>
- Rauf, I., Petre, M., Tun, T., Lopez, T., & Nuseibeh, B. (2023). Security Thinking in Online Freelance Software Development. *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*, 13–24. <https://doi.org/10.1109/ICSE-SEIS58686.2023.00008>
- Rauf, I., van der Linden, D., Levine, M., Towse, J., Nuseibeh, B., & Rashid, A. (2020). Security but not for security’s sake. *ICSEW’20: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, 141–144.



- <https://doi.org/10.1145/3387940.3392230>
- Reich, R., Sahami, M., Weinstein, J. M., & Cohen, H. (2020). Teaching Computer Ethics: A Deeply Multidisciplinary Approach. *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 296–302.
- <https://doi.org/10.1145/3328778.3366951>
- Rhee, H.-S., Ryu, Y. U., & Kim, C.-T. (2012). Unrealistic optimism on information security management. *Computers & Security*, 31(2), 221–232.
- <https://doi.org/10.1016/j.cose.2011.12.001>
- Riger, S., & Sigurvinsdottir, R. (2016). Thematic Analysis. In L. Jason & D. Glenwick (Eds.), *Handbook of Methodological Approaches to Community-based Research: Qualitative, Quantitative, and Mixed Methods* (pp. 33–41). Oxford University Press.
- Ritter, B. A., Small, E. E., Mortimer, J. W., & Doll, J. L. (2018). Designing Management Curriculum for Workplace Readiness: Developing Students' Soft Skills. *Journal of Management Education*, 42(1), 80–103. <https://doi.org/10.1177/1052562917703679>
- Robillard, M. P. (2009). What Makes APIs Hard to Learn? Answers from Developers. *IEEE Software*, 26(6), 27–34. <https://doi.org/10.1109/MS.2009.193>
- Robins, A. V. (2022). Dual Process Theories: Computing Cognition in Context. *ACM Transactions on Computing Education*, 22(4), 41:1–41:31.
- <https://doi.org/10.1145/3487055>
- Roesler, D. (2020). A Computer Science Academic Vocabulary List. *Dissertations and Theses*. <https://doi.org/10.15760/etd.7414>
- Rogers, E., M. (1995). *Diffusion of Innovation* (4th ed.). The Free Press.
- Ryan, R. M., & Deci, E. L. (2000). Self-Determination Theory and the Facilitation of Intrinsic Motivation, Social Development, and Well-Being. *American Psychologist*, 55(1), 67. <https://doi.org/10.1037/0003-066X.55.1.68>
- Salehi, S., Wang, K. D., Toorawa, R., & Wieman, C. (2020). Can Majoring in Computer Science Improve General Problem-solving Skills? *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 156–161.

<https://doi.org/10.1145/3328778.3366808>

- Salman, I., Turhan, B., & Vegas, S. (2019). A controlled experiment on time pressure and confirmation bias in functional software testing. *Empirical Software Engineering*, 24(4), 1727–1761. <https://doi.org/10.1007/s10664-018-9668-8>
- Samuels, R., Stich, S., & Bishop, M. (2012). Ending the Rationality Wars; How to Make Disputes About Human Rationality Disappear. In S. Stich (Ed.), *Collected Papers, Volume 2: Knowledge, Rationality, and Morality, 1978-2010* (Vol. 11). OUP USA.
- Sarabipour, S., Debat, H. J., Emmott, E., Burgess, S. J., Schwessinger, B., & Hensel, Z. (2019). On the value of preprints: An early career researcher perspective. *PLOS Biology*, 17(2), e3000151. <https://doi.org/10.1371/journal.pbio.3000151>
- Sarkar, M., Overton, T., Thompson, C., & Rayner, G. (2016). Graduate Employability: Views of Recent Science Graduates and Employers. *International Journal of Innovation in Science and Mathematics Education*, 24(3).
- Sasse, M. A., Brostoff, S., & Weirich, D. (2001). Transforming the “Weakest Link” — a Human/Computer Interaction Approach to Usable and Effective Security. *BT Technology Journal*, 19(3), 122–131. <https://doi.org/10.1023/A:1011902718709>
- Satchell, C., & Dourish, P. (2009). Beyond the user: Use and non-use in HCI. *Proceedings of the 21st Annual Conference of the Australian Computer-Human Interaction Special Interest Group: Design: Open 24/7*, 9–16. <https://doi.org/10.1145/1738826.1738829>
- Sawyer, T., White, M., Zaveri, P., Chang, T., Ades, A., French, H., Anderson, J., Auerbach, M., Johnston, L., & Kessler, D. (2015). Learn, See, Practice, Prove, Do, Maintain: An Evidence-Based Pedagogical Framework for Procedural Skill Training in Medicine. *Academic Medicine*, 90(8), 1025. <https://doi.org/10.1097/ACM.0000000000000734>
- Scaffidi, C. (2018). Employers’ Needs for Computer Science, Information Technology and Software Engineering Skills Among New Graduates. *International Journal of Computer Science, Engineering and Information Technology*, 8(1), 01–12. <https://doi.org/10.5121/ijcseit.2018.8101>

- Scheffler, I. (1965). *Conditions of Knowledge*. Scott, Foresman.
- Schieferdecker, I. (2020). Responsible Software Engineering. In S. Goericke (Ed.), *The Future of Software Quality Assurance* (pp. 137–146). Springer International Publishing. <https://doi.org/10.1007/978-3-030-29509-7>
- Schielzeth, H., Dingemanse, N. J., Nakagawa, S., Westneat, D. F., Allegate, H., Teplitsky, C., Réale, D., Dochtermann, N. A., Garamszegi, L. Z., & Araya-Ajoy, Y. G. (2020). Robustness of linear mixed-effects models to violations of distributional assumptions. *Methods in Ecology and Evolution*, 11(9), 1141–1152. <https://doi.org/10.1111/2041-210X.13434>
- Schneier, B. (2008). The Psychology of Security. In S. Vaudenay (Ed.), *Progress in Cryptology – AFRICACRYPT 2008* (pp. 50–79). Springer. [https://doi.org/10.1007/978-3-540-68164-9\\_5](https://doi.org/10.1007/978-3-540-68164-9_5)
- Scott, C. R. (2007a). Communication and Social Identity Theory: Existing and Potential Connections in Organizational Identification Research. *Communication Studies*, 58(2), 123–138. <https://doi.org/10.1080/10510970701341063>
- Scott, C. R., Corman, S. R., & Cheney, G. (1998). Development of a Structural Model of Identification in the Organization. *Communication Theory*, 8(3), 298–336. <https://doi.org/10.1111/j.1468-2885.1998.tb00223.x>
- Scott, D. (2007b). Resolving the quantitative–qualitative dilemma: a critical realist approach. *International Journal of Research & Method in Education*, 30(1), 3–17. <https://doi.org/10.1080/17437270701207694>
- Scott, S. G., & Bruce, R. A. (1995). Decision-Making Style: The Development and Assessment of a New Measure. *Educational and Psychological Measurement*, 55(5), 818–831. <https://doi.org/10.1177/0013164495055005017>
- Sedano, T., Ralph, P., & Péraire, C. (2017). Software Development Waste. 2017 *IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, 130–140. <https://doi.org/10.1109/ICSE.2017.20>
- Sergeyuk, A., Titov, S., Golubev, Y., & Bryksin, T. (2023). *Overcoming the Mental Set*

*Effect in Programming Problem Solving.*

- Sharot, T. (2011). The optimism bias. *Current Biology*, 21(23), R941–R945.  
<https://doi.org/10.1016/j.cub.2011.10.030>
- Sharp, H., Robinson, H., & Woodman, M. (2000). Software engineering: Community and culture. *IEEE Software*, 17(1), 40–47. <https://doi.org/10.1109/52.819967>
- Shepperd, J. A., Waters, E., Weinstein, N. D., & Klein, W. M. P. (2015). A Primer on Unrealistic Optimism. *Current Directions in Psychological Science*, 24(3), 232–237.  
<https://doi.org/10.1177/0963721414568341>
- Shreeve, B., Gralha, C., Rashid, A., Araujo, J., & Goulão, M. (2022). Making sense of the unknown: How managers make cyber security decisions. *ACM Transactions on Software Engineering and Methodology*. <https://doi.org/10.1145/3548682>
- Shreeve, B., Hallett, J., Edwards, M., Anthonysamy, P., Frey, S., & Rashid, A. (2020). "So if Mr Blue Head here clicks the link.." Risk Thinking in Cyber Security Decision Making. *ACM Transactions on Privacy and Security*, 24(1), 5:1–5:29.  
<https://doi.org/10.1145/3419101>
- Siegel, H. (2013). *Educating Reason*. Routledge.
- Siegrist, M., Gutscher, H., & Earle, T. C. (2005). Perception of risk: the influence of general trust, and general confidence. *Journal of Risk Research*, 8(2), 145–156.  
<https://doi.org/10.1080/1366987032000105315>
- Simon, H. A. (1956). Rational choice and the structure of the environment. *Psychological Review*, 63(2), 129–138. <https://doi.org/10.1037/h0042769>
- Simon, H. A. (1990). Bounded Rationality. In J. Eatwell, M. Milgate, & P. Newman (Eds.), *Utility and Probability* (pp. 15–18). Palgrave Macmillan UK.  
[https://doi.org/10.1007/978-1-349-20568-4\\_5](https://doi.org/10.1007/978-1-349-20568-4_5)
- Sinayev, A., & Peters, E. (2015). Cognitive reflection vs. calculation in decision making. *Frontiers in Psychology*, 6. <https://doi.org/10.3389/fpsyg.2015.00532>
- Singleton, R., & Straits, B. (1988). *Approaches to social research* (6th ed.). Oxford University Press. <https://cir.nii.ac.jp/crid/1130282271402220160>

- Slembrouck, S. (2015). The Role of the Researcher in Interview Narratives. In *The Handbook of Narrative Analysis* (pp. 239–254). John Wiley & Sons, Ltd.  
<https://doi.org/10.1002/9781118458204.ch12>
- Smart, W. (2018). *Lessons learned review of the WannaCry Ransomware Cyber Attack* (pp. 1–42). National Health Service.
- Smith, S. W. (2003). Humans in the loop: Human-computer interaction and security. *IEEE Security & Privacy*, 1(3), 75–79. <https://doi.org/10.1109/MSECP.2003.1203228>
- Spadini, D., Çalikli, G., & Bacchelli, A. (2020). Primers or Reminders? The Effects of Existing Review Comments on Code Review. *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, 1171–1182.
- Speirs, J. C., Stetzer, M. R., Lindsey, B. A., & Kryjevskaa, M. (2021). Exploring and supporting student reasoning in physics by leveraging dual-process theories of reasoning and decision making. *Physical Review Physics Education Research*, 17(2), 020137. <https://doi.org/10.1103/PhysRevPhysEducRes.17.020137>
- Stacy, W., & MacMillan, J. (1995). Cognitive bias in software engineering. *Communications of the ACM*, 38(6), 57–63. <https://doi.org/10.1145/203241.203256>
- Stagnaro, M., Pennycook, G., & Rand, D. G. (2018). Performance on the Cognitive Reflection Test is stable across time. *Stagnaro, MN, Pennycook, G., & Rand, DG (2018) Performance on the Cognitive Reflection Test Is Stable Across Time. Judgment and Decision Making*, 13, 260–267.
- Stamms, K., Lin, L., & Christidis, P. (2016). Datapoint: What do people do with their psychology degrees? *Monitor on Psychology*, 47(6), 12.  
<https://www.apa.org/monitor/2016/06/datapoint>
- Stanovich, K. E., & West, R. F. (2014). The Assessment of Rational Thinking: IQ ≠ RQ. *Teaching of Psychology*, 41(3), 265–271. <https://doi.org/10.1177/0098628314537988>
- Stelzer, D., & Mellis, W. (1998). Success factors of organizational change in software process improvement. *Software Process: Improvement and Practice*, 4(4), 227–250.  
[https://doi.org/10.1002/\(SICI\)1099-1670\(199812\)4:4%3C227::AID-](https://doi.org/10.1002/(SICI)1099-1670(199812)4:4%3C227::AID-)

SPIP106%3E3.0.CO;2-1

Sternberg, R. J. (1986a). *Critical Thinking: Its Nature, Measurement, and Improvement*.

<https://eric.ed.gov/?id=ED272882>

Sternberg, R. J. (1986b). *Intelligence applied : understanding and increasing your intellectual skills* / [Ressource physique]. Harcourt Brace Jovanovich,.

<https://eduq.info/xmlui/handle/11515/16423>

Stevens, M., & Norman, R. (2016). Industry expectations of soft skills in IT graduates: A regional survey. *Proceedings of the Australasian Computer Science Week*

*Multiconference*, 1–9. <https://doi.org/10.1145/2843043.2843068>

Succi, C., & Canovi, M. (2020). Soft skills to enhance graduate employability: Comparing students and employers' perceptions. *Studies in Higher Education*, 45(9), 1834–1847.

<https://doi.org/10.1080/03075079.2019.1585420>

Sukamolson, S. (2007). Fundamentals of quantitative research. *Language Institute Chulalongkorn University*, 1(3), 1–20.

[https://www.researchgate.net/profile/Vihan-Moodi/post/What\\_are\\_the\\_characteristics\\_of\\_quantitative\\_research/attachment/5f3091d0ed60840001c62a27/AS%3A922776944787456%401597018576221/download/SuphatSukamolson.pdf](https://www.researchgate.net/profile/Vihan-Moodi/post/What_are_the_characteristics_of_quantitative_research/attachment/5f3091d0ed60840001c62a27/AS%3A922776944787456%401597018576221/download/SuphatSukamolson.pdf)

Sussman, L. L. (2021). Exploring the Value of Non-Technical Knowledge, Skills, and Abilities (KSAs) to Cybersecurity Hiring Managers. *Journal of Higher Education Theory and Practice*, 21(6), 99–117.

Sweller, J., van Merriënboer, J. J. G., & Paas, F. G. W. C. (1998). Cognitive

Architecture and Instructional Design. *Educational Psychology Review*, 10(3), 251–296. <https://doi.org/10.1023/A:1022193728205>

Tahaei, M., Jenkins, A., Vaniea, K., & Wolters, M. (2020a). “I Don’t Know Too Much About It”: On the Security Mindsets of Computer Science Students. *International Workshop on Socio-Technical Aspects in Security and Trust*, 20.

<https://doi.org/10.1007/978-3-030-55958-8>

Tahaei, M., & Vaniea, K. (2019). A Survey on Developer-Centred Security. *2019 IEEE*

- European Symposium on Security and Privacy Workshops (EuroS&PW)*, 129–138.  
<https://doi.org/10.1109/EuroSPW.2019.00021>
- Tahaei, M., Vaniea, K., & Saphra, N. (2020b). Understanding Privacy-Related Questions on Stack Overflow. *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 1–14. <https://doi.org/10.1145/3313831.3376768>
- Taherdoost, H. (2022). *How to Conduct an Effective Interview; A Guide to Interview Design in Research Study*. <https://papers.ssrn.com/abstract=4178687>
- Tajfel, H., & Turner, J. C. (2010). An integrative theory of intergroup conflict. In T. Postmes & N. Branscombe (Eds.), *Rediscovering Social Identity* (1st ed., pp. 56–65). Routledge.
- Tang, A. (2011). Software designers, are you biased? *Proceedings of the 6th International Workshop on SHaring and Reusing Architectural Knowledge*, 1–8.  
<https://doi.org/10.1145/1988676.1988678>
- Tanis, M., & Postmes, T. (2005). A social identity approach to trust: Interpersonal perception, group membership and trusting behaviour. *European Journal of Social Psychology*, 35(3), 413–424. <https://doi.org/10.1002/ejsp.256>
- Tasneem, S. (2012). Critical thinking in an introductory programming course. *Journal of Computing Sciences in Colleges*, 27(6), 81–83.
- Tavares de Souza, A. L. O., & Costa Pinto, V. H. S. (2020). Toward a Definition of Cognitive-Driven Development. *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 776–778.  
<https://doi.org/10.1109/ICSME46990.2020.00087>
- Taylor-Jackson, J., McAlaney, J., Foster, J. L., Bello, A., Maurushat, A., & Dale, J. (2020). Incorporating Psychology into Cyber Security Education: A Pedagogical Approach. In M. Bernhard, A. Bracciali, L. J. Camp, S. Matsuo, A. Maurushat, P. B. Rønne, & M. Sala (Eds.), *Financial Cryptography and Data Security* (pp. 207–217). Springer International Publishing. [https://doi.org/10.1007/978-3-030-54455-3\\_15](https://doi.org/10.1007/978-3-030-54455-3_15)
- Teh, A., Baniassad, E., van Rooy, D., & Boughton, C. (2012). Social Psychology and

- Software Teams: Establishing Task-Effective Group Norms. *IEEE Software*, 29(4), 53–58. <https://doi.org/10.1109/MS.2011.157>
- Tetlock, P. E., & Kim, J. I. (1987). Accountability and judgment processes in a personality prediction task. *Journal of Personality and Social Psychology*, 52, 700–709. <https://doi.org/10.1037/0022-3514.52.4.700>
- Theofanidis, D., & Fountouki, A. (2018). Limitations and Delimitations in the Research Process. *Perioperative Nursing - Quarterly Scientific, Online Official Journal of G.O.R.N.A., Volume 7 (2018)*(Issue 3 September-December 2018), 155–163. <https://www.spnj.gr/en/limitations-and-delimitations-in-the-research-process-p160.html>
- Thissen, D., & Steinberg, L. (2009). Item Response Theory. In *The SAGE Handbook of Quantitative Methods in Psychology* (pp. 148–177). SAGE Publications Ltd. <https://doi.org/10.4135/9780857020994>
- Thomson, K. S., & Oppenheimer, D. M. (2016). Investigating an alternate form of the cognitive reflection test. *Judgment and Decision Making*, 11(1), 99.
- Tom, E., Aurum, A., & Vidgen, R. (2013). An exploration of technical debt. *Journal of Systems and Software*, 86(6), 1498–1516. <https://doi.org/10.1016/j.jss.2012.12.052>
- Tomlinson, M. (2017). Student perceptions of themselves as “consumers” of higher education. *British Journal of Sociology of Education*, 38(4), 450–467. <https://doi.org/10.1080/01425692.2015.1113856>
- Toplak, M. E., West, R. F., & Stanovich, K. E. (2011). The Cognitive Reflection Test as a predictor of performance on heuristics-and-biases tasks. *Mem Cognit*, 39(7), 1275–1289. <https://doi.org/10.3758/s13421-011-0104-1>
- Towse, A., Ellis, D. A., & Towse, J. (2021a). Making data meaningful: Guidelines for good quality open data. *The Journal of Social Psychology*, 161(4), 395–402. <https://doi.org/10.1080/00224545.2021.1938811>
- Towse, J., Ellis, D. A., & Towse, A. (2021b). Opening Pandora’s Box: Peeking inside Psychology’s data sharing practices, and seven recommendations for change. *Behavior Research Methods*, 53(4), 1455–1468. <https://doi.org/10.3758/s13428-020-01486-1>



- Towse, J., Levine, M., Petre, M., Bandara, A., Lopez, T., Rashid, A., Rauf, I., Sharp, H., Tun, T., van der Linden, D., & Nuseibeh, B. (2020 (in press)). The Case for Understanding Secure Coding as a Psychological Enterprise. *Cyberpsychology, Behavior, and Social Networking*.
- Turner, J. C., Brown, R., & Tajfel, H. (1979). Social comparison and group interest in ingroup favouritism. *European Journal of Social Psychology*, 9(2), 187–204.  
<https://doi.org/10.1002/ejsp.2420090207>
- Tversky, A. (1977). Features of similarity. *Psychological Review*, 84, 327–352.  
<https://doi.org/10.1037/0033-295X.84.4.327>
- Valstar, S., Krause-Levy, S., Macedo, A., Griswold, W. G., & Porter, L. (2020a). Faculty Views on the Goals of an Undergraduate CS Education and the Academia-Industry Gap. *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 577–583. <https://doi.org/10.1145/3328778.3366834>
- Valstar, S., Sih, C., Krause-Levy, S., Porter, L., & Griswold, W. G. (2020b). A Quantitative Study of Faculty Views on the Goals of an Undergraduate CS Program and Preparing Students for Industry. *Proceedings of the 2020 ACM Conference on International Computing Education Research*, 113–123.  
<https://doi.org/10.1145/3372782.3406277>
- van der Linden, D., Anthonysamy, P., Nuseibeh, B., Tun, T., Petre, M., Levine, M., Towse, J., & Rashid, A. (2020a). *Schrödinger's security: opening the box on app developers' security rationale*. 149–160. <https://doi.org/10.1145/3377811.3380394>
- van der Linden, D., Williams, E., Hallett, J., & Rashid, A. (2020b). The impact of surface features on choice of (in)secure answers by Stackoverflow readers. *IEEE Transactions on Software Engineering*, 1–1. <https://doi.org/10.1109/tse.2020.2981317>
- Van Rookhuijzen, M., De Vet, E., & Adriaanse, M. A. (2021). The Effects of Nudges: One-Shot Only? Exploring the Temporal Spillover Effects of a Default Nudge. *Frontiers in Psychology*, 12.
- Veracode. (2020). *State of Software Security* (No. 11). Veracode.

- Veracode. (2023). *State of Software Security*. Veracode.  
[https://info.veracode.com/rs/790-ZKW-291/images/Veracode\\_State\\_of\\_Software\\_Security\\_2023.pdf](https://info.veracode.com/rs/790-ZKW-291/images/Veracode_State_of_Software_Security_2023.pdf)
- Wason, P. (1960). On the Failure to Eliminate Hypotheses in a Conceptual Task. *Quarterly Journal of Experimental Psychology*, 12(3), 129–140.  
<https://doi.org/10.1080/17470216008416717>
- Wason, P., & Evans, J. St. B. T. (1974). Dual processes in reasoning? *Cognition*, 3(2), 141–154. [https://doi.org/10.1016/0010-0277\(74\)90017-1](https://doi.org/10.1016/0010-0277(74)90017-1)
- Watson, C., & Blincoe, K. (2017). *Attitudes Towards Software Engineering Education in the New Zealand Industry*.
- Weinberg, G. M. (1971). *The psychology of computer programming* (Vol. 29). Van Nostrand Reinhold New York.
- Weinstein, N. D. (1980). Unrealistic optimism about future life events. *Journal of Personality and Social Psychology*, 39(5), 806–820.  
<https://doi.org/10.1037/0022-3514.39.5.806>
- Weir, C., Becker, I., & Blair, L. (2021). A Passion for Security: Intervening to Help Software Developers. *International Conference on Software Engineering: Software Engineering in Practice*, 21–30. <https://doi.org/10.1109/ICSE-SEIP52600.2021.00011>
- Weir, C., Becker, I., Noble, J., Blair, L., Sasse, M. A., & Rashid, A. (2020a). Interventions for long-term software security: Creating a lightweight program of assurance techniques for developers. *Software: Practice and Experience*, 50(3), 275–298. <https://doi.org/10.1002/spe.2774>
- Weir, C., Hermann, B., & Fahl, S. (2020b). *From Needs to Actions to Secure Apps? The Effect of Requirements and Developer Practices on App Security*.
- Weir, C., Rashid, A., & Noble, J. (2016). *How to Improve the Security Skills of Mobile App Developers? Comparing and Contrasting Expert Views*.  
<https://www.usenix.org/conference/soups2016/workshop-program/wsiw16/presentation/weir>

- Welsh, M. B., Burns, N. R., & Delfabbro, P. H. (2013). *The Cognitive Reflection Test: how much more than Numerical Ability?* 7.
- West, R. F. (2008). The Psychology of Security. *THE PSYCHOLOGY OF SECURITY*, 51(4), 34.
- Whittle, J., Ferrario, M. A., Simm, W., & Hussain, W. (2021). A Case for Human Values in Software Engineering. *IEEE Software*, 38(1), 106–113.  
<https://doi.org/10.1109/MS.2019.2956701>
- Wiederhold, B. K. (2014). The Role of Psychology in Enhancing Cybersecurity. *Cyberpsychology, Behavior, and Social Networking*, 17(3), 131–132.  
<https://doi.org/10.1089/cyber.2014.1502>
- Wijayarathna, C., & Arachchilage, N. A. G. (2019). Why Johnny can't develop a secure application? A usability analysis of Java Secure Socket Extension API. *Computers & Security*, 80, 54–73. <https://doi.org/10.1016/j.cose.2018.09.007>
- Winter, E., Forshaw, S., Hunt, L., & Ferrario, M. A. (2019). Advancing the Study of Human Values in Software Engineering. *2019 IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 19–26. <https://doi.org/10.1109/CHASE.2019.00012>
- Winter, S., Timperley, C. S., Hermann, B., Cito, J., Bell, J., Hilton, M., & Beyer, D. (2022). A retrospective study of one decade of artifact evaluations. *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 145–156.  
<https://doi.org/10.1145/3540250.3549172>
- Witschey, J., Xiao, S., & Murphy-Hill, E. (2014). Technical and Personal Factors Influencing Developers' Adoption of Security Tools. *Proceedings of the 2014 ACM Workshop on Security Information Workers*, 23–26.  
<https://doi.org/10.1145/2663887.2663898>
- Witschey, J., Zielinska, O., Welk, A., Murphy-Hill, E., Mayhorn, C., & Zimmermann, T. (2015). *Quantifying developers' adoption of security tools*. 260–271.

<https://doi.org/10.1145/2786805.2786816>

Wood, R., & Bandura, A. (1989). Social Cognitive Theory of Organizational Management. *Academy of Management Review*, 14(3), 361–384.

<https://doi.org/10.5465/amr.1989.4279067>

Wu, J.-H., Chen, Y.-C., & Chang, J. (2007). Critical IS professional activities and skills/knowledge: A perspective of IS managers. *Computers in Human Behavior*, 23(6), 2945–2965. <https://doi.org/10.1016/j.chb.2006.08.008>

Wurster, G., & van Oorschot, P. (2008). The developer is the enemy. *Proceedings of the 2008 New Security Paradigms Workshop*, 89–97.

<https://doi.org/10.1145/1595676.1595691>

Wyrich, M., Preikschat, A., Graziotin, D., & Wagner, S. (2021). The Mind Is a Powerful Place: How Showing Code Comprehensibility Metrics Influences Code Understanding. *arXiv:2012.09590 [Cs]*. <https://arxiv.org/abs/2012.09590>

Xiao, S., Witschey, J., & Murphy-Hill, E. (2014). Social Influences on Secure Development Tool Adoption: Why Security Tools Spread. *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing*, 1095–1106. <https://doi.org/10.1145/2531602.2531722>

Xie, J., Lipford, H. R., & Chu, B. (2011). Why do programmers make security errors? *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 161–164. <https://doi.org/10.1109/VLHCC.2011.6070393>

Zayour, I., Moukadem, I., & Moghrabi, I. (2013). Complexity is in the Brain of the Beholder: A Psychological Perspective on Software Engineering’s Ultimate Challenge. *Journal of Software*, 8(5), 1079–1085. <https://doi.org/10.4304/jsw.8.5.1079-1085>

Zheng, G., Zhang, C., & Li, L. (2015). Practicing and Evaluating Soft Skills in IT Capstone Projects. *Proceedings of the 16th Annual Conference on Information Technology Education*, 109–113. <https://doi.org/10.1145/2808006.2808041>

Zhu, J., Xie, J., Lipford, H. R., & Chu, B. (2014). Supporting secure programming in web applications through interactive static analysis. *Journal of Advanced Research*, 5(4),

449–462. <https://doi.org/10.1016/j.jare.2013.11.006>