Using Workshops to Improve Security in Software Development Teams

Charles Weir

Security Lancaster

School of Computing and Communications

This dissertation is submitted for the degree of Doctor of Philosophy

October 2020

With thanks once again to my wife Julia

“You were saved not by work, but for work. Do it till all is done. By your Inventions, Innovations, Initiatives, Improvements, Involvements, Imaginations, Information, Interventions and Inspirations... Go the extra mile and dare to do it.*” (Israelmore Ayivor)*

Declaration

This thesis has not been submitted in support of an application for another degree at this or any other university. It is the result of my own work and includes nothing that is the outcome of work done in collaboration except where specifically indicated.

Some of the ideas in this thesis were the product of discussion with my supervisors Lynne Blair, James Noble of the Victoria University of Wellington, and Awais Rashid of the University of Bristol; and with collaborators Ingolf Becker and Angela Sasse of University College London, Sascha Fahl of the Ruhr University Bochum and later Leibniz University Hannover, Christian Stransky and Dominik Wermke of Leibniz University Hannover, and Ben Hermann of Paderborn University.

In terms of others' contributions, the dual coding for the Magid and Magid2 projects was by Ingolf Becker; he also wrote a first version of the Blockers and Motivators discussion in Section 2.2.7.

For the Developer Survey chapter, Sascha Fahl instigated and mentored the survey project, suggested Figure 20 and wrote the initial version of Section 5.1.2; Ben Hermann created and ran the application analysis software package and wrote the initial version of Section 5.2.1; Christian Stransky generated the lists of invitation email addresses, obtained the corresponding application binaries and wrote the initial version of Section 5.1.5; Dominik Wermke created the initial Python Jupyter Notebook analysis plus Figure 25, Figure 27, Figure 29, Figure 31, and Figure 35 in Sections 5.3.2 to 5.3.4.

Section 3.3 is based closely on a similar description in an earlier thesis by the author [183].

Excerpts of this thesis have been published in the following peer reviewed conference proceedings, datasets and journals:

1. Weir, C., Hermann, B., and Fahl, S. *From Needs to Actions to Secure Apps? The Effect of Requirements and Developer Practices on App Security*. 29th USENIX Security Symposium (USENIX Security 2020).
2. Weir, C., Noble, J., and Rashid, A. *Challenging Software Developers: Dialectic as a Foundation for Security Assurance Techniques*. Journal of Cybersecurity, (2020), 30.
3. Weir, C., Becker, I., Noble, J., et al. *Interventions for Long-Term Software Security: Creating a Lightweight Program of Assurance Techniques for Developers*. Software - Practice and Experience, October 2019, 275–298.
4. Weir, C., Becker, I., Noble, J., Blair, L., Sasse, M.A., and Rashid, A. *Interventions for Software Security: Creating a Lightweight Program of Assurance Techniques for Developers*. Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, IEEE (2019).
5. Weir, C., Blair, L., Becker, I., Sasse, M.A., and Noble, J. *Light-touch Interventions to Improve Software Development Security*. IEEE Cybersecurity Development Conference, IEEE Computer Society (2018), 12.
6. Weir, C., Hermann, B., Stransky, C., Wermke, D., and Fahl, S. *Public Dataset from Online Android App Developer Survey*. 2019. https://dx.doi.org/10.17635/lancaster/researchdata/319.

This thesis may contain some overlap with text in the following publications by the author:

1. Weir, C. *Secure Development,* https://www.securedevelopment.org/
2. Weir, C., Blair, L., Noble, J., Becker, I., and Sasse, M.A. *Developer Cyber Essentials: Trialling Interventions to Improve Development Security*. Lancaster, 2018.
3. Weir, C. and Ford, N. *The Secure Development Handbook: Step by Step to Software Security*. Amazon, London, 2018.
4. Weir, C., Rashid, A., and Noble, J. *Developer Essentials: Top Five Interventions to Support Secure Software Development*. Lancaster University, Lancaster, 2017.

Further papers will be published containing additional material from the chapters in this thesis.

Abstract

Though some software development teams are highly effective at delivering security, others either do not care or do not have access to security experts to teach them how. Unfortunately, these latter teams are still responsible for the security of the systems they build: systems that are ever more important to ever more people. Yet many, perhaps most, security problems can be prevented with careful design, construction and configuration of the software and systems involved, so software developers have a major contribution to make.

This research investigates how to help teams of software developers achieve better security. An initial qualitative survey of 15 secure software development professionals highlighted a range of security assurance and motivation techniques suitable for teams of developers, and emphasised the human interaction aspects. A further quantitative survey of 330 successful Android developers then identified a baseline of current security practices in software development.

Based on these surveys, the author created an intervention package to help software developers. Action Research techniques were used to trial and improve it in two one-year cycles with a total of 19 development teams in 11 different organisations. The later development of the package concentrated on empowering the developers involved, and reducing the involvement required from the researchers.

By proving that a set of structured workshops can have an impact on the security performance of a team for a reasonable cost and without the support of security professionals, this research offers a powerful means to enhance development security in the UK, creating more secure software and systems for all users.

Acknowledgements

Thank you to my supervisors Lynne Blair of Lancaster University, James Noble of Victoria University at Wellington, and Awais Rashid of Bristol University.

Thank you also to my PhD appraisers Dan Prince and Kim Kaivanto of Lancaster University.

Thank you to Ingolf Becker of UCL, who worked with me on the two Majid projects as the second coder for the data, and as a contributor of ideas and some text.

Thank you to Katherine Whitehead of Lancaster University who did all the manual transcription required for the work described in this thesis.

Thank you to Sascha Fahl, Christian Stransky and Dominik Wermke of Leibnitz Universität Hannover, who worked with me on the Android Developer Survey. And to Yasemin Acar also of LU Hannover, who provided valuable advice on the survey implementation.

Thank you to M. Angela Sasse, of Ruhr Universität Bochum, who provided expert advice on the Action Research method.

Thank you to Tamara Lopez of Open University, who reviewed and provided expert feedback on the Online Survey questions.

Thank you to Ian White, of UCL, who advised on the statistics used to analyse the online survey.

Thank you to Thomas Gross of Newcastle University, whose Workshop on Refining Evidence-Based Methods strongly influenced the online survey method.

Thank you to the over a dozen anonymous reviewers of the papers, who have all contributed significantly; and particularly to USENIX shepherd Daniel Zappala of Brigham Young University.

Thank you to Steve Wright of Lancaster University, who taught the NVivo techniques used to analyse the two Magid projects, and also found Sonix.

Thank you to Kayla Friedman of Cambridge University and Malcolm Morgan of Leeds University for producing the Microsoft Word thesis template on which the Lancaster template is based [187].

Thank you to all the interviewees in the Expert Survey, whose names must be confidential, who gave their time to provide the information in the Interventions survey.

And lastly, thank you to all the many teams of developers and security experts, whose names must also be confidential, who trialled the interventions, and contributed ideas and feedback.

Contents

List of Tables

List of Figures

List of Abbreviations and Acronyms

ACM	Association for Computing Machinery
AES	Advanced Encryption Standard
et al.	And further authors.
API	Application Programming Interface
CAR	Canonical Action Research
CEO	Chief Executive Officer
CISO	Chief Information Security Officer
CLASP	Comprehensive, Lightweight Application Security Process
CMM	Capability Maturity Model
CONSORT	Consolidated Standards for Reporting Trials
CTO	Chief Technical Officer
DBR	Design-Based Research
DCS	Developer Centred Security
ECB	Electronic CodeBook, an encryption mode.
ESE	Empirical Software Engineering
ESEC	European Software Engineering Conference
FST	Faculty of Science and Technology, at Lancaster University
FTSE 100	Financial Times Stock Exchange 100 Index, a list of some of the largest companies.
GCHQ	General Communications Headquarters, the main United Kingdom government security body.
GDPR	General Data Protection Regulation, a European Union statute.
GIS	Geographical Information System
GT	Grounded Theory
HTTPS	Hypertext Transfer Protocol over SSL, an encryption-based Internet security protocol.
IBM	International Business Machines, a company.
IDE	Interactive Development Environment
IEC	International Electrotechnical Commission, an international standards body.
IEEE	Institute of Electrical and Electronics Engineers
IRR	Inter-Rater Reliability
ISO	International Standardization Organization
IT	Information Technology
LLC	Limited Liability Company
NCSC	UK National Cyber Security Centre, a group within GCHQ
NOSQL	Non-SQL (a non-relational database).
OPAL	OPen Analysis Library for Java Bytecode
OWASP	Open Web Application Security Project, a non-profit organisation.
Pen Test	Penetration Test, a simulated cyberattack on a computer system.
QA	Quality Assurance (software testing)
SDL	Secure Development Lifecycle
SIGCAS	ACM Special Interest Group on Computers and Society
SIGPLAN	ACM Special Interest Group on Programming Languages
SIGSAC	ACM Special Interest Group on Security, Audit & Control
SIGSOFT	ACM Special Interest Group on Software
SME	Small to Medium Enterprise

SPI	Software Process Improvement
SQL	Structured Query Language, for databases
SSL	Secure Sockets Layer, an encryption-based Internet security protocol.
TLS	Transport Layer Security, an encryption-based Internet security protocol.
UCL	University College London, a university.
UNIX	A family of operating systems
URL	Uniform Resource Locator (a web address)
USB	Universal Serial Bus, a standard for connecting computers and peripherals.
USENIX	An association that supports operating system research (from Unix)
USP	Unique Selling Proposition
VBA	Visual Basic for Applications, a programming language.
VPN	Virtual Private Network, a secure Internet connection between two devices or sub-networks.

1 Introduction

Software increasingly affects everyone and everything we do. It contributes enormously to our quality of life and to our ability to communicate, learn, control and build. Software underpins virtually every form of business, much of our social interaction, and increasingly the behaviour of our homes and machines. Indeed, software is probably the most significant contributor to societal change and improvement in the past fifty years [1].

Yet with the vast benefits of software come problems: the problem of software not behaving in the way desired ('faults'); the risk of people finding ways to misuse software in ways detrimental to its intended users ('security'), and the risk of revealing information in damaging ways ('privacy').

Faults have been a familiar issue in software development from the beginning, and researchers have devoted much effort to finding effective ways to address them. Even security and privacy are hardly a new problem; banks have been using software since 1950 [21]. But security and privacy have only began to become issues in mainstream software development in the last twenty years, and only in the last five years has there become to be public awareness of the problem.

And they are now becoming major issues: almost every day we hear that several more organisations' software systems have been compromised [142]. While there are many aspects to an organisation's security and privacy, the design and implementation of the software used clearly has a significant impact on whether such breaches happen.

Two industry trends contribute to this. First, changes in web architecture such as microservices and the increasing integration of Software as a Service (SaaS) components into systems mean that perimeter security is increasingly irrelevant, making security more a feature of the developed code and therefore the responsibility of the developers. Second, the DevOps (Development Operations) movement means that security issues that used to be the province of a separate operations team, such as the management of encryption keys and deployment passwords, increasingly is becoming the responsibility of software developers. Therefore, the effectiveness of developers at creating secure software is vital.

Unfortunately, there is evidence that developers are not delivering this security. A recent report from Veracode [177] concluded that *"more than 85 percent of all applications have at least one vulnerability in them; more than 13 percent of applications have at least one very high severity flaw"*. A report from Microsoft [115] analysed storage and collaboration Software as a Service applications, and found a wide range of issues, including 28% of storage apps not supporting data encryption.

All these surveys suggest that many errors were avoidable; developers could have made choices that would have prevented the issues. Yet the developers were just doing their job, subject to their constraints, and there is little evidence of deliberate or even careless connivance at security errors. Indeed, the Ponemon Institute carried out a IBM-funded survey in 2015 of 640 individuals from organisations developing mobile apps in the US [135], and found that 77% believed that *"securing mobile apps is very hard"*, and that 73% percent believed that developer lack of understanding of security issues was a major contributor to the problem.

This demonstrates that existing industry practices are insufficient to provide the application security and privacy we need. So, how can one support developers to deliver better security?

1.1 State of the Art and Its Limitations

In industry, the international not-for-profit Open Web Application Security Project (OWASP) organisation runs regular events and provides web-based resources [206]. It delivers excellent, data-driven advice and materials, but is dominated by security professionals, and gets only limited engagement with software developers[1]. The SANS institute is a commercial organisation with a similar function in the USA and similar limitations. SANS is noted for its commercial training, and also for a comprehensive online library of security resources created by security professionals [146].

There is increasing academic research happening in this area. Several research groups are looking at technical aspects of secure programming (APIs, user interactions) For example, teams in the universities of Bonn and Hannover have been investigating Android app developer behaviour; a team in the University of New South Wales is exploring the 'process usability' issues faced by developers. Former leading players like McGraw [112] are now working at CEO level; most security specialists work within practicing teams. In the UK, the Johnny project [139] is exploring security for solo developers; and the Jenny project [108] has looked at security for unsupported development teams.

Survey evidence (see Chapter 5) suggests that few developers have any formal security training or experience. Even if some know how to achieve security, how are they to get a whole team to follow?

[1] Personal communication, Martin Knobloch, OWASP Chairman, OWASP AppSec Europe Conference 2018

1.2 Conventions in this Thesis

Many research questions distinguish ‘privacy’ (protection against unwanted disclosure of information to other users of a system[2]) from ‘security’ (protection against malicious theft, corruption, faking of information, or denial of service). This thesis uses the word ‘security’ to include both.

A software development team usually includes a range of roles: programmers, testers, project managers and product managers. Here, we use the word ‘developer’ to refer to all these roles, using the more specific terms where appropriate. Similarly, the responsibility of deciding development priorities and the allocation of financial and other resource might be have the title of product manager, line manager, technical lead—or indeed might be the responsibility of committee. Here we use the term ‘product manager’ to refer to any holder of this role.

1.3 Research Objective

Summarising the previous discussions, there is a need for better software security; developers are the main cause or otherwise of that software security; and few developers have any security training or formal security experience.

The background of the of the author was mentoring and managing software development teams in both small and large organisations. In that role he found that there was little support available to help product managers and development teams with security, whether in the form of books or professional advice. He entered research to find a way to address that problem.

Indeed, the scope of the problem is huge. There are some 200,000 software developers in the UK [159], and perhaps many more times that number worldwide.

So, this research project has the aim of finding some way to support a worthwhile proportion of those developers in delivering better software security. In the terms of the UK National Cyber Security Centre (NCSC), what is needed is an ‘intervention’ to provide that support[3]. Since a large majority of developers work in teams [157], it is reasonable to limit the intervention to team-based developers. Given the huge population of developers in the UK it is unnecessary to look further afield, which avoids the complexities of different cultures and languages. And finally, if many development teams are to adopt such an intervention, it must be cost-effective and usable in a wide range of organisations and situations.

The research question for this thesis, therefore is:

RQ 1. *What is needed to make a cost-effective and widely applicable intervention to help UK software development teams achieve better software security?*

The remainder of this introduction explains how this question was addressed, defining further research questions as required.

[2] Other aspects of privacy are outside the scope of this thesis.

[3] Helen Lovekin, NCSC, personal communication 2017.

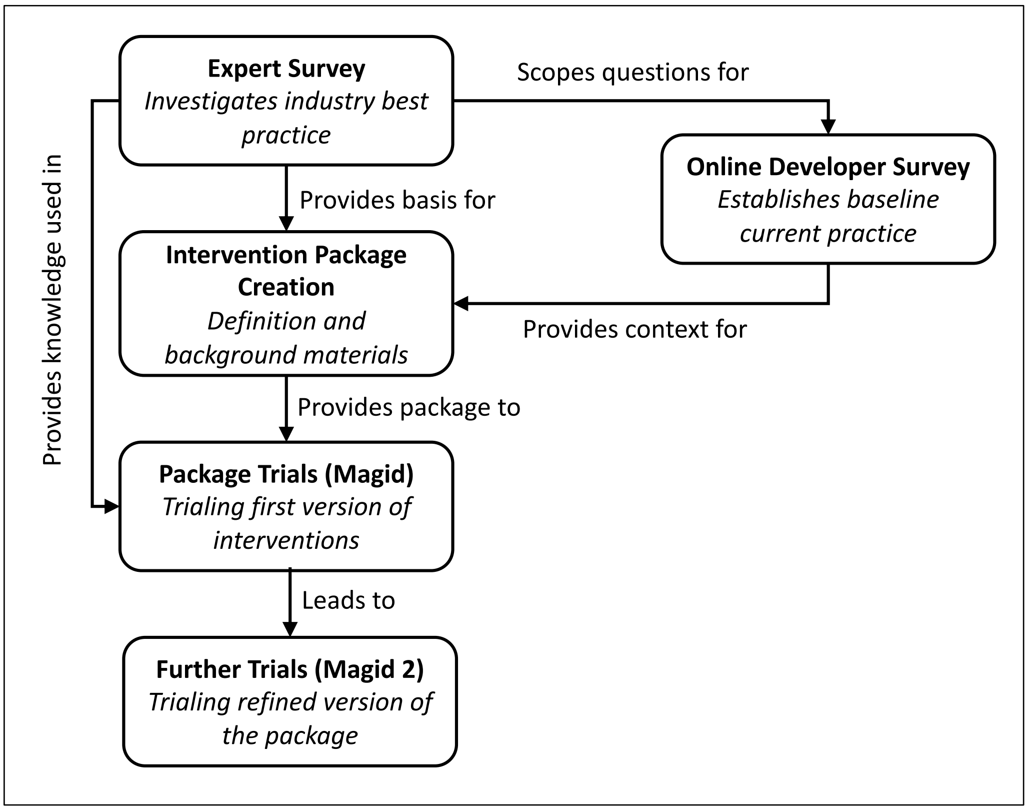
Expert Survey
Investigates industry best practice
Scopes questions for
Online Developer Survey
Establishes baseline current practice
Provides basis for
Provides knowledge used in
Intervention Package Creation
Definition and background materials
Provides context for
Provides package to
Package Trials (Magid)
Trialing first version of interventions
Leads to
Further Trials (Magid 2)
Trialing refined version of the package

Figure 1: Overview of the Five Research Projects

1.4 Research Summary

This question makes this a topic in the field of Empirical Software Engineering [49]. The research started without preconceptions of the forms that interventions might take. Possibilities envisaged included a tool, a mentoring program, an online course, a book, a website, or even contributions to an audio or video program [190].

The wide scope of the question required a variety of different research techniques, and therefore the research involved five different 'projects'. Figure 1 shows the interrelation of the different projects, which are described in the following sections.

1.4.1 Expert Survey

First, to generate theory and establish best practice, an 'Expert Survey' of face-to-face interviews with sixteen experts at improving software in development teams, asked open-ended questions about what interventions they considered most successful. The research question for the Expert Survey was developed in conjunction with Helen Lovekin from GCHQ; it expands on RQ 1; and emphasises the identification of interventions:

RQ 2. *What interventions can change the environment for members of the development team to achieve good security, considering cost-efficiency, motivational factors, choice of tools, supporting processes, culture, awareness, training and skills?*

The analysis approach was Grounded Theory, and the main theoretical conclusion was that effective developer-centred security requires 'active developers', who themselves drive the security improvement rather than having it imposed on them. A further outcome was the identification of eight techniques the experts stated they found most effective for security improvement.

1.4.2 Online Developer Survey

To provide proof of the need for an intervention, and also a baseline of existing security practice in commercial development teams, an online Developer Survey was undertaken in collaboration with groups in the universities of Hannover and Paderborn. Pragmatically, the chosen participants were Android app developers worldwide, since both the developer contact details and their delivered software are publicly available. The research question for the survey was the following:

RQ 3. *To what extent, and how, does a perceived need for security and privacy lead to security-enhancing activities and interactions in an Android development team and result in better software security?*

The results showed that around half of Android developers worked in teams, most self-identified as valuing security, they used a wide variety of development stacks; but that few had access to security professionals nor used the assurance techniques identified in the Expert Survey. They also suggested that security improvements were driven more by developers than forces external to the team.

The Developer Survey also found correlation between the need for security and both the assurance techniques used and the security update frequency. However, there was little correlation between these factors and the measured security attributes of the software produced.

1.4.3 Intervention Package Creation

Returning to the primary research question, the Expert Survey and the Developer Survey provided a context for designing interventions. Specifically, the Expert Survey suggests that to be effective an intervention must:

- Motivate ‘active developers’ to drive their own security improvements, and
- Convey a knowledge of most cost-effective security assurance techniques to use.

And the Developer Survey suggests that to have a wide appeal an intervention must:

- Work with development teams, but not require security specialists, and
- Be independent of specific development environments or domains.

The next step, therefore, was to construct such an intervention. The author had expected it to take the form or a code analysis tool, a website, a book or a training course; in practice excellent versions of all of these exist already to support developers creating secure code—yet the need for an intervention still remains.

Instead, the author created a package of workshops, ‘Developer Security Essentials’, requiring less than a day’s effort for a team. This involved two team workshops: a game using a case study to teach that security is a commercial decision; and a Threat Assessment to identify security requirements for the participants’ own projects. An online book, video, and materials [182] supported the package.

1.4.4 Package Trials (Magid)

To test the proposition that the Developer Security Essentials intervention does indeed satisfy the criteria of RQ 1 by being *cost-effective and widely applicable* and by *helping UK software development teams achieve better software security*, the 'Magid' project explored the results of using it with professional developers in UK companies. The project used an 'Canonical Action Research' method [24], addressing the research question:

RQ 4. *What security outcomes did the 'Developer Security Essentials' package have, and what aspects contributed most to those outcomes?*

Development teams were recruited in three different UK companies, and the Developer Security Essentials workshops were led by the author. Recordings were made of interviews of a range of participants before and after the intervention, along with the discussion in the workshops themselves. These were then double coded for indications of improvements in security practice or knowledge, for candidate improvements for the package itself, and for indications of the issues and solutions involved in improving security. The coded dataset was then analysed both qualitatively, for examples and indications of what had happened; and quantitively to provide an indication of the frequency of different effects.

The results suggested that the package was effective in improving development security in the two less experienced development teams, and in improving communication in the other. They also suggested two new and important requirements:

- For others to deliver the intervention instead of the researcher, and
- For developers effectively to present possible security improvements to product management, to allow informed decision making on where to spend effort and money on security.

1.4.5 Further Trials (Magid 2)

The author therefore modified Developer Security Essentials to satisfy these new requirements, by providing sufficient materials and instructions for others to deliver the intervention, and by adding a further experimental 'Threat Sales' workshop to support developers 'selling' security mitigations to product management.

The effects of this change were tested in a second cycle of trials: the 'Magid 2' project. Canonical Action Research was rejected as methodology since it requires the same participants for each cycle; instead the project used Design-Based Research, which supports different participants in each cycle of trials, focusses on designing an artifact, and still can use Action Research techniques.

The research question reflects an emphasis on making the intervention potentially scalable to use by large numbers of organisations:

RQ 5. *Which aspects of the 'Developer Security Essentials' intervention are effective at improving security when used independently by teams from a variety of cultures and different types of organisation, and why?*

Teams from a total of eight further organisations used the new version of the package; and this time the author's involvement was limited to a training session for one or two facilitators taken from the development team, a brief overview presentation, and contributions to the discussions as a participant.

Table 1: Overview of Research Questions

RQ 1	What is needed to make a cost-effective and widely applicable intervention to help UK software development teams achieve better software security?
RQ 2	What interventions can change the environment for members of the development team to achieve good security, considering cost-efficiency, motivational factors, choice of tools, supporting processes, culture, awareness, training and skills?
RQ 3	To what extent, and how, does a perceived need for security and privacy lead to security-enhancing activities and interactions in an Android development team and result in better software security?
RQ 4	What security outcomes did the 'Developer Security Essentials' package have, and what aspects contributed most to those outcomes?
RQ 5	Which aspects of the 'Developer Security Essentials' intervention are effective at improving security when used independently by teams from a variety of cultures and different types of organisation, and why?

A further change was in the evaluation of the intervention's impact. The analysis looked for and quantified improvements in 'assurance techniques': process improvements, understanding and skills that would generate better security in the longer term.

The results showed improvements in understanding or adoption of security assurance techniques in all but one of the teams. Two organisations chose to carry out further workshops independently from the researchers, and the teams in five organisations showed major improvements in their security decision making.

In particular, the results highlighted a value to presenting security enhancements in terms of their business benefits, and proved the ability of developers to do so with minimal guidance.

1.5 Novel Contributions

Table 1 summarises the research questions introduced over this section. Since research questions RQ 2 through RQ 5 were still relatively broad, each was broken down into sub-questions; these sub-questions are introduced and answered in the chapters discussing the corresponding projects.

The research makes the following contributions:

- Identification of the importance that developers drive security improvements themselves (the Active Developer Model).
- Evidence of very limited use of security assurance techniques by developers in a particular domain (Android Apps).
- An existence proof that a simple 'intervention package' structured as a facilited series of workshops can improve the security of software developed by a team.
- A new use of Design-Based Research, in the field of Developer-Centred Security
- Identification of the importance of representing security enhancements in terms of their business benefit, and the ability of developers to do so.

1.6 Thesis Overview

The following chapters in this work describe this research in detail. Specifically, the chapters are as follows:

Chapter 2	explores existing peer-reviewed and 'grey' literature. It explores the state of knowledge and previous research on security as applied to software development teams; plus some key work in related areas such as programmer motivation and the involvement of security professionals.
Chapter 3	provides an overview and justification for each of the methodologies used in the research.
Chapter 4	describes the 'Expert Survey' of security practitioners working with software developers. It provides detail of the approach used, summarises the interviews; and describes the main result of 'active developer security' and eight specific techniques used by the practitioners.
Chapter 5	describes the online 'Developer Survey' of successful (Android) software developers, establishing a baseline for developer knowledge of security.
Chapter 6	describes the 'Intervention Package Creation' of a suite of facilitated workshops for a development team and the materials created to support those workshops.
Chapter 7	describes the 'Package Trials (Magid)' project, trialling this package with development teams in three different organisations. It introduces the participants in detail and describes the results and analysis of those trials.
Chapter 8	describes the 'Further Trials (Magid 2)' project, with improvements to the package trialled in eight further organisations. It describes the changes to methodology based on learning in the first cycle, and describes the results, analysis and conclusions reached.
Chapter 9	discusses conclusions, both in terms of practical results and improvements to the package and in terms of wider theory gained. It contrasts work by previous researchers, and outlines a range of further work to take the research forward.

2 Literature Review

This chapter provides an overview of the academic and related literature associated with this thesis topic.

As a researcher in the field of Developer Centred Security (DCS), the author wanted an overview of the whole field with an indication of the relative importance of different topics. No current publications provide this, and so the first section of this chapter uses a rigorous method to address this lack.

The second section of this chapter reviews research related to the specific topic of this thesis: interventions for software developers.

2.1 Developer Centred Security Literature

This section provides an overview to introduce the reader to the discipline of DCS, with a rigorous analysis of the most important papers and books.

2.1.1 DCS Review Method

The scope of this review was peer-reviewed or professionally edited publications related to software developers' implementation of security or privacy in their code: Developer Centred Security. This included developer behaviour *related to security*, developers' *use* of static analysis and testing tools, and developer *use* of cryptography. Related topics were excluded, such as the *design of* static analysis and testing tools, *non-security-related* developer behaviour, the *implementation* of cryptography, and the work (including vulnerability analysis) of software security professionals. Identifying the works likely to be 'important' to a researcher in the field suggested the use of citation counts.

The review started with an extensive list generated in the course of the research project. While no systematic search was carried out, the list includes the papers from a previous systematic search [165] plus all DCS papers identified by an AI-based recommendation service, so it is unlikely any highly-cited papers have been omitted. For each paper, the author then assigned the citation count taken from Google Scholar.

Using citation counts alone strongly favours older publications that have had more time to accumulate citations. Instead, observe from Bai et al.'s chart of annual citation rates

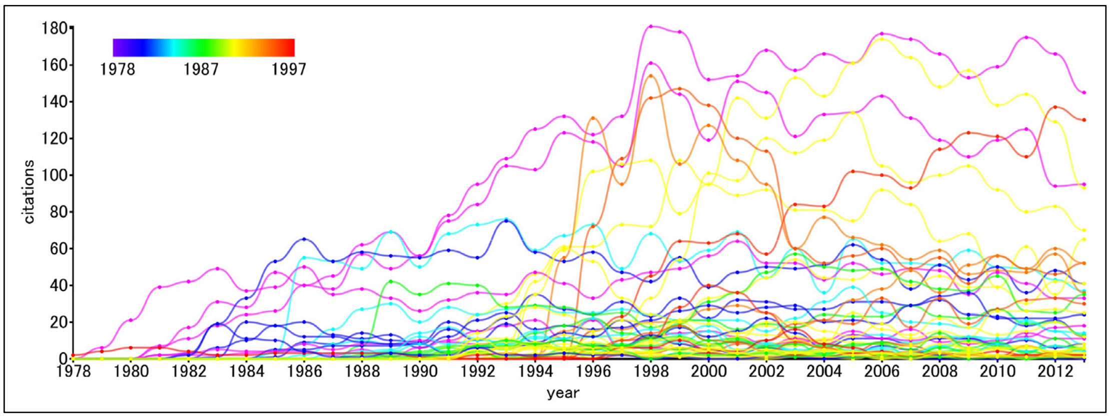

Figure 2: Annual Citations of a Range of Papers (Bai et al. [17])

for randomly selected papers in Figure 2 that the distribution of citation rates approximates to constant for a given paper after a certain time [17], and therefore the annual citation rate is a meaningful figure. Given the relative youth of the discipline of Developer-Centred Security, it is important to give weight to recent papers, so in this review, the 'important' selection criterion was interpreted as 'having a high annual citation rate'.

A total of 173 peer-reviewed or professionally edited publications were found in the field of Developer Centred Security. These had between zero and 298 average citations per annum, with a median count of 6. From this set, 37 publications were chosen as 'important': having more than 10 citations per year.

To provide context to discuss the papers, they were categorised into topics as follows:

Static Analysis	Research on developers' *use of* tools for the security analysis of both source code and binaries, including the adoption of such tools.
Literature	Information on secure development for use by developers, and analysis of how such information is used.
Requirements	Research into ways to identify security requirements, including Threat Modelling.
Developer Behaviour	Research into developer behaviour related to creating secure software.
Roadmaps	Suggestions for future Developer Centred Security research.

To check the sensitivity of this selection criterion of 10 citations per year, the range of papers with 9 and 8 citations were also reviewed to see if they added further topics; they did not. The author also checked whether there were any unexpected omissions from the list; there were none.

2.1.2 The Key DCS Publications

Figure 3 shows the literature selected. It shows the publications organised by date and topic. Red text indicates particularly highly referenced publications—with more than 40 citations per year)—and italics indicate published books. 'Et al' is abbreviated to '+' throughout. A surprising proportion of the research dealt with mobile app development and these publications are indicated in navy blue. Each paper is described in Sections 2.1.3 to 2.1.7, and the alphabetical list can be found in Appendix A.

	Static Analysis	Literature	Requirements	Developer Behaviour	Roadmaps
Pre-2005		Yoder&Barcalow98			Devanbu&Stubblebine00
2005		*Schumacher+05*	Sindre&Opdahl05		
2008	Ayewah+08	*Steel+06* Yoshioka+08 *Anderson08* *Howard+09*			
2011			Faily&Flechais11		
2013	Johnson+13			Fahl+13	
2014	Xiao+14		*Shostack14*	Balebako+14 Balebako&Cranor14	
2015				Nadi+15	
2016	Christakis&Bird16	Acar+16	Perera+16	Green&Smith16 Yang+16	Acar+16a
2017	Derr+17 Nguyen+17	Acar+17a Fischer+17		Naiakshina+17 Acar+17 Hilton+17	
2018			Tuma+18	Votipka+18 Assal&Chiasson18	
2019			Xiong&Lagerström19		

Figure 3: Frequently Cited Developer Centred Security Literature, by Date

Note that the vertical axis is not linear; earlier dates are spaced more closely than later ones to better represent the increasing amount of research taking place recently.

The following sections explore each topic in turn: Static Analysis, Literature, Requirements, Developer Behaviour and Roadmaps.

2.1.3 Research into Static Analysis for Security

One approach to improving software security is to use a ‘Static Analysis’ tool to identify possible security vulnerabilities.

Though a 2008 journal article mentions finding security bugs amongst other problems, using a static source code analysis tool at Google [15], in practice as Figure 4 shows that was only the start of a huge amount of research into static analysis tools [145]. Indeed,

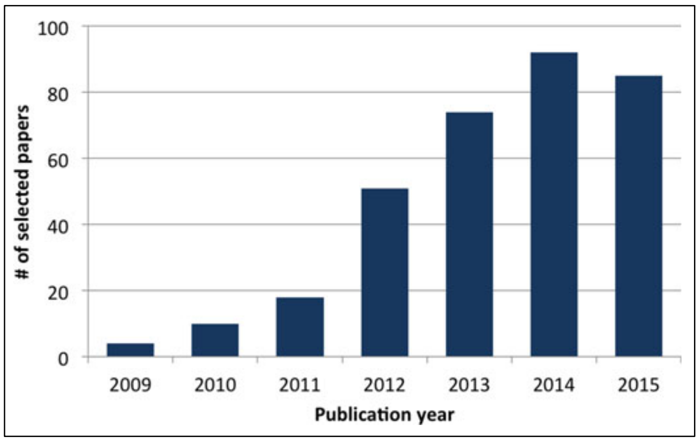

Figure 4: Android Static Analysis Papers (Sadeghi et al. [145])

the 3624 citations for Enck et al.'s 2010 paper on TaintDroid [54], and 1444 citations[4] for Arzt et al.'s 2014 paper on FlowDroid [11] suggest that this research has continued with similar intensity until now.

However, the vast majority of this research considers only the effectiveness of tools created by researchers at detecting issues in source and binary code. Developer Centred Security is concerned with how developers interact with these tools.

A survey by Johnson et al. analysed 'Why don't software developers use static analysis tools to find bugs' and produced a set of recommendations for tool functionality; in particular the ability to avoid repeated false positives and support for 'quick fixes' [92]. Christakis and Bird [34] surveyed Microsoft developers' opinions about such tools, finding that they consider security defects the most important for a code analysis tool to find, and that the key features they need are relevant results, speed, and the ability to suppress earlier warnings in incremental changes.

Xiao et al. [199] interviewed around 50 developers to quantify the factors that caused them to adopt automated security tools, finding that recommendation by trusted peers, even those only known through social media such as Stack Overflow, was the main reason. Nguyen et al. [123] explored the impact on Android developers of integrating a source code analysis tool into their editing environment, concluding a high value for '*quick fixes*': changes requiring little effort on the part of the programmer. And Derr et al. [45] surveyed around 200 Android developers to find the extent to they keep library versions up to date, finding huge scope for solving vulnerabilities by library updating without changes in code, but that frequent backward incompatible changes and incorrect Semantic Versioning in libraries currently make such updates problematic.

2.1.4 Research into Literature for Developers

Some 20 years ago the Software Patterns movement provided a mechanism and community to include developing knowledge in academically acceptable formats. A few of the pattern authors considered aspects of software security. An early patterns paper by Yoder&Barcalow describes a range of software architecture techniques to achieve

[4] Google Scholar figures for 15 March 2020. Google appears to have conflated the figures for the initial 2010 USENIX TaintDroid paper with a later extended journal version, but the number still suggests a huge research population.

security functionality [203][5]. Schumacher at al. later composed several such papers to create the book 'Security Patterns' [148]. And Yoskioka et al. later produced a survey of the security patterns literature, positioning it in the context of wider security and software methodology work [204].

Several foundational DCS books date from 2006-2009. Steel et al., though not participants in the software patterns community, produced a book with 'Patterns' in the title describing techniques for Java security architecture [160]. Howard et al. published the book 'Deadly Sins in Software Security' (the number of sins varied with the edition), containing practical support for developers to avoid coding security issues [88]. And Anderson published 'Security Engineering', which provides guidance at a higher level than code, covering economics, usability, architecture and development process [9].

More recently, a group in L.U. Hannover studied how developers learn from such information. Acar et al.'s 'You get where you're looking for' [3] uses several surveys and experiments to explore the effectiveness of different information resources at helping solve security issues. From a survey of nearly 300 successful app developers worldwide they concluded that developers learned security using web search and from peers. From a practical experiment with over 50 Android developers to evaluate the effectiveness of several different ways of learning app security they found that copying from Stack Overflow led to a variety of security problems, and also the surprising result that programmers using only digital books achieved better security than those using web search.

Fischer et al. explored copying from Stack Overflow in more depth, concluding that about 15% of apps contain insecure code introduced in this way [60]. Acar et al.'s 'Developers need support, too' explores the online guidance available to developers[6], identifying a need for resources written for other people than specialist software security experts [6].

2.1.5 Research into Security Requirements

Four well-cited publications suggest ways to elicit security requirements. Sindre & Opdahl propose 'misuse cases', as a way of drawing on the popular 'use case' methodology to address security [155]. Similarly, Faily & Flechais took the 'Persona' concept from user-centred design to provide a method to analyse the motives and approach of attackers [58]. Shostack's book 'Threat Modeling', though following Microsoft's tradition of 'Threat Modeling' meaning finding vulnerabilities rather than assessing threats, also offers developers valuable advice on techniques for security requirements [154]. And Perera et al. took earlier work identifying generic privacy requirements to create and demonstrate a method to identify privacy issues within IoT infrastructure [131].

In the last couple of years, two parallel[7] systematic literature reviews have explored threat assessment. Tuma et al. categorised the papers they found, finding a majority of the work to be about applying threat modelling, with a minority describing specific threat modelling methods [170]. Xiong & Lagerström analysed the specifics of individual papers finding most threat modelling techniques to be manual and graphically based

[5] Barcalow, the industry author, worked at that time at the same group in Reuters in Chicago as Charles, one not noted for security requirements!

[6] Charles is a co-author on this paper.

[7] Neither paper cites the other.

[201]. Both observed a lack of automation in the process and a lack of means to quality assure the outcomes.

2.1.6 Research into Developer Security Behaviour

This topic has only received research effort relatively recently: focusing on aspects of the software developer's job and investigating how improvements and changes might help support more secure code.

Fahl et al. used several forms of research with developers to evaluate how SSL problems could be reduced in mobile apps, proposing changes to the Android SSL framework [57][8]. Balebako et al. surveyed and interviewed developers about implementing privacy needs, finding widespread ignorance of the privacy implications of third party libraries, especially in small companies [20]; Balebako & Cranor suggested several possible ecosystem improvements to encourage the implementation of effective app privacy [19].

Nadi et al. investigated developers' experiences with cryptographic APIs, concluding that most APIs were too low level for practical use [117]. Green & Smith explored the implications of this, advocating 'developer-friendly security' as a target [76].

Other researchers have explored different aspects of software development. Hilton et al. explored Continuous Integration, highlighting the security threat of attackers changing delivered code [85]. Votipka et al. investigated the extent to which software testers, rather than specialist penetration testers, might find vulnerabilities; they found that, instead, security training and experience was key to finding such defects [180].

There has been some research into how developers handle security aspects of programming. Yang et al. explored the security questions developers ask on Stack Overflow, finding more than half related to Web Security, and particular interest in passwords, hashing, signing and SQL injection [202]. Naiakshina et al. used a small-scale study with students to investigate the causes of developer errors in password storage, recommending nudges: asking for security, and secure-by-default frameworks [118]. And Assal & Chiasson surveyed developers in industry to explore their security practices, finding that corporate pressures meant that few kept to the best practices defined in the Secure Development Lifecycle literature [13].

Finally, there is the possibility of paving the way for future research. Acar et al. investigated the validity of experiments on student participants rather than professionals, using a programming study with Python open source developers, and finding the determinant to be participants' experience using the language, and not their status as a student or professional, nor their security experience [5].

2.1.7 Roadmaps

Lastly, two frequently cited papers propose roadmaps: justified lists of topics for future DCS research. Devanbu & Stubblebine's 2000 paper explores topics of relevance to software engineering, exploring a range of different issues and possible approaches to their solution, and highlighting five areas for research: cost-benefit analysis of security requirements; architectures to support changing security policies; models of adversary behaviour; verification tools; and infrastructures for the security administration of deployed systems [46]. Acar et al. in 2016 explored existing research and suggested a

[8] Google subsequently implemented these changes.

three-part research agenda: how to do experiments on developers; influences on developer behaviour; and improvements in APIs, documentation and tools [4].

2.1.8 Limitations of this Review

To conclude, though this review approach provides a good overview to the discipline, there are limitations in the selection criteria.

The number of citations is an inaccurate way to estimate the importance of a paper to Developer Centred Security: citations may be for a variety of reasons and do not indicate that the citing papers were in the same field; and citations do not necessarily prove quality.

Another limitation is that using citations excludes very recent papers that have had insufficient time to be cited.

2.2 Security Interventions Literature

The reader will observe that the 'mainstream' Developer Centred Security research identified by the review in the previous section included little work on interventions to improve developer security behaviour. One paper did suggest API changes to better support SSL development that have since been adopted [57]; another suggested a set of guidelines for cryptographic APIs [76]; a third provided advice to include security in requirements and to make security APIs opt-out only [118]; but none have provided validation of the effectiveness of carrying out their suggestions.

This section, therefore, reviews literature applicable to the research topic of this thesis: interventions. That is both narrower than the selection criteria for the review in the previous section, in that few widely cited DCS papers deal specifically with interventions; and wider, to include interventions and related developer behaviour research not related to security. Accordingly, this section defines a range of topics, and reviews publications related to each topic.

Research related to interventions for secure software has taken a variety of approaches. In this section, we explore several areas in turn: research into security requirements; ways to get developers to adopt process improvements; ways to get developers to adopt analysis tools; consultancy and training interventions; using interactions with security specialists; motivating developers to improve their processes; and motivating employees more generally to adopt secure practices. Each section also outlines omissions and opportunities arising from that area of research so far.

2.2.1 Security Requirements

There has been further research into security requirements, especially related to privacy. A literature survey by Türpe [171] found a range of research related to security requirements, mainly exploring Threat Modelling techniques, but no agreement on terminology or approach. Senarath and Arachchilage [150] used a programming task given to 35 developers to explore issues related to user privacy, finding it to be difficult to understand such requirements and translate them into engineering techniques, and recommending solutions in the specification of privacy requirements. Similar research by the same authors [151] found that developers use their own privacy expectations to guide software privacy decisions; these differ from the expectations of non-developer users, though the authors point out there is no easy solution for this problem.

Other researchers have explored requirements for security updates and fixes. Nayak et al. [122] found that less than 15% of known vulnerabilities were actually used in attacks, suggesting an opportunity for a more nuanced approach than just fixing everything. Vaniea and Rashidi [176] used a survey to analyse user thinking around the update procedure, deriving advice for developers planning such mechanisms including a recommendation for a ‘recovery path’.

A few researchers have developed means to gamify the finding of security requirements. Shostack’s Elevation of Privilege [154] concentrates on technical issues and vulnerabilities; Denning’s Security and Privacy Threat Discovery Cards [149] concentrate more on high-level security assessment; and Merrill [113] successfully used a role-playing game with professional developers; none have been evaluated in any detail.

One may conclude from this and the analyses in Sections 2.1.3 and 2.1.5 that there is a need for ways to support developers in determining wider security needs, finding effective ways to identify both a Threat Assessment, and wider security-related requirements.

2.2.2 Adoption of Security-Enhancing Activities

To achieve such motivation and culture changes, one possibility to change development processes and there has been significant research into applying such changes to software security improvement. Indeed, prior to about 2010 the accepted way of improving software security was a ‘Secure Development Lifecycle’ (SDL), a prescriptive set of instructions to managers, developers and stakeholders on how to add security activities to the development process. A paper by De Win et al. compares the three major SDLs of that time, OWASP’s CLASP, Microsoft’s SDL and McGraw’s Touchpoints, contrasting their features in the context of a simple project [197]. However, other research from that time suggests resistance from development teams to adopting a prescriptive methodology. For example Conradi and Dybå found in a survey that developers are sceptical about adopting the formal routines found in traditional quality systems [37]; others came to the same conclusion [83,103,141]. Indeed Geer’s online survey of 46 developers recruited from those already specialising in secure software development found only 30% of them using SDLs [70]; Xiao et al.’s later survey of 40 developers [199], found only 2 using them. While these sample sizes were fairly small, the findings provide a plausible explanation for the abandonment of SDLs. Since 2010, SDLs have been replaced in the research literature by ‘Security Capability Maturity Models’, such as BSIMM [207], which measure the effectiveness of corporate security enhancements rather than mandating how they are achieved[9].

Caputo et al. [32] used three case studies to explore several theories about what changes in software development might lead to more usable security, concluding a need for the alignment of security goals with business goals. Recently Assal and Chiasson [13] interviewed developers from 13 different teams and organisations about their security practices, concluding ‘a need for new, lightweight best practices that take into account the realities and pressures of development’. Van der Linden et al. found from a task-based study and survey [107] that developers tend to see only the activity of writing code to be security-relevant, suggesting a need for a stronger focus on the tasks and activities surrounding coding.

Taking a different approach, Such et al. investigated the economics of software security, surveying 150 security specialists to analyse the economics of different assurance

[9] The author is aware, however, of several multinational companies still successfully using SDLs.

techniques [164]. The survey defined a taxonomy of twenty assurance techniques and found wide variations in the perceived cost-effectiveness of each. It found Public Review and (tool-based) Static Analysis to be the most cost-effective, and Formal Verification and Cryptographic Validation to be the least. Interestingly the researchers also identified frequently used combinations of techniques, finding that the combination of Architectural Review, Configuration Review, (manual) Penetration Testing and (automated, web-based) Vulnerability Scan was seen as most cost-effective.

These results suggest a need for more work exploring lightweight, cost-effective, enhancements to development practices to improve security.

2.2.3 Encouraging the Adoption of Tools

In addition to the work described in Section 2.1.3, there has been other research on how to support developers in adopting static and dynamic security analysis tools. Though insufficiently well-cited to qualify in the earlier survey of key DCS papers, these are particularly relevant to the topic of this thesis.

Witschey et al. [198] suggested that adoption of tools could be modelled by Rogers' Diffusion of Innovations theory [143], and used a survey of 40 developers recruited opportunistically to explore the model. They concluded that more experienced, and more inquisitive, developers are more likely to adopt tools, and that key deterrents were the difficulty of trialling new tools, and their invisibility – that developers are unlikely to notice a colleague using one.

Xiao et al. [199] reported a similar Diffusion of Innovations study, interviewing 40 professional developers to explore the social factors that led to security tool adoption. They found the main reason for adoption was recommendation by trusted peers, including high-rated experts in discussion forums. Interestingly company policies mandating the use of such tools were highly effective; all 13 who had security tools mandated did use them.

An article by Bessey et al. [29] describes the experience of Coverity in building and marketing a source code checking tool. In particular they describe the motivation and issues with creating code checkers for large codebases. Their main interest is in the politics of tool purchasing; for example, the tool needs to deliver a true defect in its first three error messages to generate a sale.

There is a discrepancy between the large amount of work on analysis tools (Section 2.1.3) and the relatively small amount of research on getting those tools adopted; given the availability of such tools, this suggests that techniques to help developer adopt such tools will have large impact.

2.2.4 Consultancy and Training Interventions

Turning to the question of how to promote security enhancements, several research teams have explored the impact of training and external involvement on teams' delivery of secure software. Türpe et al. [172] explored the effect of a single penetration testing session and workshop on 37 members of a large geographically-dispersed project. The results were not encouraging; the main reason was that the workshop consultant highlighted problems without offering much in the way of solutions.

Poller et al.'s later study [134] followed an unsuccessful attempt to improve long term security practices in an agile development team of about 15 people. The study investigated the effect of security consultants whose task '*was not to advise the product group on how to change their organisational routines, but to challenge and teach them about security issues of their product'*. This proved insufficient, for two reasons. First, pressure to add functionality meant that attention was not given to security issues. Second, developers had trouble 'improving security' because their normal work procedures and ways of structuring their work did not support that kind of quality goal. The authors concluded that successful interventions would need "*to investigate the potential business value of security, thus making it a more tangible development goal*;" and that security is best promoted as a team, not individual, effort.

Others have investigated the effect of programmer learning on security improvement. Yskout et al. [205] tested if 'security patterns' (such as those described in Schumacher et al.'s book [148]) might be an effective intervention to improve secure development in teams of student software developers; the results suggested a benefit but were statistically inconclusive.

In terms of practical support for developers, a recent book 'Agile Application Security' by Bell et al. [28] provides guidance, a discussion of tools and detail on a range of assurance techniques; regrettably it is not selling widely[10].

The research findings do leave, as an alternative to traditional training, books or interventions based on penetration tests, the possibility of a team-based intervention.

2.2.5 Improving Security Experts' Interactions with Developers

Other work has investigated the impact of stakeholder relationships on software security: Werlinger et al.'s ethnographic study and survey [194] explored the relationships of security practitioners (mainly operations staff) on the effectiveness of security, and proposed several tool enhancements to improve this, particularly in the control of information being communicated to other stakeholders. Haney and Lutters found from a survey of security practitioners [81] that the role is service-oriented and requires both customer service and advocacy skills.

Ur Rahman and Williams [175] surveyed web-based information and nine teams of developers to investigate how DevOps—the integration of operations procedures into code—incorporates security into projects, finding increased collaboration between developers and security specialists, and security benefits in the automation of testing, configuration and deployment. Lopez et al. [109] identified from interviews with one professional team that security was introduced through a complex combination of processes, standards, practices and training.

Ashenden and Lawrence [12] took a proactive approach, using an Action Research method to investigate and improve the relationships between security professionals and business people in a single company, and found the approach effective in improving communication, though no evidence is yet available of longer-term impact. Their Action Research approach offers a suitable methodology for trialling other forms of intervention.

Werlinger et al [194] explored the relationships of security practitioners (mainly operations staff) on the effectiveness of security, and proposed several tool enhancements to improve this, particularly in the control of information being communicated to other

[10] Amazon rank 202,000, March 2020. #361 in Computer Security.

stakeholders. Weir et al. [183] used interviews with a dozen security specialists to elicit ways of improving developers' implementation of mobile app security, and concluded that the major opportunities were in encouraging 'dialectic': specific kinds of challenging communication with both the specialists themselves and a range of other stakeholders.

Another technique used to improve software security is code review by peers or specialists. OWASP have published a Secure Code Review Guide Book [129]. However, research on the effectiveness of code review has been limited to more general software improvement: Baum et al. reviewed a variety of earlier work and interviewed 24 professionals in 19 German companies. They conclude that cultural issues, rather than practical ones, determined whether code reviews were used, and that reviews were best embedded in the development process from the beginning of a project [25].

All this research concluded—or assumed and did not conclude otherwise—that the security impact of interactions between security specialists and developers is positive. There is a relative scarcity of security specialists, with a 2016 UK total of 58,000 [167] compared with 308,000 programmers (not to mention testers and managers) [159], so there is a definite need for cost-effective ways to leverage those interactions more effectively to empower developers to write secure code.

2.2.6 Motivating Change in Development Teams

To move from delivering insecure code to delivering secure code requires a change in thinking in the development teams. A variety of research has explored how to engender similar changes in software development teams.

Dybå [48] performed a wide-ranging quantitative survey of Software Process Improvement (SPI) in 120 organisations, and concluded that organisational factors were as least as important as technical ones. In particular, he identifies business orientation, the extent to which SPI goals and actions are aligned with explicit and implicit business goals and strategies, as one of the factors with the strongest influence on SPI success; together with employee participation, the extent to which employees use their knowledge and experience to decide, act, and take responsibility for SPI. Surprisingly, management commitment was not required. The paper also strongly recommends that, for SPI, measurement systems be designed by the software developers themselves.

Beecham et al. [27] conducted a literature review of 92 papers on programmer motivation in 2008. Though virtually all of the research cited is about motivation to do the job of programming rather than motivation to change behaviour, the survey identifies that professional programmers tend to be motivated most by problem solving, by working to benefit others and by technical challenges. Hall et al. [79] framed these as 'intrinsic motivators' (things done for their own sake), relating them to Self-Determination Theory, a theoretical framework that argues that a person's perceptions of their own autonomy, competence and relatedness (close personal relationships) provide the strongest motivation [67].

Fear of failure was not among Beecham et al.'s list of motivators, which suggests that merely frightening developers into security ("a terrible thing might happen") is unlikely to be an effective strategy to promote secure software. This is consistent with Xie et al.'s interviews of 15 professional programmers [200] to investigate why they believed they made security errors; they found a consistent tendency to treat security as 'someone else's problem'.

Umarji and Seaman [173] adapted two social psychology models to help analyse when developers might accept Process Improvement techniques. However, Riemenschneider et al. [141] used a ethnographic field study of 128 developers to compare five social psychology models of how developers adopt new 'methodologies', and concluded that none of the models matched actual practice, pointing instead to the need for an 'organisational mandate', compatibility with the developers' working practices, and acceptance by colleagues.

Singer [156] proposes 25 interventions to improve software development, suitable for a project manager or coach (such as '*Recommend similar and more proficient peers to less proficient developers*'), citing real-life examples for each; the approach was trialled successfully in a student experiment to encourage adoption of version control systems.

In terms of indirect influencing, an ideas paper by Lewis [105] suggests a score of 'gamification' approaches to encourage users to engage with software. Underlying such work is Fogg's book 'Persuasive Technology' [62], a seminal work on persuading people using computers. Fogg specifies three ways we see computers: as tools, as media (simulation), and as social actors (individuals in their own right); he describes some 40 'principles', aspects of software behaviour that can be persuasive.

Lopez et al., in the research project, 'Motivating Jenny' at the Open University [108], have investigated means to encourage developer teams in doing security. Based on ethnographic research, they concluded a need to *raise developers' security awareness* [109]. They explored formats for workshops to engage professional developers in discussions about security [110], ultimately releasing them as a series of workshop instructions and information sheets for developers to use themselves [111][11].

The work described in this section offers a range of possible approaches to influence developers to adopt secure practices.

2.2.7 Motivating Security Changes in Employees

The study of behaviour changes in people belongs to Social Psychology and Management Theory[12]. Fogg explored motivation for human behaviour, finding that individuals need to be motivated to perform the required behaviour; to have the ability to do it; and to be triggered to perform it [63]. Myers and Titgjen explored motivation in the form of job satisfaction, finding that positive motivations do not cancel out negative ones [169].

Research into corporate security behaviour has built on Fogg's theory; in this thesis we shall call motivating factors and triggers, 'Motivators'; and lack of ability, 'Blockers'. Thus, Motivators are events and factors that encourage a new behaviour, and Blockers are events and factors that discourage it.

Exploring how to promote effective security practices, Kirlappos et al. found from a large survey of employees in two organisations that a strong Blocker to the introduction of effective security lies in many organisations' attitude to security: information security was seen as a 'bolt on' compliance exercise at odds with the employees' main productive work [94]. Kirlappos et al. found in a later survey that this frequently results in 'shadow security', where the employees find their own ways to achieve security, different from

[11] Most of that content was written by the author of this thesis, based on work done by Lopez et al.

[12] The references in this section come from an introduction to Blockers and Motivators by the author and Ingolf Becker [185].

the organisation-enforced approach, with varying effectiveness [95]. Both surveys proposed a solution of empowering employees to create effective security rather than enforcing particular company-wide security strategies. Blythe et al. used two different behaviour change theories to explore employees' security practices, interviewing 20 employees in two organisations, explicitly identifying blockers; they conclude that feedback on the effectiveness of employees' security measures are likely to be more effective as Motivators than compliance approaches.

Pfleeger et al. analysed a range of social psychology literature to find its applicability to security behaviour, identifying Motivators such as a 'keystone habit' (a comprehensible goal and steps to achieve it), and 'moral foundations' (appealing to people's sense of rightness); and mentioning a few Blockers, such as the way existing security systems have conditioned people to ignore security warnings [132].

Moving to Developer-Centred Security, Assal and Chiasson explored security Blockers and Motivators in development teams [14], finding a frequent Blocker to be 'lack of management and process support for security', and an effective Motivator to be 'identifying the importance of security.'

The Blockers and Motivators concept offers a simple but powerful means to analyse the working of practical interventions.

2.3 Summary and Limitations of Existing Literature

In summary, the field of Developer Centred Security has followed four main topics of research:

- Adoption of static analysis tools (especially for Android);
- Literature providing guidance to developers;
- Means, such as Threat Analysis, to establish security requirements; and
- Research into developer security-related behaviour.

Little of this, even the last, relates to means to influence or change developer security behaviour. So, for interventions research we looked further afield, considering a range of topics and identifying outcomes from each as shown below:

Process improvements	A need for cost-effective ways to support developers in determining wider security needs
Analysis tools	Techniques to help developer effectively to adopt analysis tools will have large impact.
Consultancy and training interventions	The possibility of a discussion-based intervention emphasising the positive business value of security enhancements
Interactions with security specialists	A need for ways to improve communication between developer and security specialists
Motivating process improvement	A range of possible approaches to influence developers to adopt secure practices
Motivating general secure practices	Blockers and Motivators provide a powerful means to analyse the working of practical interventions

The review found relatively little research discussing successful security interventions to support development teams, even though the security track records of many large companies suggest that such interventions must exist. In the research discussed above, even code analysis tools required other interventions to get them adopted; and other approaches required both security professionals and interventions that were costly in terms of effort involved.

The research project, ‘Motivating Jenny’ at the Open University (Section 2.2.6) has created lightweight interventions, but there is no evidence as yet into their effectiveness.

We are aware of no other academic literature investigating lightweight approaches to encourage teams of developers to adopt successful security practices. Based on the previous discussions such an approach would need to:

- Be cost-effective, and not require security professionals;
- Support developers in determining wider security needs;
- Encourage developers to investigate security code analysis tools, especially just-in-time solutions; and
- Analyse and address the resulting Blockers and Motivators impacting on the teams.

This thesis offers one such approach.

3 Methods Used

This chapter explores the methodology of the research projects. It introduces the philosophical stance of the author, and introduces the methods for each separate project, specifically:

- Grounded Theory,
- Online Survey,
- Canonical Action Research and
- Design-Based Research

This chapter discusses only the methodological aspects of each project—aspects that would be relevant to other implementations; details specific to the specific implementation, such as numbers and details of recruits, are covered in the specific chapters describing each project.

3.1 Philosophical Approach

An important starting point is to define the philosophical approach we have as researchers. Goldkuhl [75] identifies three common philosophical stances depending on the aims and requirements of an Information Systems research project, as follows:

Positivist research looks for a single objective truth;

Relativist (or *interpretivist*) *research* rejects the idea that a single objective truth exists, and looks instead for a more local truth; and

Pragmatic research seeks specific social or business benefits as a result.

In this research, ultimately, we are looking to find ways to make a large number of teams more effective at software security. Our overall philosophical viewpoint, therefore, is Pragmatic.

3.2 Choice of Research Methods

The research question:

RQ 1 *What is needed to make a cost-effective and widely applicable intervention to help UK software development teams achieve better software security?*

identifies the research topic as one in Empirical Software Engineering (ESE). Empirical Software Engineering is the research area concerned with the study of how software creation happens in practice. The terms 'cost-effective', 'widely-applicable', 'intervention' and 'achieve' all mean that the question is about what happens in practice, and the emphasis on software development makes it an ESE question. We can identify five generally used research methods for this field [49]:

1. Controlled Experiments (including Quasi-Experiments);
2. Case Studies (both exploratory and confirmatory);
3. Survey Research;
4. Ethnographies;
5. Action Research.

Most have been used in Developer Centred Security research. For example, Acar et al. [3] used Controlled Experiments with individual software developers to evaluate learning about security, and Yskout et al. used A-B Controlled Experiments with students to evaluate the use of security patterns [205]; Ge et al. [69] wrote up a secure web project as a Case Study; Balebako & Cranor [19] used a Survey to evaluate app developers' attitudes to privacy; and Poller et al. [134] used an Ethnographic study to study the effect of an intervention by security professionals. Variants of Action Research have been widely used in Empirical Software Engineering [147], though as far as we know, it was not used not prior to this project in Developer Centred Security research.

Since we want a wide impact for the intervention, we need it to work with teams of professional software engineers. Though Controlled Experiments are possible on individual software professionals [3], we knew of no professional development teams who might cooperate with such experiments, and we rejected this approach.

Further, since the aim was to study new means to change the behaviour of software engineers, we rejected Case Studies as an approach.

Instead, therefore, we used Survey Research for the initial steps to provide knowledge and theory to support the creation of an intervention. This took two forms: a qualitative interview survey of professional security experts, and a quantitative online survey of software engineers.

For the qualitative survey, the choice of methodology was important, since taking recordings of unstructured responses to qualitative interviews and deriving academically sound knowledge from them is a difficult challenge. Two competing methodologies both address that problem:

- Grounded Theory [74] concentrates on deriving a single overarching 'theory'.
- Thematic Analysis [35] concentrates on deriving multiple themes.

Both use similar techniques in practice, though Grounded Theory's 'Everything is Data' principle tends to apply it to a wider range of data. The main distinction is in the aim of each methodology. Grounded Theory (GT) looks for a single 'theory', an overarching conceptual framework that can help a researcher or practitioner understand and perhaps change a complex situation [74]. Thematic Analysis looks for multiple 'themes', usually within the context of an existing ‘theory’ or conceptual framework [35]. Grounded Theory doesn’t ‘prove’ the theory it generates, nor even claim that two different sets of researchers given the same data will generate the same theory, but its methods allow GT practitioners convincingly to claim a measure of dependability in the process: that the correct use of GT is likely to generate a theory consistent with the data [33]. This research takes a pragmatic approach (Section 3.1): the resulting theory can later be accepted if it helps understand a complex situation, if deductions made using it are all confirmed, and if using it to plan research or interventions leads to desired outcomes; it can be rejected if not.

In the qualitative survey, the author was faced with large amounts of transcribed text representing a wide variety of ‘expert opinion’. To take action based on this expert opinion, a simplifying ‘theory’ was needed; none existed beforehand. This pointed to a need for Grounded Theory rather than Thematic Analysis. GT is widely accepted by the software research academic community [162], and was adopted on this basis. Section 3.3 describes the use of GT in detail.

For the online survey, by contrast, the choice of analysis method was straightforward; it used statistical methods, as described in Section 3.4.

The next research stage required the creation of an intervention as a proof of concept and exemplar to support learning more about the subject. The findings of the surveys provided outline requirements for the intervention, and ruled out, surprisingly, the conventional approaches to improving software security discussed in Sections 1.4 and 2.2.4. The experience of the author as a software consultant and team leader provided a range of facilitation and game techniques to use. The author therefore combined a game idea from the work of colleagues with ideation techniques to create the format of workshops for teams of developers: the Developer Security Essentials intervention. Chapter 6 provides more detail.

Trialling this intervention with professional developers required the direct involvement of the researchers as facilitators. This ruled out an Ethnographic approach, since Ethnography requires the researchers to remain relatively uninvolved. Instead we used Action Research [196], an accepted methodology used in many forms of academic social research including software engineering [47,147,153]. Specifically, we used the methods and approach prescribed for Canonical Action Research, using the entire project as a single ‘action research cycle’. Section 3.5 describes Canonical Action Research in detail.

Based on the findings from the first trials, the author modified the Developer Security Essentials package and embarked on a further set of trials with professional teams of developers. Constraints in the pure Canonical Action Research methodology led to the adoption of a different overall methodology, Design-Based Research (DBR). Section 3.6 describes Design-Based Research.

All four research projects were approved in advance by the Lancaster University Faculty of Science and Technology Research Ethics committee.

3.3 Grounded Theory

This section explores our use of Grounded Theory. Our Grounded Theory methods were identical to those used in the author’s previous project, and much of the content of this section is based on the author’s earlier Masters by Research thesis [183].

3.3.1 Introduction to Grounded Theory

Grounded Theory (GT) is a systematic methodology to construct theory through the analysis of data. It originally developed in the US medical field, where the direct value of discoveries about human social behaviour is high. Glaser's early works, 'The Discovery of Grounded Theory' and 'Theoretical Sensitivity' [73,74] contain a good deal of discussion of the social benefits of the process, and a polemical style against alternatives.

Within 10 years Glaser and Strauss/Corbin [163] were competing for 'ownership' of the technique and the two resulting 'flavours' of GT still remain distinct. Both approaches are *positivist* in essence, though Glaser's aim is to `discover' a single overarching theory, while Strauss is more interested in causes and effects [120].

Later still, as the technique was adopted by European researchers, Charmaz [33] tailored it to support the *relativist* philosophy [125]. The resulting variant is now known as Constructivist GT, an approach that emphasises the researcher's impact and the restricted applicability of any results. The approach used in this research is Constructivist GT.

Glaser and Strauss/Corbin's works are strong on justifications and in some cases team approaches, but less so on practical instructions how to go about making detailed choices in following the method [66]. However, software engineering researchers now have access to a range of work filling this gap. A good starting point is Hoda et al.'s set of patterns instructing a novice how to set about a GT research project [87]. Other work in this area extends basic GT with detailed advice, such as Adolph's 'Lessons Learned' [7] and Allan's 'Critique of Using Grounded Theory' [8]. Finally Stol et al.'s 'Grounded Theory in Software Engineering Research' [162] provides a detailed and very explicit set of instructions on how to achieve academic rigour.

3.3.2 A Brief Overview of Grounded Theory

Traditional science assumes that theories are generated as hypotheses by the researcher, which are then repeatedly tested against reality [133]. The concept is that random theories are winnowed by the scientific process to leave only those which match observable and testable fact. Grounded Theory attempts to make theory generation into a more dependable process, based on textual analysis. Rigorous testing of the theories generated is expected to happen via other approaches.

The textual analysis is of everything relevant that is available to the researcher. Thus, it might include interview transcripts, survey comments, relevant research literature, field notes from observation and anything else that can be reduced to text form. This is summed up in the GT principle *all is data*.

The process is iterative, with analysis of initial findings from interviews or similar typically leading to changes in the research thrust and direction, and with every code written being matched against all the others, a technique called the *constant comparative method*.

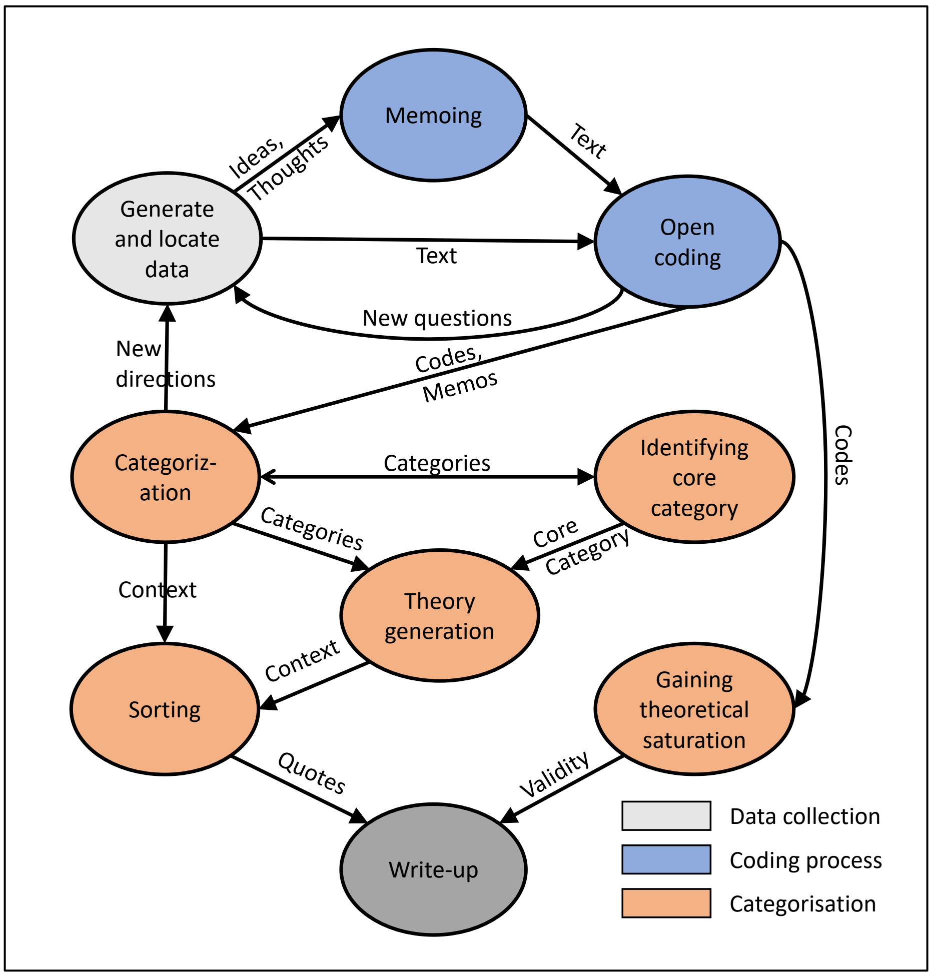

Figure 5: Using the Techniques of Grounded Theory

3.3.3 Grounded Theory Step-by-Step

Figure 5 shows the techniques used in the Grounded Theory process. As can be seen, the process is highly iterative, with the practitioners moving from technique to technique in response to discoveries, new data, and observations concerning the data.

Table 2 describes each technique in more detail.

Table 2: Steps in Grounded Theory

Technique	Description
Open coding	We scan each text line-by-line, highlighting points of interest. We then 'code' each to represent specific concepts, creating new codes as required.
Memoing	As we do that, naturally, ideas will occur to us and thoughts about how the terms may be interrelated. We write these in separate texts called memos. In doing this, we are open to novel ideas and concepts that may change and affect our future gathering of data. For example, if we see concepts emerging, we may explore these in more detail in future interviews.

Categorisation	Gradually, as we assign codes, we will naturally see them appear in groups of related concepts. We name these groups 'categories'.
Identifying core categories	In traditional Glassarian research, the aim is to find a single overarching Core Category: the one which covers the most interesting features in the data. Researchers are encouraged to look for categories that cover as much of the variation in the data as possible.
Gaining theoretical saturation	The data gathering is considered complete when further data received does not lead to significant new concepts. This is termed 'theoretical saturation'.
Theory generation	In building a theory, we are looking for relations between concepts and categories that explain the relations between concepts and categories.
Sorting	To provide a coherent output, we need a narrative. Strauss in particular talks a good deal about the literary aspects of the narrative, in particular 'grab', the relevance to the reader [4]. The sorting process is arranging the codes, memos and categories in such a way as to produce a convincing narrative.
Write-up	Here we write up the narrative as a coherent report. We use extensive anonymised quotations from the data sources to provide rigour, and use illustrations and where appropriate to convey the information clearly.

3.3.4 The Use of Literature Surveys

Hoda et al. [87] recommend using existing literature in a different way from other forms of research. Since the aim of Grounded Theory is to generate *new* theory, there is a natural concern not to be biased by existing thinking. So GT's original recommendation was to leave any literature survey until after the main bulk of coding and ideas generation [7]. Hoda et al. disagree with this, pointing out that literature surveys are often needed due to academic pressures and the need for the researcher to be up to speed with the subject terminology. They therefore recommend a short literature survey to begin with, and a longer one once the majority of data has been coded. The longer survey may itself contribute to the coding and memo generation. However, we have not seen a suggestion that academic papers should be coded with the same level of detail that is given to other research data.

Others suggest a similar approach: Allan [8] used a literature survey in advance of GT work to identify if there were compelling theories already in existence.

We have therefore taken the approach of an initial literature survey to learn nomenclature and avoid 'reinventing the wheel'. A final, post-research survey (Section 2.2) added further detail in the context of the discoveries from our interviews.

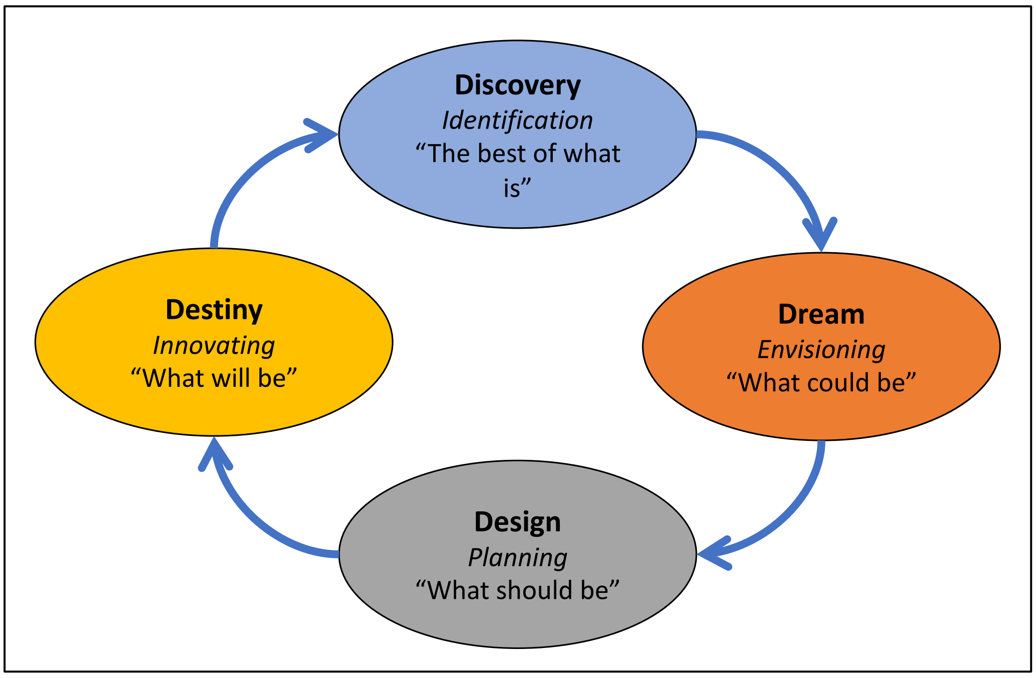

Figure 6: The Processes of Appreciative Inquiry

3.3.5 Incorporating Appreciative Inquiry

A further important question is how to structure the interviews to get the most helpful results. Our major concern in discussing security with experts was the danger of a litany of complaints and major problems they had seen. Much of the security literature concentrates on vulnerabilities, and we perceived a tendency among security experts to concentrate on the negative. We wanted instead to avoid the details of problems, and to focus on what had actually worked well for our interviewees.

This led us to look at Appreciative Inquiry. This method is primarily used as an Action Learning method, with its purpose to change the participants’ behaviour for the better as they reflect on their answers to questions. In this research we did not need to change the participants (the security experts). Instead we want to find the means to help *achieve the results being achieved by the interviewees, but in other contexts*, and so the Appreciative Inquiry method offers a valuable contribution.

The full Appreciative Inquiry method [41] involves a cycle of four processes as shown in Figure 6. The Discovery Phase typically concentrates on the positive aspects of the current situation, encouraging participants to visualise what has worked, and what is now working – hence the name ‘Appreciative’. The Dream Phase also stresses the positive, with participants working to establish a shared vision of the future. The Design and Destiny phases then continue to produce a plan for future change that has buy-in from the participants.

Table 3: Principles of Appreciative Inquiry

Principle	Summary
Constructionist	Our beliefs determine what we do; thought and action emerge from relationships.
Simultaneity	Enquiry changes systems
Poetic	Organisational life is made up from stories co-generated by the participants
Anticipatory	What we do today is guided by our image of the future.
Positive	Positive emotions are needed to generate sustainable change.

Underpinning this cyclical process is a set of five principles, which we may summarise simply as shown in Table 3.

So, when using Appreciative Inquiry, a community will work together (Constructionalist), starting with the Discovery of the current situation (Simultaneity), then generating a (Poetic) Dream of how they would like things to be. They then Design (Anticipatory) steps to take them there, creating a Destiny, which then becomes the baseline for the next cycle. To generate and sustain the energy required for such change, Appreciative Inquiry replaces the conventional trouble-shooter focus on problems, complaints and issues with a Positive focus, especially in the Dream activity.

Appreciative Inquiry has been effective in creating organisational change in a large variety of different organisations [40]. To use it as a research tool requires some changes that are explored in detail by Reed [140]. In our interviews of security specialists, however, we were neither working with a community nor did we have the opportunity to work with the specialists over the timescale involved in a cycle; so we used only the two elements that could work effectively in a single interview: each interview concentrated on AI's Discovery: "What do you do and have you done in the past that was successful" and Dream: "What would you like to see in an ideal world?" In terms of principles, the interviews strongly adopted Appreciative Inquiry's Positive principle; and gathered Poetic stories to characterise the effective techniques discussed by the participants.

3.4 Online Survey

This section discusses the methodology behind the creation of an online survey. Such surveys are valuable to find objective, quantitative, data about the current situation, and indeed in this research we used a survey to find background information about the secure development habits of a community of professional developers.

Chapter 5 covers the practical elements of the specific survey we did, such as choice of participants, scope of questions, and the additional artefact analysis carried out.

The important methodological aspects can be summarised as:

- Ethical issues
- Questionnaire design
- Questionnaire implementation
- Analysis design
- Review and piloting
- Sample sizing and marketing
- Survey follow-up

The following sections discuss each in turn, describing each in terms of the activities we carried out and the reasons involved.

3.4.1 Ethical Issues

Probably the biggest ethical issue is the privacy of the participants, since the responses contain personally identifiable information. Addressing this requires:

- Keeping all responses secure. Universities provide guidelines for securing personally-identifiable data [101].
- Ensuring that publications do not permit the identification of any individuals. This applies particularly to the publication of the research data set, and includes ensuring that no combination of data in a response might permit the identification of a particular respondent.

A second ethical issue is when it is appropriate to 'spam' the members of a mailing list. This is a complex issue, governed partly by legal constraints such as GDPR.

Another concern concerns the use of rewards to the participants. Some surveys offer a lottery, offering benefits such as cash payments. There are practical issues with implementing this, and possible selection biases it might instigate, such as encouraging less well-off participants. Where, as in this survey, the rewards are altruistic benefits from the research and a report sent to participants who request it, the ethical requirement is to ensure that the analysis is properly carried out and that the report is indeed sent. In particular, we interpret the commitment to provide research benefits as a requirement to make the data set available for future researchers.

A particular concern is the effective use of participants' time. By sending out large numbers of survey requests we have an ethical responsibility to ensure that the time that participants spend is not wasted, by ensuring that the questionnaire is as effective as possible. This requires effective review and piloting (see section 3.4.5).

Finally, there is the issue of the accountability of the researchers involved, providing publicly available details of the study purpose and contact details for the researchers and their superiors with the survey invitations.

3.4.2 Questionnaire Design

The first decisions were whom to survey, and what to learn about them. Whom to survey is partly a pragmatic decision, depending on what datasets of contacts are available; an online survey requires a large list of the email addresses of potential contacts. The questions to ask derive from the research questions.

Before trialling questions on target participants or offering it for external review, it is important to challenge it based on several criteria:

Survey completion time: Longer surveys tend to get a high dropout rate. Naturally the acceptable time depends on the nature of the participants, but expert sources cite maximum recommended times between eight minutes (SurveyMonkey) and 10 minutes (Qualtrics). A practical rule of thumb allows us to calculate objectively how long participants will take for a given set of questions: allow a point for each simple question, a point for each row in a grid question, a point for each two responses in a multiple select question, and two points for each mental calculation required; then multiply by 7.5 seconds per point [178].

Practical implementation: The practical limitations of creating questions on a web-based survey platform may require changing the phrasing or the nature of questions.

Statistical relevance: it is important to take account of the analysis planned on the Survey results, to phrase and to target the questions in such a way as to get an effective result (see section 3.4.4).

3.4.3 Questionnaire Implementation

It is essential to use an online platform, and the better platforms improve both implementation and results. In practice we used Qualtrics, one of the market leading platforms. Several points are important in implementing the survey on such a platform:

Linking results with invitations: where there is metadata about the invitees: for example, their location and product name, this much be matched to the corresponding survey results.

Ease of statistical analysis: the survey platform codes most multiple-choice answers as integer numbers. For analysis it is helpful if these numbers correspond to useful values for statistical input. For example, Likert-type scales such as 'Not knowledgeable at all …. Very knowledgeable' might use increasing integer values, such as 1 to 5.

Multiple platforms: participants are likely to be responding on anything from a PC to a mobile phone. Grid-style questions, where a set of questions each has the same range of answers, are more difficult to answer on a mobile device, and are worth avoiding.

Encouragement: We want each participant to finish the survey if possible. So, it is important to keep encouraging them to carry on. A good solution we used in this project is to add a progress bar to show users how close they were to completion.

Irrelevant questions: some questions are only meaningful in certain circumstances. For example, if the participant answered "No" to the question "Are you in a team?" then it would be meaningless to ask the team size. The survey software handles these using constructs called 'conditions' and 'blocks'; it is important to implement these carefully and test them in the initial face-to-face trials.

Randomisation: the order of the questions and of the answers within questions is important. To prevent this having a statistical effect, it is worth using randomisation where possible. Randomising the order of all of the questions would create problems around the 'irrelevant questions' discussed above. Instead one can randomise the order of questions within grids, and the ordering of answers where these have no logical built-in ordering.

Standard answers: Where possible, one can build on the experience of other researchers by reusing tried-and-tested answer sets provided by the platform. For example, for knowledgeability Qualtrics provides a Likert-type scale from 'Very knowledgeable' to 'Not knowledgeable at all'.

Optional answering: some questionnaire systems force the participant to answer every question, by not allowing progress until an answer is present. This can be irritating and deter participants from completing the rest of the survey. However, it is important to avoid participants accidentally missing out questions; a compromise is to use a dialogue "… Are you sure?".

Coding answers: in some cases, we did not know what set of answers to expect. For example, the question "What development environment do you use?" might have some obvious answers but might also have a range of possible answers we were not expecting. So, we created the survey with an initial fixed set of multiple-choice options, and an additional "Other" option with a corresponding free text field. Following the pilot study, we manually 'coded' the answers for that text field, then incorporated the most common 'codes' as further multiple-choice options. We retained the 'Other' option in these cases, but in practice none of the text answers proved of value in reporting the survey results.

Low-value responses: Questionnaires, especially in psychology, sometimes include 'attention questions', where the same question is asked in two different ways: typically negated the second time. Researchers can then either reject responses with inconsistent answers, or do statistical analysis [166] determine whether responses are sufficiently internally consistent. However, in questionnaires about facts rather than opinions this approach can appear contrived and irritating to the participants. An alternative to reject low-value responses is to analyse the time taken to complete the survey, rejecting those completed so fast that it was unlikely that the respondents were responding accurately.

3.4.4 Analysis Design

In parallel with the survey there needs to be an analysis plan. This fed back to the question design and implementation.

Our initial inclination was to gather the survey results first and then decide on statistical analysis according to what we found. This approach works fine for purely descriptive statistical analysis, such as graphical representations of the results of questions and simple calculations from those numbers. However, for statistical hypothesis testing—specifically any statistical analysis that returns a p-value—this approach is unsatisfactory.

The reason lies in the problem of 'data dredging'. When we run a test of statistical significance we are always implicitly or explicitly comparing two hypotheses; one that the data tested shows a particular feature, and the other, 'null hypothesis', that it doesn't. A 'p-value' is the probability that the data analysed could have shown that feature even if the null hypothesis was true, just by random chance. The arbitrarily accepted p-value threshold for statistical significance, 0.05, is likely to be achieved by random chance every now and again, even if the feature is not present. In fact, if we do as few as 14 analyses on different samples of data that doesn't have some feature, we still have a more than 50% chance that one will show that feature with a 'significant' p-value of less than 0.05[13].

This means that if we take a dataset and run enough statistical tests on it, we will find 'statistically significant' results purely by chance. Doing this is called 'data dredging' and is poor research practice; in particular it leads to conclusions that are unlikely to be replicable [89].

The main ways to avoid this are twofold:

First, we define, before data collection, what statistics analysis we are going to do, what the tests will be, and what are the hypotheses and null hypotheses. Indeed, many journals now accept papers that define the research process and analysis prior to the data collection, with a guarantee of publication regardless of the subsequent results, because this leads to more replicable, and therefore higher value, research results. Research best practice guides also require this approach [38,39].

[13] $\Pr(no\ significant\ results\ in\ 14\ analyses) = (1 - 0.05)^{14} = 0.49$

Second, when the planned and statistics do require several tests on the same data, we use corrections and techniques to ensure that the criteria for statistical significance are modified accordingly. For example, the Bonferroni correction divides the threshold for statistical significance by the number of tests made [144].

In deciding statistical techniques, we need to both identify the techniques required, and find ways to ensure that correct preconditions hold for each. For example, many statistical calculations assume some form of Normal distribution in the data; to use one, for those we would need to ensure that the dataset satisfies this precondition.

In the statistical analysis for this project we had two forms of question:

1. In the survey results, a proportion satisfied some criteria (e.g., 'works with security professionals'); what can we deduce about the wider population?
2. We have two sets of 'factors', measured numeric values (e.g. rating of importance of security, frequency of use of security techniques); do they show a mathematical relationship?

For 1, ignoring concerns about sample validity, we can express the answer in terms of the 95% confidence limits on the proportion in the wider population. This is a simple formula, calculated from the sample proportion and the sample size [99], and requires only that the sample has the same variance as the wider population—i.e. that the sample is representative.

For 2, the usual relationship to look for is a linear one. If there is no need to use the result to make predictions, linear regression is unnecessary; instead we used the Pearson Correlation Coefficient ('Pearson R') calculation [44] to establish whether pairs of values had a linear relationship. Note that in representing the relationship graphically it helps to show a line on the graph; to calculate this line the graph plotting software used Simple Linear Regression [44].

The preconditions for the Pearson Correlation Coefficient test are that the data tested are continuous (i.e. numeric, with equal intervals between adjacent units); come in matched pairs (x,y values); have no outliers (values far from the majority); and have linearity and homoscedasticity (criteria that amount to 'look vaguely linear on a scatterplot'). We tested these by plotting a scatterplot or similar to show the data.

Following normal statistical practice, we coded Likert-style responses [106] on a range of 1 through 5 to create effectively continuous data [91]. It is accepted practice to use Pearson R tests on such coded scores [96,124].

Some of the data (such as the number of errors found by analysis in an application) forms a Poisson distribution[14]. To permit linear analysis on this data we use a transformation, $\log(x_i + \mathrm{k})$, where k is chosen to minimise skewness [30]; in practice we trialled different values of k, finding no difference to the results, so used the conventional research practice of k=1.

One form of analysis is to look for correlation between a matrix of 'factors', where each factor is represented as a numeric score. For example, we might have had A, B, and C as input ('independent') factors—things that we might think of as causes—and X, Y, Z as output ('dependent') factors—things we might think of as effects. In that case we would

[14] The Poisson distribution may be thought of as the 'bus waiting times' distribution; if buses arrive randomly, then the distribution of waiting times obeys the Poisson distribution: Pr $(waiting\ time\ > t) \propto e^{-kt}$

have tested X’s correlation with each of A, B, and C; Y’s correlation with each of A, B and C; and Z’s correlation with each of A, B, C. Since this constituted several tests on the same data, we applied the Bonferroni correction [144], reducing the threshold for ‘significance’ accordingly.

To investigate the sample validity, we used one further statistical technique. We investigated whether results from a subset of the data (e.g. developers of apps with less than 1000 downloads) differ materially from the remainder. Since the two samples—subset and remainder—are independent and we have no knowledge of the distribution of the values, we used the Mann Whitney rank sum test [144] to compare the sample medians. The other preconditions for this test—that the two populations have the same distribution and variance—are reasonable for this situation. We used the same test to compare the survey results with those from the corresponding larger populations.

3.4.5 Review and Piloting

There are a variety of approaches to help ensure the survey is effective [64].

Expert Reviewing: This is a method that supports identifying questions that require clarification and uncovering problems with question ordering or potential biases [136]. It involves asking an experienced usable security and privacy researcher with survey expertise, who is not part of the research team, to review our survey questionnaire and evaluate question wording, ordering, and bias.

Face-to-face Testing: Next, is face-to-face pilot surveys with a few candidate participants who were not previously involved in the research project. This supports revising the survey questions based on realistic feedback.

Pilot Survey: Before conducting the full survey, one can send invitations piecemeal until a few dozen replies are received: enough for simple analysis and feedback but minimising the effort ‘wasted’ by participants whose responses would not appear in the final survey. The goal is to test the survey questionnaire under realistic conditions and with participants drawn from the same pool as we used for the full survey. The pilot results are not incorporated into the main survey.

The pilot gives an indication how many responses to expect from a given number of invitations. It can also check that the number of dropouts during the survey was acceptable. A further benefit is the opportunity to manually code the responses of some of the text-based questions, taking the most frequent codes to create new coding answers (see section 3.4.3).

3.4.6 Sample Size and Marketing

There are several ways of identifying a sample size [64]. Some researchers use analysis of the data collected to decide on a cut-off point (such as when the Cronbach Alpha between a pair of attention questions is sufficiently high). This is justifiable but has a tension with the principle of defining the statistical analysis in advance.

This project used Fowler’s guidance [64], We identified the smallest subgroups for which we wanted data, and used the pilot data to estimate the proportion of these in the population. We then calculated a sample size large enough to get significant data from these groups, based on the calculation for the Confidence Interval for a Population Proportion [44].

To maximise the effectiveness of the survey, and to avoid wasting the limited resource of invitee details, we wanted a good response rate. In particular we wanted to encourage participation by making it completely clear that this was a bone fide university research project. There were several aspects to achieving this, as follows.

Email distribution: Mass distribution of emails is fraught with problems. Most email services have complex arrangements to detect incoming spam and prevent it from reaching their uses. And universities discourage individual users from sending mass emails. To address this Qualtrics provides a mass mailer system that can be set up to send emails legitimately associated with a university address; other systems may have other solutions.

Invitation wording: The invitation must make it clear that the senders are legitimate researchers, acting with the approval of, and subject to the constraints of, their universities. It is also useful to personalise the emails as much as possible.

Project description: ethics standards require a full explanation, with contact details and constraints on the researchers, to every participant in advance. Best, to make it look as legitimate as possible, is to host it as a webpage on the university domain.

3.4.7 Survey Follow-up

To publish the data for future researchers requires a version of the results that preserves the privacy of the responders. This requires deleting any personally identifiable data including:

- Names, emails and application names and identifiers, and
- Locations (Qualtrics provides precise locations).

3.5 Canonical Action Research

This section introduces Action Research generally, and the Canonical Action Research method in particular.

3.5.1 Action Research Methods

Action Research is an approach to research in communities that emphasises participation and action; Action Research aims at understanding a situation in a practical context and aims at improving it by changing the situation. It has its roots in research in the 1940s by Kurt Lewin, a relativist sociologist, whose work on community social issues led him to conclude that "*mere diagnosis… does not suffice*," and therefore that the role of the researcher was to add "*experimental comparative studies of the effectiveness of various techniques of change*". This 'action-research', or "*research to help the practitioner*", needs therefore to involve the practitioners, and to occur in cycles of planning, execution and evaluation [104].

This approach has clear advantages in software engineering, where development team leaders tend to see 'time not spent on development' as an expensive luxury, and therefore many will not engage with researchers unless there are probable benefits from doing so. Action Research offers such benefits and is now widely used in mainstream research. One

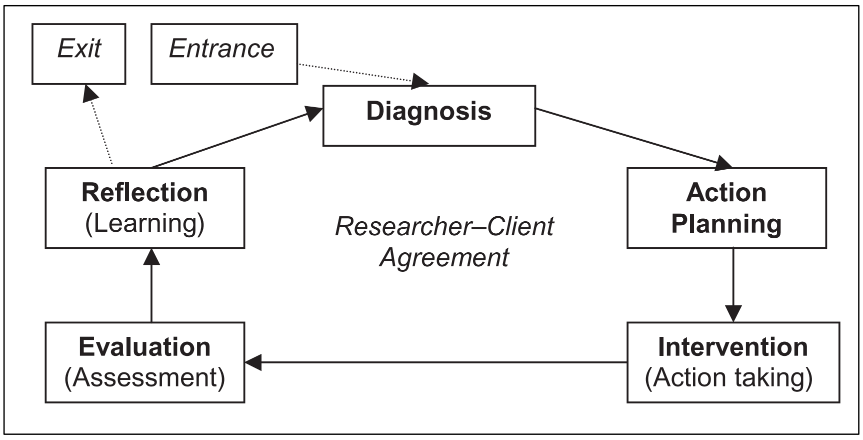

Figure 7: Canonical Action Research Activities (Davison et al. 2014 [43])

taxonomy identifies four main approaches [102], all used in Empirical Software Engineering research [147], and each with many variants:

Action Learning	Uses group reflection techniques (Action Learning Sets), which may be facilitated, to help teams address and learn from complex problems.
Action Science	Addresses organisational change; and particularly concentrates on participants' own thinking and habitual behaviour as the source of the most intractable and challenging problems.
(Canonical) Action Research	Addresses a specific problem with researchers working with teams in an iterative fashion: the researcher is responsible for theory; the participants for action.
Participatory Action Research	Has its basis in working in communities; it extends Canonical Action Research by emphasising the equal involvement of participants, including in the generation of theory.

Other taxonomies identify many more variants [23]. Note that the name 'Action Research' refers both to the generic form of research and to a specific variant; in this thesis we use the term 'Canonical Action Research' for the variant to avoid confusion.

3.5.2 Canonical Action Research

Canonical Action Research follows a cyclical model, as shown in Figure 7. It is characterised by five principles: researcher-client agreement, cyclical process model, theory use, change through action and learning through reflection [43]. These are summarised below.

Researcher-client agreement: The researcher and client must agree that CAR suits the situation and that the research goals are appropriate.

Cyclical process model: The work follows one or more cycles of diagnosis, planning, intervention, evaluation and reflection, as shown in Figure 7. Each cycle builds on the previous. The remaining principles refer to the stages in this cycle.

Theory use: The work may start based on a specific theory; if not, the researchers will derive one through the reflection process, typically using an approach akin to Grounded Theory (Section 3.3). Focussing on a theory ensures that the research remains relevant to the research community and supports sense-making from the large amount of data collected.

Change through action: “The essence of CAR is to take actions in order to change the current situation and its unsatisfactory conditions”. Both researcher and ‘client’ are motivated to improve the situation, with an intervention appropriate to the diagnosed problem, based on a clear understanding of the problem and its context and causes. A thorough analysis of the outcomes then informs both general knowledge and future interventions.

Learning through reflection: The researcher reports back to both clients and to the research community. This involves both keeping the client informed of progress; and reporting via publications to the research community, concentrating on what has been learned, especially where related to the theory.

3.5.3 Practical Canonical Action Research

The practical aspects of carrying out CAR are less well documented than the theory. The following is the approach we adopted.

First, we interviewed a selection of the future participants to establish a baseline in terms of their current understanding, practice and plans. Each interviewee completed and signed a consent form, and a representative of each team signed a different one for the organisation, before either interviews or workshops.

We then carried out a series of workshops with members of the development teams, led by the intervener. Then, a suitable time after the final workshop, we carried out ‘Exit Interviews’ with the same participants as before. Both Entry and Exit Interviews were semi-structured using open questions. Later, we attempted ‘One Year Interviews’ with the leaders of each team, to find out to what extent the security effects of the package were long-lasting, using the same questions as the earlier exit interviews.

We recorded the audio of all the interviews and all the workshops. Given the importance of the recorded interviews, we used two recorders of different makes for all recordings, to reduce the risks from error, battery failure or recorder failure. Ethical considerations and university guidelines [101] required us to keep all such data only on encrypted storage; since encrypting recorders are both expensive and cumbersome to use we used standard recorders, moving the recordings onto an encrypted laptop and erasing the recorders’ memory immediately after each session.

The recordings of the interviews and workshops were transcribed and qualitatively analysed. In an iterative process, the author and a colleague coded all transcripts. Initially both coders used open coding [74] on the first two hours of material, then agreed on a coding scheme based on that and the research question. Then both coders independently coded all the remaining material to this coding scheme and compared the results. Differences in coding were discussed and resolved between us, which often led to the creation of new codes or redefinition of existing ones. Following the one-year interviews, we both again independently coded the transcribed interviews and compared the results.

We used the industry-leading tool NVivo [137] for coding. The teamwork and merging facilities of the Windows version[15] make remote working straightforward, and its analytical functions [71] allow sophisticated analysis of both text and coding.

[15] The Mac version of NVivo lacks many of the features and uses a different file format; the author ran the Windows version on a Mac using the Parallels VM software.

To analyse change caused by the process improvement, we coded the initial interviews to provide evidence of a baseline before the start of the process.

Using a dedicated text analysis tool such as NVivo permits sophisticated analysis in terms of 'queries' on the coding [71]. To support this, we used automated features of the software to 'auto-code' each interview transcript to distinguish speakers: using anonymous codes to identify the interviewer, and each interviewee, and to assign interviewees to company codes; and we categorised all transcripts as 'before', 'during' or 'after' the workshops. This allows queries such as "Mentions of Feature X identified before the intervention, categorised by company." Using such queries in conjunction with spreadsheets to analyse numeric data further allowed us to create several forms of visual representation of the data (see).

3.6 Design-Based Research

The projects were focussed on the intervention more than on the specific improvements achieved in each of the organisations involved. A methodology designed for this situation is Design-Based Research. The motivation for using it in the Magid 2 project is discussed in Section 8.1. This section explores its salient features.

3.6.1 Introduction to Design-Based Research

Design-Based Research (DBR) has its roots, and is used most, in education research. Its foundation lies in the 'Design Experiments' of Brown [31], and Collins [36] prior to 1992: a specific method that worked with teachers as co-experimenters and emphasised the development of design theory in parallel with the creation of teaching innovations. Though it does not derive directly from Action Research, DBR shares the principle that the researcher may themself be part of the research project [22].

By 2003 the approach was widely adopted in education research, and researchers from ten different institutions agreed to term the approach 'Design-Based Research', to avoid confusion with other forms of research [86]. Initially used predominantly to support the design of 'Technology Enhanced Learning Environments', DBR is now an accepted research paradigm, used to develop improvements ranging from tools to curricula [86]. with online tutorials [174] and a recent comprehensive guide book for practitioners [18].

The characteristics of Design-Based Research are that it is *pragmatic, grounded, interactive, iterative and flexible, integrative and contextual* [181]. The next sections explore these attributes in turn.

Pragmatic: DBR aims to solve current real-world problems, by creating and trialling interventions in parallel with the creation of theory and design principles. Promoters contrasted it to the 'Controlled Experiments' used previously in education research, in that the purpose was not primarily to test theory, but to create interventions or improvements in (educational) practice.

Grounded: The research is grounded in both in the creation and development of theory; and in the practicalities of real-world trials in the '*buzzing, blooming confusion of real-life settings*' [22]

Interactive, iterative and flexible: To create real-world changes requires interaction between researchers and practitioners, since typically it is only through practitioners that change can take place. It also requires an iterative process, usually over a long time, with

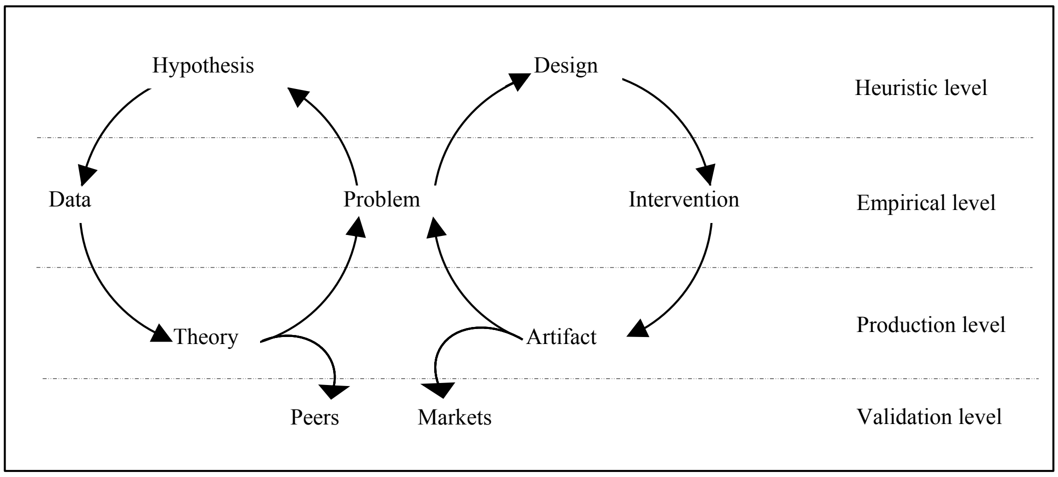

Figure 8: Design-Based Research Activities (Ejersbo et al. 2008 [53])

multiple trials and experiments taking place as the theory develops. This iteration permits flexibility, allowing changes in process as well as theory between iterations.

Integrative: DBR practitioners may integrate multiple methods, and vary these over time, to support the *credibility of findings* [181].

Contextual: The results depend both on the design process and intervention created, and on the context of the real-world trials.

There are several types of theory that DBR research may generate [50]:

Domain Theories:	General theories about the situation and interactions between participants.
Design Frameworks:	Prescriptive guidelines to design a particular type of intervention or product.
Design Methodologies:	Prescriptive guidelines for an approach to design a range of interventions or products.

The classic illustration of the processes that make up DBR is shown in Ejersbo et al.'s Figure 8 [53]. It shows the two cycles of research: creating theory and creating the artefact. The researchers are involved in both, generating academic output from the theory cycle and practical impact from the artefact cycle. The creators of that illustration, however, are careful to stress that this virtually never happens, and that in practice particular projects use variations, swapping emphasis across parts of each cycle.

3.6.2 Practical Design-Based Research

Elegant though that illustration is, it does not capture the interaction between the two cycles, which seems to be limited to a shared Problem; nor is it clear how a Hypothesis might generate Data. We offer instead Figure 9, which shows the source of the Data, and also re-labels the Intervention-Artifact sequence as the creation of an Artefact followed by a Trial. In this model the Hypothesis also feeds into the Design process, and the Trial generates the Data that supports or denies the Hypothesis (shown with a dotted arrow), generating Theory. The result is that the 'Design Theory' cycle does not exist as a separate entity, but instead depends crucially on the practical Trial. Although this representation is less elegant, we suggest that this offers a more practical view of Design-Based Research, at least in the context of this research.

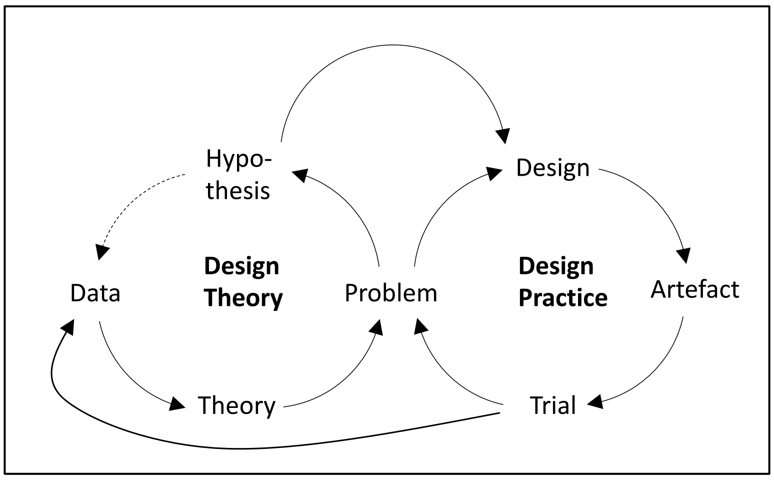

Figure 9: Activities in Practical Design-Based Research

Curiously, there is little discussion in the literature about the practical aspects of carrying out Design-Based Research. Even Bakker's 'Practical Guide' book [18] describes how to frame a question, keep the project manageable, write up DBR papers, and the philosophy of DBR; but does not describe how to carry out the major part of the work: creating and assessing a product or intervention.

We conclude that the reason for this is captured in the 'Integrative' nature of DBR: both design and assessment techniques must come from other research methodologies. Such research methodologies may be based on Action Research, Ethnography, and even on occasion Surveys, Case Studies and Controlled Experiments [49]. Thus, the techniques of the Canonical Action Research method—though not that method's overriding paradigm—provide an acceptable basis for data gathering.

3.6.3 Practical Implementation

Since DBR concentrates on the intervention, it works well with a large number of trials in different situations. To have many trials is highly desirable but requires a change to the CAR techniques described in the previous section; this section describes how we made that change.

The problem is that manual transcription of all the sessions becomes prohibitively expensive. As a solution, only the entry and exit interviews—the most valuable and data-rich content—were transcribed manually; we coded all the workshops and training sessions from audio.

Coding from audio alone is difficult; and also makes it hard to make sense of summaries, such as a summary of all the statements assigned to a particular code. However, shortly before the start of the project Google had made their speech-to-text services available as paid APIs. Several online services have appeared using these services to provide automated transcription at small cost; after some experimentation, we concluded that, while far from perfect, the transcription quality was sufficient to support coding in NVivo: it makes it clear at a glance what the speakers are discussing, even though the actual transcriptions can be nonsensical or hilarious. Even automated transcription of group discussions, though not recommended by the service suppliers, was helpful. The automated transcriptions made the audio faster to code, and easier to analyse, than using NVivo's 'audio-only' coding feature.

We used the Sonix automated transcription service and created a simple script to change the format of the transcripts to that supported by NVivo. In view of the privacy concerns

Table 4: Example NVivo Report Showing Aspects and Levels

	Level 1	**Level 2**	**Level 3**	**Level 4**
1 : Aspect A	0	0	0	0
2 : Aspect B	3	3	3	3
3 : Aspect C	3	3	10	5
4 : *… etc.*	0	0	0	0

of using such a service, we discussed it in advance with the leaders of each of the teams (all agreed to its use) and amended the participant and organisation consent forms accordingly.

3.6.4 DBR Data Analysis for an Intervention

Design-Based Research includes the evaluation of an artefact. When the artefact is influencing behaviour, we need a way to establish what changes have occurred. Here is the approach we used.

The author and fellow coder agreed a coding scheme in advance. Specifically, we decided to code the extent to which several different aspects of the participants' behaviour had changed. We therefore identified a list of aspects, and a grading scheme to identify what level the participants had achieved for that aspect.

We wanted to compare levels for each aspect 'Before' and 'After' the use of the artefact. Rather than have many separate codes (such as 'Level 2 for Aspect A', 'Level 4 for Aspect C', etc.), we used 'code pairs': coding each mention twice, once for the aspect and once for level. We used 'auto-coding' to assign codes Before and After to the different participants and companies. That allowed us to use NVivo queries similar to the following:

> *"Text coded to 'Before' and 'Company E', cross-tabulated by Aspect and Level"*

Which produces a 'cross-tabulated' matrix[16]. Table 4 shows an example for a specific company, from the interviews Before the intervention. It shows the Aspects in rows, and the Levels as columns; the value in each cell is the number of code pairs that showed that Level of for that Aspect.

From that table, it is then simple to calculate the highest Level coded for each Aspect, representing it as an integer from 0 (not coded) through to 4 (Established). So, for example, in Table 4 for that particular company and time, we see that Aspect A scored 0; Aspect B scored 1; and Aspect C scored 4.

We repeated this for every company for both Before and After the intervention, and copied the resulting tables into an Excel spreadsheet.

On occasion a statement made by an interviewee in an *exit* interview or a workshop might provide information about plans or engagement *prior* to the interventions. We handled this by adding the code Before to the corresponding statements (thus the statements are coded both Before and After). So, the NVivo queries above will include the appropriate

[16] In practice, NVivo queries are less sophisticated. We achieve the search above by starting with a query like 'Coded to Before and Company E', saving the result as a 'result set', then using that 'result set' as the scope for the cross-tabulated query. NVivo also lacks any automation, so repeating this for different parameters is a long manual process.

Table 5: Interpretation of Cohen's Kappa (Viera and Garett [179])

Kappa	Agreement
< 0	Less than chance agreement
0.01–0.20	Slight agreement
0.21– 0.40	Fair agreement
0.41–0.60	Moderate agreement
0.61–0.80	Substantial agreement
0.81–0.99	Almost perfect agreement

text about engagement before the interventions, even when the corresponding statements were made afterwards.

As is normal practice, both coders coded independently, then met up to discuss, and each then independently modified their coding according to the discussion.

3.6.5 Validation of Analysis

For this project, we wanted to be able to cite a statistical rating of the accuracy of our coding analysis.

The accepted approach to this is to use 'Inter-Rater Reliability' (IRR) [78]. This calculates a number between 0 and 1 that can be interpreted as shown in Table 5 by Viera and Garett [179]

The main decision in using IRR is the selection of what ratings to compare: possible ratings include the extent to which each rater assigned the same codes to each *character* in the text (as NVivo does in its default IRR analysis); or the extent to which each rater assigned the same codes to each *paragraph* in the text; or the extent to which the *final results* derived from each rater's codings agree with each other.

Since we were using coding to establish the extent to which the participants had changed in each aspect as discussed in the previous section, the relevant ratings to compare with IRR analysis were these *final results* figures as calculated for each coder.

Another consideration is how to handle aspects that are 'No mention' for both raters, representing aspects that had not been discussed in the interviews, such as Aspect A in Table 4. Whilst this was certainly 'agreement' between the raters, arguably it was not agreement that was interesting in the context of Inter-Rater Reliability. Accordingly, we calculated a version of the IRR calculations based only on cases where at least one rater had assigned a value ('Active rating subset' [78]).

There are several possible calculations: Cohen's Kappa, Krippendorff's Alpha and others for different numbers of coders and situations [78]. We did the calculations for both Cohen's Kappa and Krippendorff's Alpha, finding the results reassuringly identical to two significant figures, so in this report we quote the numbers as Krippendorff's Alpha.

NVivo supports queries specific to each separate coder, and so we could create queries similar to:

> *"Text coded to 'Before' and 'Company E',*
> *cross-tabulated by Aspect* ***as coded by Coder 1,***
> *and Level* ***as coded by Coder 1****"*

As before, we exported the resulting tables into an Excel spreadsheet: we calculated the maximum Level for each Aspect, for Before and After, for each company, repeated for each coder. We then compared those figures for each coder using the IRR calculations[17].

To give a meaningful indication of the IRR values, we did calculations of Krippendorff's Alpha for both the full set of ratings, and the 'Active rating subset'; and both after the first, independent, coding activity, then after the discussion and modification.

3.7 Visualisation of Results

Finally, an important aspect of the research method is the presentation of the findings. Though calculated figures are sufficient as a basis for evidence claims, visualisations can be much more effective at conveying the meaning of figures. Both Excel, and Python in Jupyter Notebooks, are widely used to create visualisations, and both are used to create figures in this thesis.

Excel proved the most flexible tool, since it supports 'manual' changes in the manner of a drawing package. There are a variety of possible Excel techniques, as follows:

Standard Excel Charts: For these one creates cell ranges containing the data to plot in the number or text format required and uses the 'create chart' functionality; then uses mouse and menu commands to edit the colours, sizes, axes, legends and other features.

Advanced Excel Charts: Excel supports more sophisticated charts, allowing one to have multiple series types (bars and lines, for example) and to manipulate these; or to change the representation of individual points in a series.

Compositions of Charts Figures and Shapes: By removing the axis, title and key displays from, for example, a ring pie chart or single stacked bar chart, one achieves a structured shape that one can then position on a two-dimensional 'composite diagram' to convey several dimensions of information. Excel also supports the drawing shapes, text, lines and arrows familiar to users of PowerPoint, and one can position those as axes and other shapes on the composite diagram. One can also use, for example, white shapes on a white background to blank out unwanted aspects of any of the charts. Figure 10 shows a detail from a visualisation created in this way.

Excel Visual Basic for Applications: Excel has VBA scripting, and supports functions that can take data from ranges of spreadsheet cells. Each shape and chart on an Excel diagram has a text name. These names have defaults (such as 'Chart 2') but can be edited to be meaningful ('Company I'), and the corresponding 'objects' can be manipulated from VBA. One can therefore program a VBA function to set the sizes, positions, colours or other attributes of shapes on a diagram according to spreadsheet data; then use additional mouse and menu manipulation to achieve the correct data representation.

So, for example, Figure 10 shows a visualisation created using these techniques. It is a simplification of Figure 42 in Section 8.2.1. Each of the shapes in the diagram is an Excel chart (hollow pie charts and stacked bar charts), based on spreadsheet data and with colours assigned consistently from a standard palette. These had all text removed, and VBA scripting was used to set their sizes based on further spreadsheet data. They were then positioned manually and annotated manually with identification letters.

[17] We used a Python implementation [121], using a Jupyter Notebook [97] to read data directly from that Excel spreadsheet.

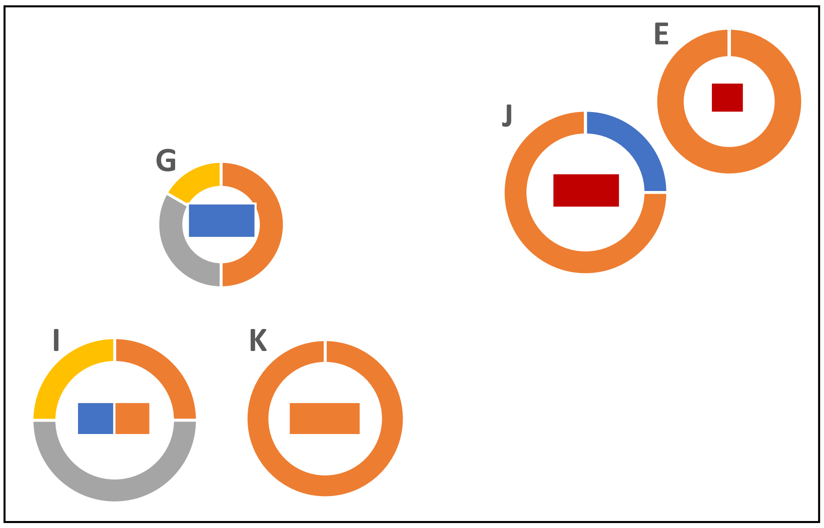

Figure 10: Example Excel Visualisation

4 Expert Survey

4.1 Introduction

As a first step, the author wanted to understand industry best practice with regard to interventions.

In prior work [183], the author had carried out a Constructivist Grounded Theory [33] study, involving face-to-face interviews with a dozen experts in security related to mobile app development, whose cumulative experience totalled well over 100 years of security work. The early analysis of the interviews [191] found a wide range of difference between interviewees, and concluded that there was little consensus in the industry. Deeper analysis of the same interviews [192] then identified that the most important and successful secure development techniques share a quality called 'dialectic,' meaning learning by friendly challenging. Based on this observation, the analysis then highlighted a range of 'assurance techniques' used within industry to find security issues using dialectic.

The next step was to find ways to introduce such techniques to development teams who were not yet using them. None of the techniques, nor the dialectic concept, appeared specific to mobile app development. Indeed, though the previous study's interviewees had been chosen for their experience with app security, in discussing the questions most had cited experience with a far wider range of software domains. The author therefore conducted a second interview survey to find ways to induce teams to adopt assurance techniques, extending the scope to cover professional software development in general.

The research question for this next step, therefore, was:

RQ 2 *What interventions can change the environment for members of the development team to achieve good security, considering cost-efficiency, motivational factors, choice of tools, supporting processes, culture, awareness, training and skills?*

The author accordingly interviewed sixteen experts in secure software development, asking them about these topics. These specialists ranged from senior experts in the major multinational online service providers to solo consultants.

The analysis of those interviews identified a consistent perception of the developer as an active agent in their own right, whose decisions could be influenced by security experts but not controlled by them. More specifically, it identified six assurance techniques recommended by the experts, and a further two techniques used by the experts to help introduce them to development teams.

This chapter describes the survey and analysis in detail.

4.2 Approach

Section 3.2 explains the choice of analysis method: Grounded Theory. The method is described in detail in Section 3.3.

4.2.1 Research Sub-questions

The wide scope of the question

RQ 2 *What interventions can change the environment for members of the development team to achieve good security, considering cost-efficiency, motivational factors, choice of tools, supporting processes, culture, awareness, training and skills?*

led to a need for a more specific focus. Accordingly, in the analysis the author focussed on two further questions, one addressing the approach to creating and introducing interventions:

RQ 2.1 *What approach to interaction with software development teams leads to the best results in encouraging secure development?*

And one addressing the practical aspects:

RQ 2.2 *What specific intervention techniques do specialists consider most cost-effective in helping developers improve security and privacy in their code?*

4.2.2 Research Participants

Interviewees were chosen opportunistically; the author's personal connections in industry provided introductions to a number of successful, and mostly senior, practitioners with considerable experience of helping teams achieve software security. Table 6 shows the interviewees, with an indication of organisation size: Solo for consultants working with a variety of organisations, Medium for organisations between 10 and 1000 people, and Large for larger and government ones; also, a description of the interviewee's main day-to-day role. Each interview was with a different organisation, other than P14/P15; and P12/P16 who were interviewed together. Most are based in the UK other than P10, P13 based in Germany, and P5 in the USA.

Each expert worked with a different set of software development teams. The table also gives a subjective indication of the 'secure software capability maturity' [90] of those teams, based on the author's observations during the workshops: an indication of how expert we might regard the teams involved at delivering secure software, and therefore at what level the expert was normally working:

Low	Teams had little or no awareness or activity related to software security
Medium	Teams were aware of and addressing security issues, typically including some developers with good security knowledge.
High	Teams are expert at software security, within an organisational culture that assigns it a high priority.

Table 6: Experts Interviewed

ID	Org. size	Organisation type	Est SCMM	Main Role
P1	Medium	Outsourced software developer and consultant	High	CEO
P2	Solo	Security consultant	Low–Med	Consultant
P3	Large	Security and military supplier	High	Team leader
P4	Large	Research organisation	Medium	Research and support
P5	Large	Operating system supplier	High	Security team leader
P6	Large	Security and military supplier	High	Security expert
P7	Medium	Software security tool supplier	Medium	CEO
P8	Large	Telecommunications provider	Medium	Security expert
P9	Solo	Security consultant	High	Consultant
P10	Large	Software package supplier	High	Security expert
P11	Medium	Software security service supplier	Low-Med	Training and consultancy
P12, P16	Medium	Telecoms service provider	High	Security expert, Team lead
P13	Large	Research organisation	Low-High	Research and consultant
P14	Medium	Outsourced software developer	High	Principal engineer
P15	Medium	Outsourced software developer	High	Security expert

Figure 11 visualises the same data, illustrating the range of participants. The horizontal axis indicates their organisation size; the vertical axis the ‘secure software capability maturity’.

To show further the range of participants involved, Figure 12 summarises the recruitment process. Various people were approached; the horizontal axis indicates how long the author had known the people approached. Some were interviewed directly, and these are shown closest to the axis. Professional and personal contacts are shown above the axis, and contacts encountered via academia—mostly encountered at industry-academic workshops—below. In other cases, the contacts approached referred others, and the resulting interviewees are shown further from the axis.

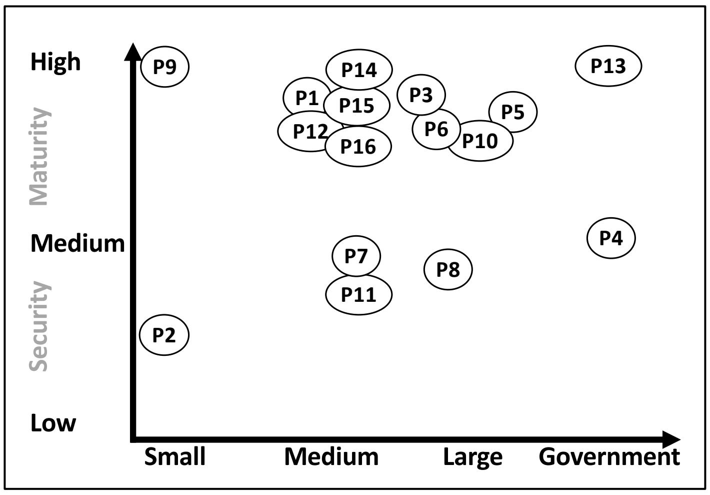

Figure 11: Organisation Size and Security Capability

Those chosen were experts in secure *software development*; many were therefore predominantly developers first and security experts second. As Table 6 shows, only ten of the fifteen had roles or company missions specifically related to security. Their qualification as experts was based on their reputations, either directly from the researchers' knowledge; or as validated by the people who referred them.

4.2.3 Interview Questions

We consulted the interviewees as experts, rather than analysed them as subjects. Our questions aimed to draw out what they themselves had found most effective, and what they had seen to be most effective in other teams. Figure 13 shows the main questions we used. These were generated with an initial ideation session, adding further open questions based on the Appreciative Inquiry (Section 3.3.5), and detailed sub-questions based on findings in the author's earlier interviews of App Security Software Specialists [183].

In the following sections, quotations from the interviewees are in *italics*. Quotations are edited to protect confidentiality and indicate context: square brackets show additions and replacements; ellipses show removals.

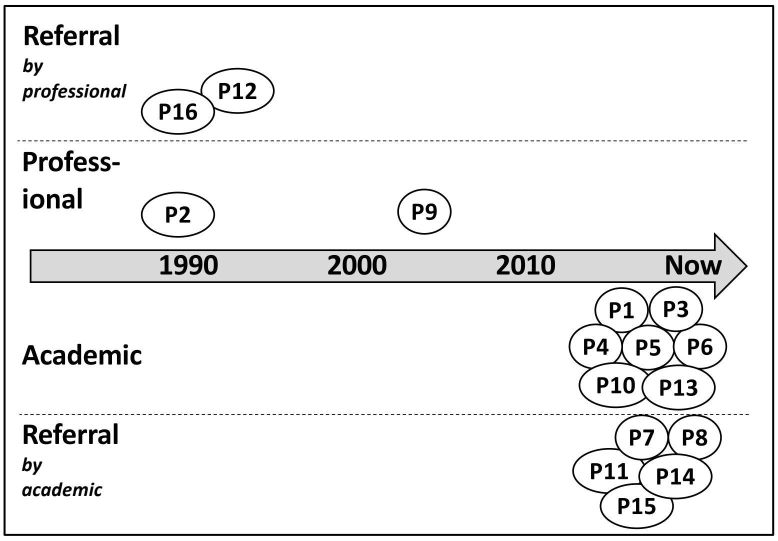

Figure 12: Recruitment Methods for Expert Survey

Introduction – establish context

- What is your current role, and what do you find yourself doing day-to-day? Tell me about a typical day at work?
- Briefly, how did you first get involved with secure software development?

Exploration

- What's your interest in security? What do you do about it, and how do you deal with it day-to-day?
- What do you want to achieve when you're helping a team improve software security? How do you define and measure success?
- What is the most successful intervention technique you've found? Where do you concentrate your efforts?
- Can you think of a particular triumph in your work – where you've worked with a team that has improved their security? How did you achieve that?
- Have any of your teams used code checking tools? How happy were you with their effectiveness at finding problems; and their ease of use?
- What do you find effective as motivation for secure development?
- How do you frighten developers into security, or emphasise the positive aspects?
- To what extent are laws and standards helpful in getting teams to be effective at software security? How do you find out about them and keep up to date?
- When new people join an existing team, how do you motivate them and how do they learn what's required? Do you encourage double checking of contributions from new people or treat them 'as usual'?
- What are the best ways you've found to get teams to tackle specific things:
    - Security coordination with other teams;
    - Reviews and penetration testing;
    - Designing to get feedback from the users?
    - What else?
- Have you had a nightmare scenario? Or consider this nightmare scenario. You're working with a team that's just learned they have a security flaw in a website that's very heavily used. Have you even had a situation like that (no details required)? What did or would you do to help the team tackle it?

Vision

Let's imaging we're a few years in the future, and the problem of getting teams up to speed with app security has been licked; it's now a part of everyday software development life. How was it done? What were the first small steps?

Clarification (as appropriate)

- And how did you achieve that?
- Oh I see. Could you give an example?

Figure 13: Expert Survey Questions

4.3 Results: Active Developer Model

Section 3.3 describes the analysis approach. The audio of all the interviews was recorded transcribed, and coded by the author using the NVivo tool [137].

The 12 interviews generated 15 hours of audio. The final code book had 14 top-level categories and a total of 409 codes and categories, applied to 2977 references in total.

In line with the tenets of Grounded Theory, the researchers' primary aim was to find a 'Core Category', a concept that covered the widest possible scope of the points raised by the interviewees. This Core Category then formed a basis from which to construct theory and to view the practical findings from the research.

In the analysis, the common Core Category found was what we termed the 'Active Developer' model, where developers themselves are the agents driving security adoption. Figure 14 and Figure 15 illustrate the difference between the Active Developer model and the model implied by traditional security literature. In traditional literature, the role of the intervener is to tell the developers what to do, and to provide the techniques and tools that the developers are required to use. In the Active Developer model, the role of the intervener is to sensitise the developers to the implications of their security needs; they then choose for themselves which tools and techniques best work for their situation.

4.3.1 Description of the Active Developer Model

One interviewee described the difference between the Active Developer Model and the model implied in previous security literature as follows:

> *It's not just about educating the developers, well, I guess it was, but we had to get the developers on side, the developers had to understand why we were doing this, as well as what it was that we needed them to do, so it was a kind of two pronged thing. (P2)*

Others simply talked in terms of security motivation as a fundamental requirement.

> *People need to be motivated. As a unit you are motivated; as an individual you are motivated. (P4)*

They were clear that even those with significant power in the organisation still have to work by persuasion, not coercion. For example, even though P8 is a Head of Security for a large multinational company, he said:

> *So, working with Dev Teams, I think the important thing, is to get them to buy into the approach and to understand the value of what you are asking them to deliver. ... You can't go in and say, 'you must do it this way', it would never work, they would just say p*** off, who's this idiot, to come in and tell us how to do our work. What you have got to do is go in there, and you have to convince them that it is to their advantage to do it that way. So, you have got to sell the benefits of any particular approach and persuade the lot of them to follow the same common approach. (P8)*

Some organisations have incorporated this change in thinking into the way they approach software development altogether, giving their developers the power to arrange their own processes to incorporate security in the ways they think best:

> *We have really changed in the last couple of years, in redesigning our [Software Development Lifecycle] processes, based on how product groups*

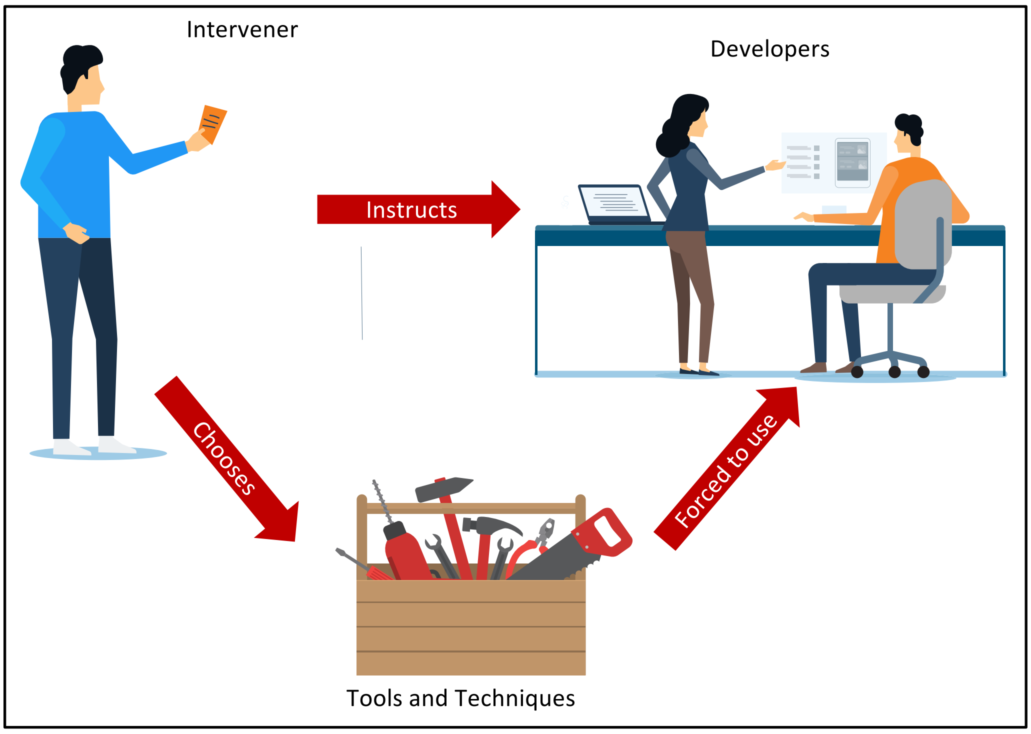

Figure 14: Traditional Model from Security Literature

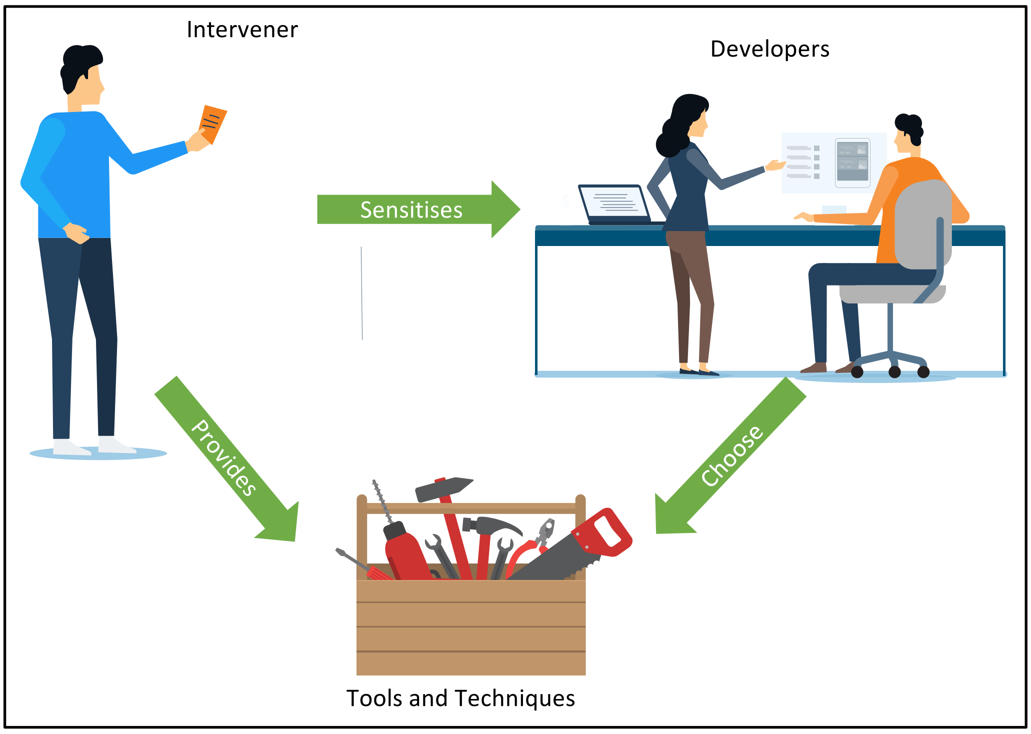

Figure 15: Active Developer Model

work. Rather than in the year 2000 when we launched the SDL: "this is the process – everyone's 'thou shalt do' irrelevant of how you work", now we say "here are the characteristics we want you to employ; how you build that into your process is up to you". (P5)

However, not all of the interviewees felt that motivation was the primary problem; one felt that skills were more important.

I don't think the main problem is lack of motivation, I think it is lack of skills. And I think most people want to do a good job, and want to write secure code. (P1)

But all agreed that motivation to do security was essential.

It is part of the motivation to be aware that real actual harms can follow, when people don't get this stuff right. (P1)

All of the interviewees, in one way or another, phrased their discussion and interventions in terms of the developer as the agent making decisions, and the intervener as persuader.

You need to have the good arguments to convince people to work on [security] ... – the motivation aspect. (P10)

They therefore evaluate their interventions' success in the extent to which the developers themselves changed.

The rest they did all themselves, I would call that a triumph. (P10)

4.3.2 Active Developer Model and Interventions

The Active Developer model helps to explain which interventions will be most effective. The most helpful interventions are not those that identify the most security defects, but rather, those that have most impact in persuading developers of the need to deal with security issues and the possibility of their doing so.

For example, in talking about automated code review tools, their primary importance is their effect in motivating developers.

Because, in the end, this code scanning tool is not fixing any bugs on its own. It is, hopefully, motivating the developer to fix them. But if that motivation is not working, then the tool is also not effective. (P13)

Moreover, usually the intervener gets only one initial shot at this persuasion, and therefore the choice of interventions is vital; most valuable are interventions like Threat Modelling that establish a positive relationship for future work.

Yes, I mentioned beforehand, if you want to motivate people in a cross-development team environment, you cannot come by two or three times for the same topic to try to elaborate on that, to describe it to them. ... We were talking about threat modelling as one concept, the Microsoft conception, of which I am a big fan of, which not only this kind of check mark, or checklist that you have to go through, but it allows you a more interactive base, and by that, having a better relationship with the development team. (P10)

It follows that an important attribute for successful interventions is:

Sensitisation: Helps sensitise developers to security concerns

A second implication of the Active Developer model is that developers have the choice to accept or reject interventions individually or as a team, and the default will be to reject:

The technical inertia is such that doing anything new or radical is seen as risky... And therefore the default choice in those organisations is do the

> *same as last time, because that won't get me fired, if I'm the project manager or technical authority. (P15)*

Therefore, the developers must see the interventions as worthwhile in themselves to achieve the goal of security: the interventions must be sufficiently easy to introduce, yet deliver significant impact in terms or teaching or helping with security.

> *Unless you have a way of getting people to use the Thing, and make it the easiest thing for them to use, and then wherever they are going to deploy is already pretty set up, you need to have all of those steps, you can't say ' this one thing, solves that one problem'. (P9)*

This can by represented as:

Support: **Helps a developer efficiently to achieve the goal of security.**

Finally, each intervention needs to be acceptable to the organisation. While there may be other considerations, the primary consideration will be cost: financial cost and time cost in development and management time.

> *There is always pressure on delivery, delivery, delivery, get that product shipped on time, to a price. So, quite often, security will impact that delivery timeline, and impact that delivery cost. (P8)*

For every team, there is always a trade-off between cost and benefit for each intervention.

> *This is a strange tension here for our team, because we work for ... a FTSE 100 company, and vendors of static analysis tools might reasonably expect that we have an infinite budget, but we are broken down into teams and teams, and they are not selling us a site wide corporate enterprise license, so when they come to me with a quote for 10 users for £30,000 a year – that sounds quite a lot... because it is always a business case. (P3)*

Thus, a third important consideration for the adoption of an intervention is:

Affordability: **Has an acceptable cost in terms of effort and financial impact.**

4.4 Results: Techniques Used

Next, the Grounded Theory analysis of the interviews identified six software assurance techniques as the most effective. These are as follows:

Threat Assessment	Identifying and ranking the threats to computer software, a component, or an IT system.
Stakeholder Negotiation	Discussion and negotiation with stakeholders, such as product managers, on security choices
Configuration Review	A review of the way a system or its software has been configured to see if this leads to known vulnerabilities, using manual checking software versions or automated build review scanners.
Automated Static Analysis	The process of using an automated scanner on a web application or network to identify vulnerabilities.

Source Code Review	The manual examination of source code to discover faults that were introduced during the software development process.
Penetration Testing	A simulated attack on a component or system, carried out by a security expert using similar techniques to that of a real-world malicious attacker.

For consistency with other literature, the definitions above are taken from Such et al. [164], except for Stakeholder Negotiation, which was not discussed in that paper; and Automatic Static Analysis, since Such's name for the technique, 'Vulnerability Scan', proved confusing. Note that the author was not aware of that paper at the time of coding, and the assurance technique names derived from the Grounded Theory analysis were different. This set of effective interventions is considerably smaller than the full range of industry assurance techniques, which include also Self-Assessment Forms, Architectural Review, Automated Static Analysis, Formal Verification and seven others [164].

The contribution of the Grounded Theory analysis was not the identification of the assurance techniques—they were already well known—but the identification of a practical subset of the techniques suitable for cost-effective use in development teams, and the justification of that subset in terms of the Active Developer model.

The Grounded Theory analysis also identified a further two techniques that the experts used when intervening with development teams. These are to support the developers themselves, and are as follows:

Incentivisation Session	A presentation, discussion or workshop to help motivate those involved for the need for security.
On-the-job Training	Mentoring or informal workshops, used regularly with the development team.

4.4.1 Technique Names Used in this Thesis

A complication in discussing Developer Centred Security is that there is no standard taxonomy of Assurance Techniques. As far as possible this thesis uses the names defined from Such et al. [164], but those were taken from the Security Expert domain. They are not consistently used there, nor are they necessarily the ones that software development teams would use; indeed, the author found the need to use different names in talking to different groups.

Appendix C shows a table of all 14 different Assurance Techniques mentioned in this thesis, along with the names for and references to each in the different phases of the research. So, for example, from the Grounded Theory Analysis of the expert surveys discussed in the previous section, we identified a technique the experts called 'Plugin reviews'; we identified that it was the same as Such et al.'s 'Configuration Review', and used that name in this chapter. The table also shows the names used in the research in later chapters.

In some cases (Red Teaming for example) we identified that a technique was insufficiently important to the developer community to justify being separate and we described it along with another technique (Penetration Testing, in this case).

4.4.2 Expert Discussion of the Techniques

Table 7 shows the participants' discussion of these techniques. The numbers indicate the percentage of the words by each participant that were coded to each assurance technique: the share of the discussion devoted to that technique within that interview. Cells are highlighted based on their values. The most discussed techniques are on the left; less discussed ones towards the right.

Sections 4.5 to 4.10 explore each assurance technique in detail, discussing the implications for each. Each section discusses the context of the problem, and outlines how the technique solves it, exploring how the interviewees use the technique, evidencing the discussion with quotations from interview participants

4.5 Technique: Threat Assessment

Threat Assessment is the activity of identifying and ranking the threats to computer software, a software component, or an IT system as a whole [164].

4.5.1 Context

Any system can be broken with sufficient determination, ingenuity, and resources.

> *You can't defend against an attacker who has unlimited resources, you can't do it. Not for long. (P8)*

As a result, secure development is not a matter of making a completely secure system. Instead, it becomes a question of which defences to implement: where one should spend the time and effort defending the system to deter the largest and most damaging potential exploits. Making those choices requires an understanding of the potential attackers:

> *But, from the vulnerability side, it is all about assessing how we are. (P16)*

> *You need developers to ... understand how the attackers brain works, what is the methodology, the way I know how to do that is to make them do the steps, make them ask the questions (P11).*

4.5.2 Solution

To address this, security experts use 'Threat Modelling' techniques: identifying the causes or motives and possible scenarios for a full range of threats to the systems in question.

> *Your answer to any kind of security question anywhere should almost always start with a threat model. (P9)*

Several interviewees indicated that developers must drive the threat modelling process:

> *You need developers to do threat models, but you need developers to understand how the attackers brain works, what is the methodology, the way I know how to do that is to make them do the steps, make them ask the questions. (P11)*

Table 7: Interviewee Discussion of Assurance Techniques

ID	*Role*	*Organisation*	Automated Static Analysis	Penetration Testing	Code Review	On-the-job Training	Incentivisation Session	Product Negotiation	Threat Assessment	Configuration Review
P1	*CEO*	*Outsourced software developers*	4	23	7	2	3			6
P2	*Consultant*	*Security consultancy*	7		2	2	7		2	
P3	*Team leader*	*Security and military supplier*	14	2	13	10		1		
P4	*Researcher*	*Research organisation*	4	4		4			2	
P5	*Security team leader*	*Operating System Supplier*	7	5	3	7	8	5		2
P6	*Security expert*	*Security and military supplier*	2	4	4	1	1			
P7	*CEO*	*Software security tool supplier*	33		2		4			2
P8	*Security expert*	*Telecoms provider*	12	1	1		3		2	
P9	*Consultant*	*Security consultancy*	5	6	1	1	3	3	2	3
P10	*Security expert*	*Software package supplier*	7		1	8	4		4	1
P11	*Trainer and consultant*	*Software security service supplier*	17	8	5	8	8	2	2	
P12	*Security team lead*	*Telecoms service provider*	4	3	5	4	4	8	1	
P13	*Researcher*	*Research organisation*	12		6		6	2	6	
P14	*Principal engineer*	*Outsourced software developers*				8	2	2		
P15	*Security team manager*	*Outsourced software developers*	25		20	2			2	

4.5.3 Execution and Counterparties

The counterparties discussed for Threat Assessment included other developers:

Threat modelling: what I see as the big benefit here is, is that it is not that you are coming up with a list of issues that product of the team has, it is

> *more putting the team into the perspective, to think about the functionality from a different aspect, from a different point of view, that that brings up a list of issues, no doubt about that, but it also fosters the understanding to see the coding from a different side, to also to think about it when you create code: what else can there be? (P10)*

Some experts emphasised the importance of including more senior stakeholders:

> *If [the developers] don't know what a threat model is, then tell them what a threat model is, and the simplest explanation in the world is 'who is going to attack you, and where is the gold?' 'Where's the gold' answers need to come from managers, and company owners. Not from developers because their answers will be totally wrong. And actually that is a big learning that you can give to any company when you go in. Whatever the CEO's threat model is, explain it to all of the devs and go 'if you don't know what the worst nightmare is for your CEO, then how can you be expected not to make that cock up' (P9)*

For developers more experienced in secure development, threat modelling can become part of normal requirements gathering, and no longer be considered an explicit separate process.

> *We did requirements engineering based upon the work of Michael Jackson's Problem Frames model, and as part of that you sit there and you do your security engineering, you look at assets and threats and risks, and all of that. And you decide what you are trying to protect, and how much we need to spend to protect it (P15)*

4.6 Technique: Stakeholder Negotiation

Stakeholder Negotiation is the activity of discussion and negotiation with stakeholders, such as product managers, on security choices.

4.6.1 Context

Merely identifying the possible attackers and exploits does not itself deliver software security. The need is to prevent them from causing damage to users, stakeholders or others.

Given the Threat Assessment, a development team can take the list of possible attacks and work out possible mitigations for each. These mitigations will each have costs in development time, commitment, finance, and sometimes usability. The team can estimate financial and other costs for each. How, though, do they make the decision which to implement?

The decision of what aspects of security to implement is a commercial one. Implied in every decision about software security is a trade-off of the cost of the security against the benefit received. Every security enhancement needs to be weighed against other uses of the investment—financial, time, usability—required.

4.6.2 Solution

The solution is for development teams and security experts to express risk and costs to stakeholders (project managers, senior management, customers) in terms they can understand and use to question about security concerns against other organisation and project needs.

> *[If] it looked like there were some problems …, then they would mark it like an orange, and say to the customer :'what is your risk appetite?(P9)*

4.6.3 Execution

The primary stakeholders are those who prioritise development tasks, such as product managers. Evaluating and expressing the threat in business terms requires discussion and experience, so other Team Members and Security Experts are also important:

> *The teams take a risk management approach, full stop. For every risk, we try and weight it on two axis – one is the severity, if it happens. So, ignore the likelihood, how bad is it if it happens? And the other is the likelihood, how likely is it to happen… so the project teams will plan their work, and come up with an end date, and the project manager will make sure that they add enough capacity to the project for the weighted risks to happen. (P14)*

Many of our interviewees made the point that 'security is not an absolute,' but that security is what the users and stakeholders need for a particular situation at a particular time. There are techniques available to give objective assessment of security risks, such as work by ben Othmane et al. [127].

The stakeholders will be making cost benefit trade-offs comparing various business risks. Given that each mitigation now has a cost and benefit, the decision on whether to do it becomes part of standard project management process. It is outside the scope of this technique – and indeed of the topic of software security – to explore how to make these decisions in general; the balancing of risk cost and reward is a well-understood aspect of business life.

> *And that's what managers do all the time, they make common sense decisions – they refer to it as common sense, but they make risk decisions: that is the way security look at it. The way managers look at it, they make common sense decisions all the time … and that is their job, and they manage time, and they manage resources, and they know there is a trade off with all of these things. (P9)*

4.7 Technique: Configuration Review

Configuration Review is a review of the way a system or its software has been configured to see if this leads to known vulnerabilities, using manual checking software versions or automated build review scanners [164].

4.7.1 Context

Most software development reuses code developed by others, whether purchased or obtained 'open source'. In this section, such reused code is called 'components', and includes frameworks and toolchains.

Using an insecure component automatically makes a developed system insecure, regardless of the quality of the code developed by the team:

> *WordPress plug-ins are an enormous liability. Anyone can write one, most of them are rubbish. Anyone can get them put up on the plug-in directly, which gives them this air of authenticity, and quality that they don't deserve, frankly. (P1)*

So, a 'low hanging fruit' for development is to use only components that are well written and securely implemented. This is non-trivial, given the wide range of components available.

4.7.2 Solution

Configuration Review is the activity of reviewing the codebase to evaluate the components used for security: using public repositories of security vulnerabilities to assess well-known components; avoiding little-known components where possible; and using Source Code Review and Penetration Testing to evaluate such components if they must be used.

4.7.3 Execution and Counterparties

For some development systems there are corresponding web sites with security reviews of plug-ins; cross referencing with these sites is a powerful security technique. This may use automated analysis tools:

> *[Our tool chain] also queries Wpvulndb for the plug-in that you are expecting and tells you if there have been any published vulnerabilities in it. (P1)*

Where such sites are unavailable, or for new plug-ins, there is value to Source Code Review (Section 4.8) to establish the likely security attributes of a given plug-in. This does however have a significant cost to development teams since it takes effort, however much automation may be involved, and restricts the plug-ins that developers can use.

> *And so that is one of the things you end up sitting down with developers going "I'm sorry, but I know this is actually going to slow you down". And we are desperately normally trying to avoid that, I'm trying to make your lives as easy as possible. ... But you have to say "well, no, you can't just add [components] – you have to review them. You don't have to do the most detailed review in the world, but if you think it looks worrying, then ... don't put it in your code". (P9)*

A second issue is that, since plug-ins are widely shared, any weakness in a plug-in becomes known to attackers, and therefore it is important to keep plug-ins upgraded to the latest versions in which defects have been corrected.

> *We keep track of all the patches and everything for all our systems. (P7)*

Since Penetration Testing tests a whole system, it will also find vulnerabilities in components, so testing is another way to assess components:

> *[When Penetration Testing] you'll find the OWASP Top 10 in one [component] alone! (P9)*

4.8 Technique: Source Code Review

Source Code Review is the manual examination of source code to discover faults that may have been introduced during the software development process [164]. Note that throughout this Thesis we use the term 'Source Code Review' to refer to review for security and privacy issues, rather than more general code review.

4.8.1 Context

It is notoriously difficult to spot one's own errors. This is especially true when the errors are faults in complex reasoning or are due to misunderstandings. Thus, a programmer working solo is likely to create avoidable security problems, just because they can naturally have only one point of view.

> *So, it is very easy when you are trying to deliver something yourself, as a developer, to pass over the bit that you are not doing* [188]

This problem extends to programming teams. A team will always suffer to some extent from 'groupthink;' the need to generate a shared understanding brings with it the danger that that understanding may include misunderstandings and blind spots. This is particularly important with software security, since such blind spots often lead to vulnerabilities in the developed software.

4.8.2 Solution

In Source Code Review, development teams provide a counterpart who reviews the security and privacy aspect of assumptions, decisions, and code. Several interviewees stressed the importance of this questioning process for security reviews:

> *What is our most successful technique for secure software? In terms of what I have seen, certainly talking amongst the developers, the code reviews have been very useful. (P16)*

> *I would say, without a doubt, [our most effective technique for getting teams more secure] is code review. This affects everything we do, and our newest grads understand this from the very first day they join our team. (P3)*

4.8.3 Execution and Counterparties

The reviewer is typically another developer from the same team.

> *And that is something that is a process within the team, and the way that we encourage that – first of all, all the teams I have worked with, do some sort of at least cursory code reviews. So somebody will look at the code. (P11)*

If a Security Expert is available, however, they can often provide more security-relevant questioning than development team members:

> *[We have] 'software pen testing,' where you have some Subject Matter Experts who take a piece of code and review that in detail, and work with the developers in tandem, looking at the code and saying, "we think this is really high risk, we really want to look at this." And then somebody goes through that code with the mindset of "how do I exploit the code" (P5)*

Other approaches to Source Code Review include providing dialectic questioning to a developer via pair programming, where the code is written by two developers sharing one computer:

> *We have [security reviewed] a couple of companies that already use concepts like pair programming and so on. (P13)*

And enforcing an informal check of code before it is released:

> *Code review can [just] mean that every bit of code that is committed is looked at by somebody else (P11).*

The heavyweight approach is a formal review process, with separate review meetings delivering lists of defects for a developer to fix.

> *It is in the culture. We do reviews; we always have to do reviews. We set things up – and it is not regarded as a second class. (P6)*

4.9 Technique: Automated Static Analysis

Automated Static Analysis is the process of using an automated scanner on an application or network to identify vulnerabilities [164].

4.9.1 Context

It is a poor use of expensive resources to find problems that are cheap to find in other ways. Source Code Review, Configuration Review and Penetration Testing are all expensive in financial or human effort:

> *[A pen test] is quite expensive. We don't do cheap ones. [My company's] pen test will be upwards of £8 grand or so. (P1)*

> *In particular in small teams, ... you can't spend the additional effort in doing reviews. (P13)*

Indeed Such et al.'s survey of Assurance Technique costs—cheap, moderate or expensive—for these three techniques, taking the modal choice, are that Source Code Review and Penetration Testing are expensive; Configuration Review is moderate [164].

4.9.2 Solution

Automated Static Analysis uses automated tools to look for possible security flaws in the written code. Tools to do this are sometimes called 'lint' checkers, after a UNIX tool that does extra checking for C code:

> *There are many tools that are for looking at things that could be helpful in [checking code]. There are Linters, there are all sorts of things (P1)*

Such et al.'s results suggest that Automated Static Analysis is cheap [164]. Indeed, automated tools can be used as an extension to the compilation process of the code, and many interviewees saw them as valuable in automating the removal of certain classes of security bugs:

> *We use excellent tooling from the Alassian stack, the Crucible Tool ... We do a lot of static analysis. We review for security, we certainly do, but the*

> *point is, is that we will try to automate the removal of whole classes of problem from that. (P3)*

4.9.3 Execution

There are now many such tools, some produced by commercial companies, supporting different languages and purposes. They work similarly to compilers, and generally use analysis techniques developed for compilers. Since codebases vary enormously in their style, requirements, and ways of using code, such tools often require significant work to configure or rebuild:

> *Certain tools are very good with custom rules, they are very easy to tweak to try the custom rules. If someone is doing something different from a web application, I contend that they need to find a static analysis tool where they can write the rules easily, because they are not going to have cross site scripting, they are not going to have the standard thing that the tool looks for. (P11)*

Such tools typically generate large numbers of ‘false positive’ warnings. So, developers need to use them as questions (‘is this piece of code OK’) rather than as notifications of changes to be made. This is particularly relevant when examining components (see Section 4.7):

> *We have a tool that we wrote … for checking … plug-ins, which is intended to make code reviews of plug-ins more focused. It looks for things that are indications of badness and you go and review that list of things, rather than sitting down and reviewing the entire plug-in.(P1)*

The benefit of using such tools is that they can reduce the time to find vulnerabilities:

> *I like them because I think there is no value – this is almost a philosophical thing – there is no value in a human being finding a vulnerability, if that vulnerability can be found automatically in a second by a vulnerability tool. (P11)*

Some interviewees, however, had found the time cost of analysing the responses to be too great:

> *If you mean static analysis type tools, the answer is no [we do not use them]. Two teams I have worked with have used them. One gave up entirely, because they looked at the amount of time that they were spending on it, versus the reward that they were getting, and then looked at what happened in the first pen test that they had, where they just got completely pwned. (P9)*

4.10 Technique: Penetration Testing

A Penetration Test is a simulated attack on a component or system, carried out by a security expert using similar techniques to that of a real-world malicious attacker [164].

4.10.1 Context

All the assurance techniques discussed so far (Sections 4.5 to 4.9) need access to the development team or source code. Yet the delivered system is what the attackers see and where the privacy issues occur:

> *I like to do [external reviews to authorise projects to go live] on a White Box basis (P8)*

So, there are situations where none of the other assurance techniques will help.

4.10.2 Solution

In Penetration Testing an external 'white hat' security team simulates what an attacker would do to attempt to gain access or disable the service. The team then feeds back any 'successful' exploits they have found to the development and operations teams:

> *[The most successful intervention technique we have found] comes down to using security experts. We ... have Penetration tests. (P5)*
>
> *If the team have developed something new… and it is a significant change, we might get it externally pen tested, if we think that we can't test it ourselves. (P12)*

Many security specialists offer external Penetration Testing as an explicit service:

> *You sit down and ask what their worst nightmare is, their second worst nightmare, and you deliver that in as many ways as possible, and anything else you think is interesting, that you can do in the time period, and then the really important bit is the non-sexy end of that, which is a big long detailed report… The two sections that people care about is the executive summary which is one page and a non -technical person can read the entire executive summary, it does not contain anything technical, 'I can do the following things' It doesn't say how I've done it. (P9)*

4.10.3 Execution

Essential for this assurance technique is a Penetration Test Expert. Few developers have more than basic skills at Penetration Testing, so it would be unusual for a development team member to take the role. Instead, the role of developers is to take the results of the Penetration Test and use them to plan future security enhancements:

> *Normally the way the report would be dealt with, we would sit down in a sprint planning meeting and we would go through the report: us, the client, sometimes the tester, ... and we would discuss the findings and produce work that would be added to the next sprint to address what had been found. (P1)*

Some interviewees have success extending the Penetration Testing to involve direct discussion with the developers:

> *So, the idea is the penetration tester is testing the web application, and the developer is sitting with the penetration tester. That is very effective. ... I do not claim to be able to find all the vulnerabilities in an application ... but if you are sitting with a developer they are part of the process and they tell you 'ah, you know that request you have just intercepted in Burp: I'll*

> *change it to that!' And you find out that you have completely broken the authentication and you wouldn't have found that on your own, because you really need to have the knowledge that the developer has about the application. ... I would say it is also much quicker to do a pen test with a developer who spends the whole time with you. (P11)*

4.11 Technique: Incentivisation Session

Lastly, we turn to the two further techniques used by the experts interviewed in their intervention work. First, an Incentivisation Session, which is a presentation, discussion or workshop to help motivate those involved for the need for security.

4.11.1 Context

Given that a large number of developers have no current interest in security [200], it is vital that at least one intervention generates such interest. The Active Developer Model makes it clear that without such interest, developers will avoid security-improving activities.

4.11.2 Solution

Thus, most of the interviewees discussed a form of presentation or workshop to help motivate those involved for the need for security:

> *Everyone who joins [this company] gets a security talk, when they are a developer that security talk is longer. And it includes examples of things that have gone wrong and why, and how badly these things can go wrong, and how easy it is to screw it up, and some pointers on things to read about, to learn about. (P1)*

Some provide it as a scheduled training course for new employees:

> *I get an email every week, and it tells me how many new starters there are, so I know, okay, yeah, there is about 4 or 5, I'll do an induction. (P12)*

Others make it a one-to-one between the intervener and each developer:

> *The conversation can take anywhere between 40 mins to several hours depending on who the person is, and you won't know until you've had the conversation. And the conversation is: you explain how to break into, how an attacker would attack the systems, and what the various things you need to be aware of, are. (P9)*

Bigger companies may offer a security sensitisation course over several days for every programmer:

> *So we run a very large scale education program ... where we ... tell developers exactly what happens in the real world, how TalkTalk was hacked, how Sony was hacked. And then we go in detail how we have been attacked, and whether they were successful and how they were detected. Then we also show them all the stuff that our red teams do – our internal hackers – which really scares them! (P5)*

Sometime the Incentivisation Session is based on a penetration test of the live systems; the intervener carries out the penetration test, identifies a list of vulnerabilities in the software, and uses those to convince the developers of the importance of improving their security. From the point of view of an external security specialist this may also be an opportunity to establish their credibility:

> *What is often a door-opener for us is, with these companies, we would do an initial project where, for example, we pen test one of their existing products, and show them "this is how you designed this product, and this is how you went through this process, and this is what we found". And ideally, we try to then point out to them "this is where you might have detected it earlier, but this is why it failed, why you failed to detect it". And this is often an eye opener for them, because then they see that a better process might be more worthwhile. (P13)*

4.11.3 Execution

However it may be carried out, the aim of the Incentivisation session is to motivate the developers themselves to understand and prevent security problems. So many such sessions include stories and warnings about the consequences of security failures:

> *I talk to my engineers about this, about look at what is happening in the courts – look at the class actions, look at these things. It is a hard thing to say, but you guys do not get away with saying "it's nothing to do with me" (P6)*

Many covered a variety of different kinds of security problem, not limited to purely technical ones. So, for example, some include discussions about how aspects of team working may cause security problems:

> *So we give them that kind of perspective, it really brings it home: if you do X, this is the result. If don't work together as a team – if you don't work with your dependencies, this is what happens to you. (P5)*

Conventional software security wisdom used to be to 'scare developers into security'. We wondered if this was actually the best approach, and asked our interviewees directly if they did this. The consensus from our interviews was definitely not to leave them scared:

> *I must admit, we had [high-profile expert] to visit and he goes off on one of his rants and scares the shit out of people, and they think "oh there is no point then!" (P5)*

Instead it is essential to leave developers knowing that the problem is solvable, and so each session needs to show that solutions do exist for the problems they are introducing. The Incentivisation Session works by highlighting problems and leaving possible actionable solutions:

> *It needs to be positive. That is why the day is balanced. For every one of these problems we show you in the commercial world and internally, we absolutely have a way of mitigating it. And even if we know we can't stop it, we can certainly detect it, contain it and then exfiltrate those people. (P5)*

Overall, the interviewees designed their sessions to convey a positive aspect to software security: that it is an interesting and exciting topic, a valuable skill to learn, and that secure implementations give value to their organisation and end users:

> *I try to emphasise the positive aspects: "This is exciting, this is interesting." Fear is probably the wrong word, but awareness is the right word. ... The*

> *awareness that someone can attack your application is definitely scary – especially from the perspective of someone who owns the business – there is nothing wrong with that, you shouldn't try to scare them; you try to share solutions to empower them. (P11)*

All of the experts who discussed an Incentivisation Session emphasised this in one way or another.

4.12 Technique: On-the-Job Training

Finally, the experts interviewed also made considerable use of different forms of On-the-Job Training: mentoring or informal workshops, used regularly with the development team.

4.12.1 Context

We had expected a good deal of formal training in the techniques required to provide secure code; however only one interviewee (P11) provided this. More than formal knowledge, developer need practical, contextual, reminders of the importance of security issues and the ways of addressing them in their own work.

4.12.2 Solution

The experts used two approaches to On-the-Job Training: informal workshops and mentoring.

The informal workshops are usually based on security learning from recent project work or from external research, and take the form of presentations by one of the team or an external expert:

> *[Our security specialist] will take the most interesting or most relevant findings for the team out, and those go into a slide deck that we keep, and that deck is used as part of a show and tell. That happens… a few times a year. (P1)*

A different and widely used form of intervention is mentoring of various kinds. This is particularly valuable in that it supports a security-aware team culture:

> *I think it is an issue of team culture, where teams are usually led or mentored by our more senior people who try and set the standard. And teams will say 'we are just not going to have any [security] bugs in this system'. A team culture emerges. And that is always the right way to do it. (P15)*

4.12.3 Execution

Some organisations teach penetration and attacking techniques to developers: sometimes as formal taught courses; sometimes through online resources. The emphasis is to show the developers what kinds of weakness might be present, and also how to prevent such weaknesses:

> *And, today, I was teaching software developers how to basically pen test an application and exploit an authorisation issue. ... They know the technology*

> *and they now understand it. They went back to their application and they started to apply it: "maybe I can do this here". (P11)*

A few had used formal training in the past, and now use online resources instead, thereby reducing cost and making the training easier to arrange:

> *I think training is obviously very effective, and we sometimes do specialised training. ... So we had pen testers coming in, and I have got it so that we can now do it ourselves, where we have got a VMWare image with all the hacking tools on it, and vulnerable webpages, so they can play and see how easy it is – and what issues they need to look for. (P12)*

The Grounded Theory analysis identified three approaches to this. First is mentoring by security experts, who are not themselves developers, but who work with the developers to sensitise them to security issues and address those issues in practice:

> *So there is that side, and then there is a SME, Subject Matter Experts, side, which we call our advisors. And these are people we actually plug into different parts of the service teams, the engineering teams, and they work as a security SME. And they are highly skilled security people. They are people who couldn't write a product, but they certainly know everything about [specific security features], and how would you do that at scale, how would you do the threat modelling. And they work as a team. (P5)*

A second approach is to send a developer experienced in security to work with a less experienced team. The experienced developer acts as a consultant to the team, and as a team member helps to create ways of working that promote security:

> *We send people on site, and we embed them into other teams. The normal outcome, and I can't think of a situation where this hasn't happened, is for them to export our processes like it was the obvious thing to do – and I think it is! And for that then to be taken up by customer [developer] teams. (P3)*

Finally, where neither of the first two approaches is appropriate—or in addition to them—there is the option of encouraging a 'security champion' from amongst the developers within a team. This developer learns as much as possible about the subject, and then provides support to others in the team on security matters:

> *One thing that we find works with software development teams is ... Security Champions – the idea is that one person in the team is more interested in security – not responsible – but who is the 'go to' person in the team if there is an issue in the team before they go to an external consultant. ... You need that person in a team, you actually do. (P11)*

4.13 Discussion

In response to:

RQ 2.1 *What approach to interaction with software development teams leads to the best results in encouraging secure development?*

the Expert Survey established the 'Active Developer Model' theory, that developers must drive the introduction of security improvements.

Table 8: Security Assurance Techniques and Participants

	Members of Development Team	Stakeholders, e.g. Product Managers	Automated Code Analysis Tools	Security Experts	Penetration Test Experts	Other Development Teams	End Users and Operations
Threat Assessment	🟥	🟥		🟥	🟥		🟥
Stakeholder Negotiation*	🟥	🟥		🟥			
Configuration Review	🟥			🟥			
Source Code Review	🟥			🟥		🟥	
Automated Static Analysis			🟥				
Penetration Test					🟥		

Encouragingly, we note that the 'Motivating Jenny' project (Section 2.2.6) came to a similar conclusion of the need to *raise developers' security awareness*, from their later ethnographic work.

In response to:

RQ 2.2 *What specific intervention techniques do specialists consider most cost-effective in helping developers improve security and privacy in their code?*

the Expert Survey established six key Assurance Techniques as the most recommended by the experts for adoption by developers: Threat Assessment, Stakeholder Negotiation, Configuration Review, Source Code Review, Automated Static Analysis, and Penetration Testing. Plus, it established two further techniques used by the experts to encourage developers to drive security: Incentivisation Session and On-the-Job Training.

4.13.1 Choosing Assurance Techniques

Table 8 summarises the Assurance Techniques discussed in Sections 4.5 to 4.10. It highlights the 'dialectic' nature of each of the techniques (see Section 4.1), by showing 'friendly counterparties' involved with each one; such a counterparty may be a tool. Interestingly, end users and operations staff were not mentioned in the context of any of the Assurance Techniques.

We observe that this list of key Assurance Techniques is nevertheless a small subset of the 20 techniques identified by Such et al. [164]: the explicit focus that the surveyed experts placed on them is therefore important.

Based on Table 8 and the expert cost estimates from Such et al., we can further characterise the assurance techniques in terms of two important practical considerations: their cost and their need for security expertise. Figure 16 shows this characterisation, using Such et al.'s modal estimate of cost for each technique, and assigning 'cheap' to Product Negotiation, since it is incorporated into activities already carried out by a development team. In this and the following figures, the Assurance Techniques are coloured according to their type: blue for process techniques; orange for vulnerability finding techniques.

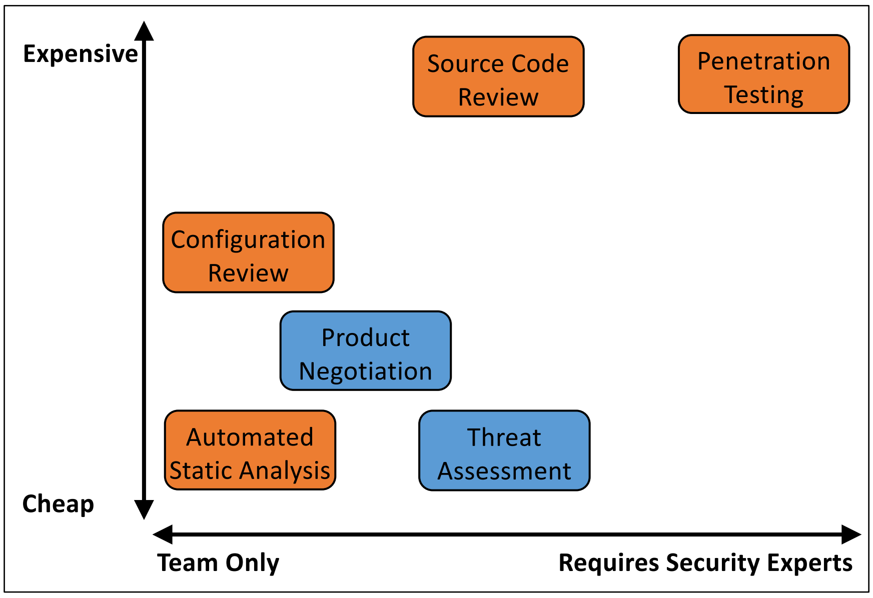

Figure 16: Interventions in Terms of Cost and Specialist Requirements

Based on this, we deduce that the most promising Assurance Techniques are Threat Assessment, Product Negotiation, Configuration Review and Automated Static Analysis; and that teams with greater resources will benefit from adding Source Code Review and Penetration Testing.

Interestingly, the choice of which tools to use for Configuration Review and Automatic Static Analysis, and indeed for the other techniques, was rarely discussed by the experts. We conclude that developers are already highly skilled at choosing between competing tools and methods, and do not need explicit support for choosing security assurance tools.

4.13.2 Incorporating the Assurance Techniques into the Development Cycle

The ordering of the assurance techniques in Section 4.4 is roughly chronological within a development cycle. Figure 17 illustrates how they might be incorporated into an iterative cycle. Threat Assessment and Product Negotiation affect what functionality is

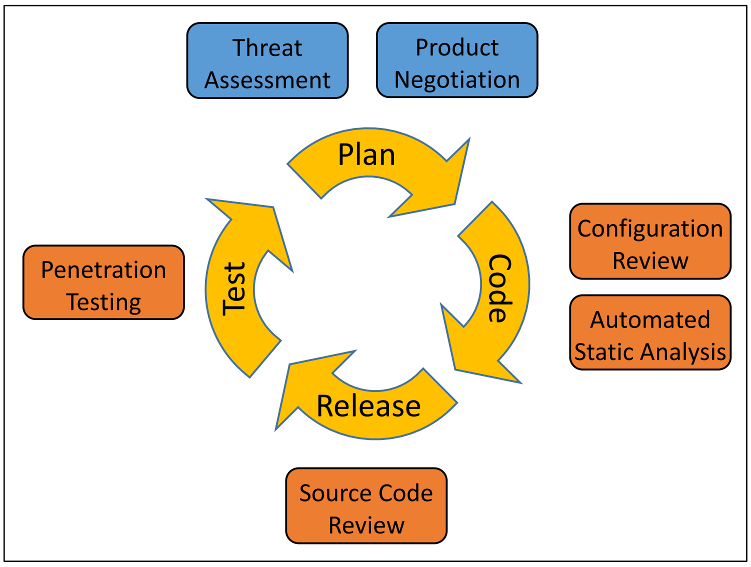

Figure 17: Assurance Techniques in the Software Development Cycle

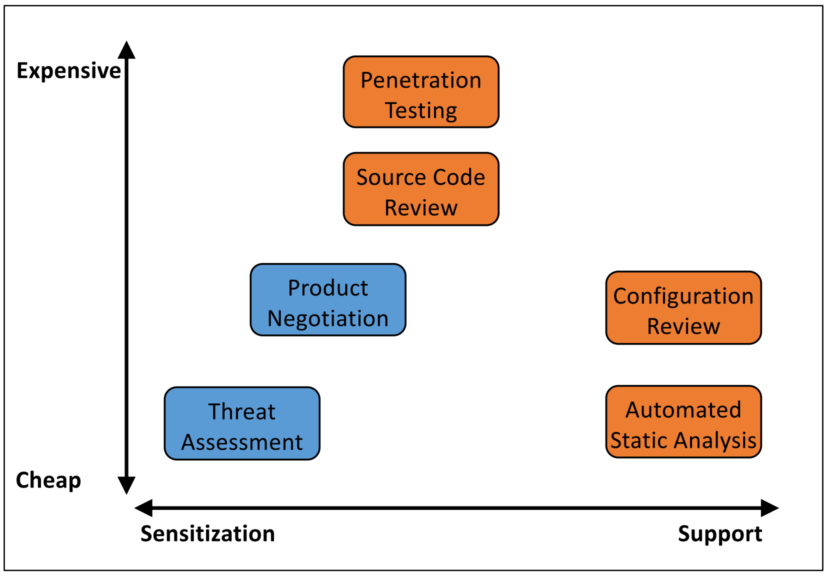

Figure 18: Sensitisation, Support and Affordability

produced, so apply to the planning element; Configuration Review and Automated Static Analysis can be automated into a product build; Source Code Review needs a candidate complete implementation, so is typically done at the release stage; and Penetration Testing applies to an installed system, so comes in the test phase.

4.13.3 Sensitisation, Support and Affordability

The Active Developer model implies that for an intervention to be effective the development team must actively accept it, and therefore it must qualify in terms of Sensitisation, Support and Affordability (Section 4.3.2). Figure 18 shows how each of the identified Assurance Techniques qualifies, showing their effectiveness in providing Sensitisation or Support on the horizontal axis, and their Affordability on the vertical one.

4.13.4 Research Validity

What measure of certainty can we offer for the theory, the Active Developer Model, and for the assertion that the six Assurance Techniques and two further techniques highlighted in the research do represent best practice in Developer Centred Security interventions?

Considering first Conclusion Validity, do the research data justify the conclusions? Grounded Theory's rigorous process of line-by-line coding, categorisation, and sorting generates a theory that does reflect the interview data. The use of extensive quotations ensures that this can be at least partially checked.

In terms of Construct Validity, does the Active Developer theory represent real-world practice? Grounded Theory handles this primarily in terms of 'theoretical saturation', reached when new interviews do not add substantially to the theory (Section 3.3.3). A dozen interviews are often sufficient [77]. We believe we have reached saturation as regards the Active Developer concept and the list of Assurance Techniques, in that further interviews would be unlikely to modify the concept, or the list; obviously they would produce more detail for the descriptions. There is also a risk of bias in the choice of interviewees, and of questions. Examples might include selecting participants using only one form or intervention or one interaction approach; or asking only about certain aspects of their interventions. We addressed this bias with interviewees from a wide range of industry roles, and completely open questions.

In terms of External Validity, can the results be generalised to a wider scope? Grounded Theory's conclusions are always limited to the scope studied [33]. Specifically, the scope in this case covered commercial companies, predominantly in the UK (Section 4.2), and was limited to the views of security experts rather than software developers.

4.14 Conclusions

Combining the answers to the two research sub-questions discussed in section 4.13, leads to an answer to the original research question:

RQ 2 *What interventions can change the environment for members of the development team to achieve good security, considering cost-efficiency, motivational factors, choice of tools, supporting processes, culture, awareness, training and skills?*

The Grounded Theory analysis in this chapter selected eight such interventions to achieve good security. The primary *motivating factor* identified was the Active Developer Model: that developers must drive the security improvements themselves (Section 4.3). The *training* role of an intervener, therefore, must be to generate *awareness* with the Incentivisation Session (Section 4.11), and to promote it within the team *culture* using On-the-Job Training (Section 4.12).

Figure 17 shows how a *supporting process* fits Assurance Techniques into the development cycle. The six Assurance Techniques were chosen for their *cost-efficiency*; as Figure 16 shows the most *cost efficient* were Threat Assessment, Configuration Review and Automatic Static Analysis, plus a new 'Assurance Technique', Stakeholder Negotiation; with Code Review and Penetration Testing were more expensive (Section 4.13.1). The *choice of tools* is of relatively little importance compared with the decision to use them. Similarly, the *skills* required are accessible if required, and achievable if the motivation and culture are right (also Section 4.13.1).

4.14.1 Improvements on Existing Practice

Current practice in interventions is often based on challenges by the intervener based on Penetration Testing (Section 2.2.4). Aside from the high cost [164], this approach proves ineffective in the longer term (also Section 2.2.4). The findings of this chapter suggest a quite different approach: sensitising developers to the importance of security so that they drive the security improvements, then providing cost-effective tools to support them.

The specific techniques suggest a practical model for future interveners. An intervener might carry out an Incentivisation Session to motivate security improvement, then carry out a Threat Modelling session. This identifies the risks and benefits to the organisation from security issues. This then justifies Product Negotiation, plus use of Configuration Review and Automated Static Analysis, and in some cases the more expensive Code Review and Penetration Testing. Meanwhile, the intervener uses On-the-Job Training to keep the team actively considering software security. Figure 19 on page 87 illustrates this model: the training items (in grey) are provided by the interveners; the process items (in blue) and vulnerability finding items (in amber) are typically driven by the developers.

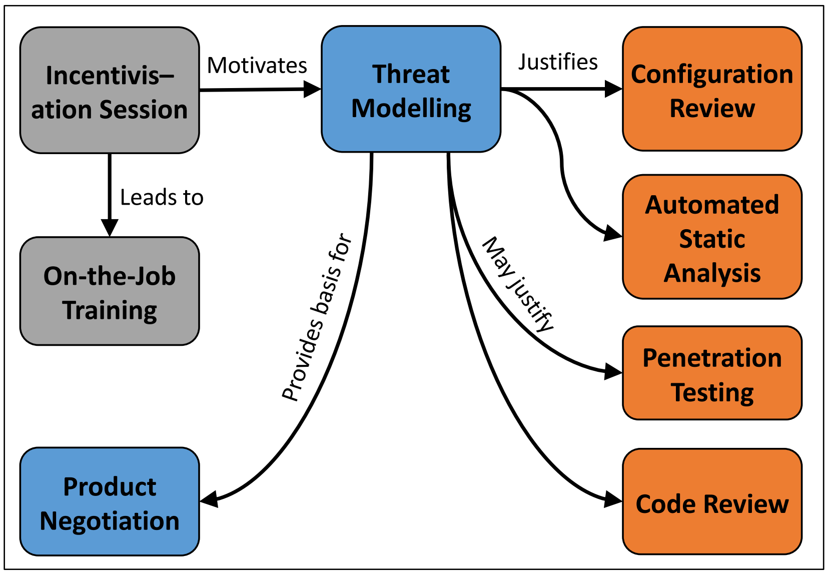

Figure 19: A Practical Approach for Interveners

4.14.2 Next Steps

The findings in this chapter provide a basis for the creation of an intervention package to help developers improve security. Specifically, we deduce that an intervention must:

- Motivate Active Developers to drive their own security improvements;
- Provide an Incentivisation Session to help do so;
- Encourage developers to adopt Threat Assessment, Stakeholder Negotiation, Configuration Review, Source Code Review, Automated Static Analysis and Penetration Testing; and
- Deliver continued On-the-Job Training.

Chapter 6 describes the creation of such a package.

5 Developer Survey

To provide proof of need and a baseline of existing security practice in commercial development teams required an online survey. To do this, the author collaborated with researchers in the Ruhr University of Bochum, Leibnitz University of Hannover and Paderborn University[18].

We chose a specific set of software developers to investigate: Android application developers. Our reasons for choosing these were twofold:

1. The research team has considerable experience in Android development security research [3,126]
2. The Android ecosystem provides access to both developers and the software developed, along with an indication of application usage.

The research question for the survey was the following[19]:

RQ 3 *To what extent, and how, does a perceived need for security and privacy lead to security-enhancing activities and interactions in an Android development team and result in better software security?*

Accordingly, we carried out an online survey of professional Android developers, asking for details of their security practices and interactions. From statistical analysis of the 330 completed and accepted surveys we deduced, with 95% confidence[20], that:

- No more than 22% of the developers of successful, maintained, Android apps have regular access to security professionals;
- Although more than 71% have used at least one of the basic assurance techniques; less than 49% use any regularly; and security updates for apps generally happen less

[18] Specifically, as stated in the Declaration at the start of this thesis, Sascha Fahl instigated and mentored the survey project, suggested Figure 20 and wrote the initial version of Section 5.1.2; Ben Hermann created and ran the application analysis software package and wrote the initial version of Section 5.2.1; Christian Stransky generated the lists of invitation email addresses, obtained the corresponding application binaries and wrote the initial version of Section 5.1.5; Dominik Wermke created the initial Python Jupyter Notebook analysis plus Figure 25, Figure 27, Figure 29, Figure 31, and Figure 35 in Sections 5.3.2 to 5.3.4..

[19] RQ 3 was modified to include 'how' and 'perceived' following feedback on a submitted paper.

[20]Assuming the sample is representative of the wider population. See Section 5.3.1.

than once a year. Among the roughly 40% who work in teams, up to 64% may use at least one assurance technique regularly.
- Less than 15% of them have made more than cosmetic changes as a result of the new GDPR legislation.

We also found that:

- Android app developers' use of assurance techniques is positively correlated with the perceived need for security, the involvement of security experts or champions, and the security expertise of the developers;
- The reported frequency of app security updates is positively correlated with the perceived need for security, the security expertise of the developers, and the developers' use of assurance techniques.

A second phase investigated how these aspects of the development process were reflected in objective app security outcomes. The research question for this phase was:

RQ 3.1: *To what extent do the perceived need for security, the involvement of specialist roles, and the use of assurance techniques in a development team lead to fewer security defects?*

We analysed the Android applications created by each developer and matched the findings to the questionnaire results, concluding that:

- There was no correlation found between the perceived need for app security, nor the use of assurance techniques, and the defect count of the resulting app; and
- Surprisingly, the involvement of security professionals and 'security champions' is correlated with higher cryptographic API defect counts.

This chapter is structured as follows. Section 5.1 describes the survey design, participant recruitment approach, analysis plan, survey trials and limitations; Section 5.2 describes the same for the app binary analysis; Section 5.3 explores both the survey and app analysis results; Section 5.4 explores the implications of these results; and Section 5.5 summarises the main learning points and conclusions.

5.1 Survey Methodology

We conducted an online survey of Google Play Android developers in May 2019, receiving 345 complete responses. Section 3.4 provides a detailed overview of the methodology used. Figure 20 summarises the study procedure, the stages of which will be unpacked in the following sections.

5.1.1 Ethics

We addressed the ethical issues discussed in Section 3.4.1 as follows. All the institutions' Institutional Review Boards approved this study, including the use of publicly available contact details for the survey invitations. With the invitation, we provided all participants with a link to a web page that informed them about the study purpose, the data we collected and stored, and an email address and phone number to contact the principal investigators in case they had questions or concerns.

The use of data from non-participants (the Android applications created by the developers) can also raise ethical issues; the approach in this survey was also approved by the Institutional Review Boards.

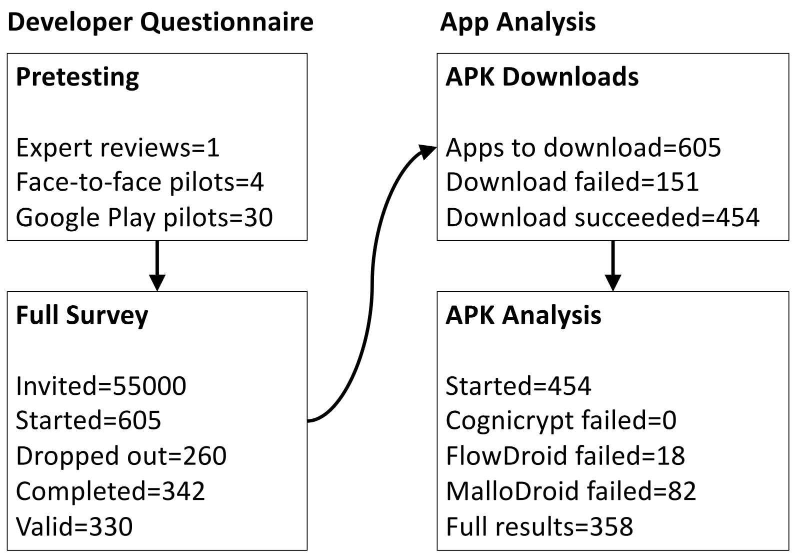

Figure 20: Developer Survey Study Procedure

5.1.2 Survey Questionnaire Structure

We asked our respondents to answer questions about their Android application development behaviour and context relevant for application security and privacy, and a set of demographic questions. Although this might have led to self-reporting bias and social desirability bias, we considered this approach the best practical approach to address the research. We implemented the questionnaire in Qualtrics [138], and developed it using an iterative process.

Appendix I contains the full list of questions. In summary, we asked respondents:

- Whether they worked in a team, and if so their role and the team size;
- The Android development environments they used;
- The number of recent releases for their most frequently updated app, and the proportions of updates addressing new features, addressing library updates, and addressing security or privacy issues;
- Their attitude to security and of privacy, both implicitly and for sales;
- Whether they receive support from security professionals or internal security champions, and if so, the nature of that support;
- What events had led to recent changes in security;
- Which secure development practices they used, and to what extent;
- How long they had been programming, both generally and with Android;
- How many apps they had developed, and whether it was their primary job; and
- Demographic information about gender, language, and country of residence.

Definitions: In the questions, ‘recent’ was defined as the previous two years, and ‘security champion’ to be a non-expert who takes a particular interest in security [26]. We asked developers with more than one app to provide answers for the most frequently updated one.

Secure Development Practices: The questions about secure development practices asked specifically about five of the most frequently-used assurance techniques identified in the Expert Survey (Section 4.4) as follows:

Threat Assessment	Working as a team to identify actors and potential threats; following this up with risk assessment and mitigation decisions.
Configuration Review	Keeping components up to date using component security analysis tools to the toolchain.
Automated Static Analysis	Using code analysis tools to identify certain categories of security vulnerability.
Code Review	Having other programmers or security experts review code for security problems.
Penetration Testing	Having external specialist security testers identify flaws.

Question Wording: All the questions about security processes were worded as questions of fact, rather than of future intentions as in some security surveys [51], to reduce the impact of desirability biases.

Omissions: We considered asking about code analysis tools, since these are of particular interest to researchers. However, static analysis is only one of the five assurance techniques considered, so to be consistent we would need to investigate tools for the other four techniques as well, which would have made the questionnaire unacceptably long without contributing to answers for the research questions.

5.1.3 Survey Pretesting

Section 3.4.5 describes the pretesting done. The specific results from the two pre-tests were as follows.

Face-to-face Testing: From the face-to-face testing, we modified the wording of two questions and added one, to improve clarity. We also noted that responses from those who had produced little-used apps were not interesting from a security viewpoint. Accordingly, we modified our criteria for invitations to only invite developers of 'successful' and 'maintained' apps: ones that had received more than 100 downloads and at least one update.

Pilot Survey: In the pilot surveys, 5000 were invited using the email in Appendix H, producing 30 completed entries. The number of dropouts found in the pilot responses was acceptable; since of those who completed the first page of questions, only 21% dropped out later in the survey. We manually coded the changes respondents had made as a result of GDPR, and provided the most frequent answers as 'tick boxes' in the final survey.

In addition, the pilot survey identified the following additional research questions to help scope the problem of supporting developers:

RQ 3.2 *What proportion of Android developers have access to security experts, and*

RQ 3.3 *To what extent do Android developers use assurance techniques?*

Given that RQ 1 relates specifically to UK development teams, we specifically want also to investigate the figures for developers in teams and for UK developers.

5.1.4 Required Sample Size

Using Fowler's method described in Section 3.4.6, we chose the sample size to get between 50 and 100 in each group, which would give typical sampling errors on data based on each subgroup between 4% and 15%: a sample size of 310. From the pilot survey response rate, we calculated that this required us to send 55,000 invitations.

5.1.5 Recruitment

Only registered Google Play developers were invited. From January to February 2019 the team crawled the details' pages of 3,608,673 (2,087,829 free and 1,520,844 paid) Android applications from those published in Google Play. For all apps, we stored their last update time, name, developer data and download counts.

Overall, we identified 312,369 developer accounts that match the 100+ downloads and update requirements in Google Play. The number of apps published by a single developer account in that sample ranged from 1 to 3,302 with a median of 2. From these 312,369 developer accounts, we selected a random sample of 55,000, and the author used Qualtrics to send a single invitation email to each to ask each to kindly to support the research (Appendix H). Of the invited 55,000 participants, 605 started and 345 completed the survey. Ten of the invited developers reached out via email. None complained about being contacted; three asked to be removed from the mailing list; the remainder provided various reasons for not completing the survey, including two who noted the security questions and stated that their apps had no security aspects. 240 took the opportunity to leave their email address in the survey questionnaire for us to send them the results of this work.

5.1.6 Filtering Invalid Results

In psychological surveys, a common stratagem is to ask a question twice, once negated. One can then filter out meaningless responses (or use them to calculate a "self-consistency" score for the survey). Since the survey was asking facts rather than attitudes, we concluded that this would be contrived and irritating to the respondents. Instead the author looked at response times, experimented to find a minimum time that a participant might be expected to take to complete the survey: 3 minutes. We then filtered out the few (10) surveys that had taken less than that minimum time to complete.

5.1.7 Survey Statistical Analysis Plan

Four forms of statistical analysis were used:

1. Population analysis, to explore how well our sample corresponds to the larger population;
2. Graphical analysis, to show the nature of the data;
3. Confidence limits for proportions in the wider population based on proportions in the sample; and
4. Correlation analysis, to identify relationships between different data items.

The statistics scores and outline analysis methods were defined before the main survey data collection, as required for research best practice (see Section 3.4.4). The analysis used Python statistical packages, including Pandas, Statsmodels, and Seaborn, in Jupyter Notebooks [97].

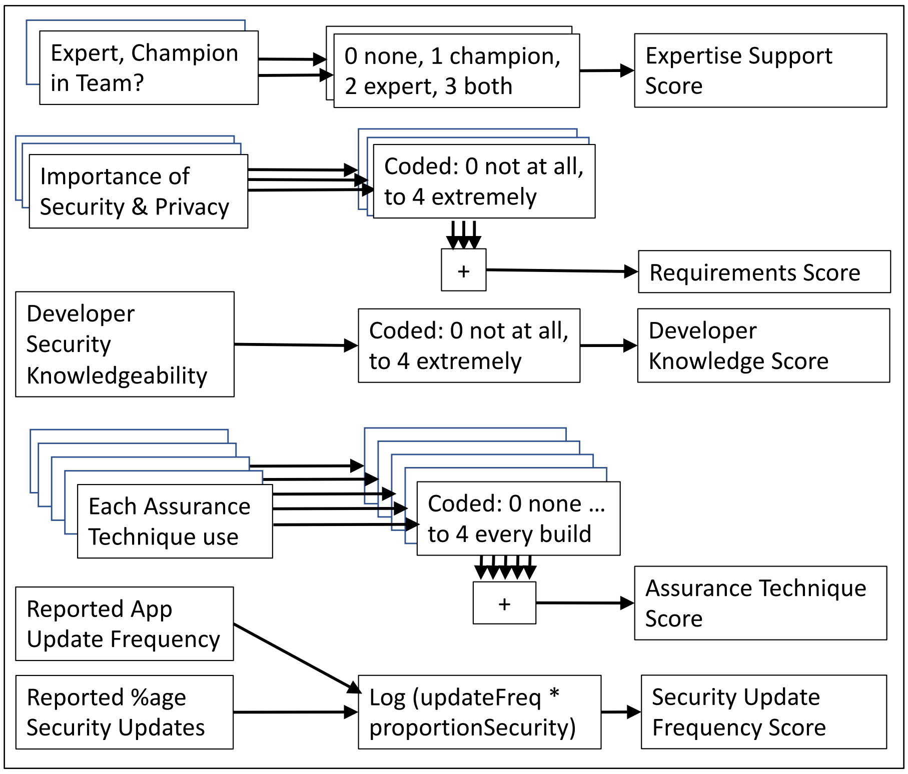
Expert, Champion in Team?
0 none, 1 champion, 2 expert, 3 both
Expertise Support Score
Importance of Security & Privacy
Coded: 0 not at all, to 4 extremely
+
Requirements Score
Developer Security Knowledgeability
Coded: 0 not at all, to 4 extremely
Developer Knowledge Score
Each Assurance Technique use
Coded: 0 none … to 4 every build
+
Assurance Technique Score
Reported App Update Frequency
Reported %age Security Updates
Log (updateFreq * proportionSecurity)
Security Update Frequency Score

Figure 21: Developer Survey Security Scores

Linear Analysis for RQ 3: Addressing **RQ 3** required scores based on each respondent's survey answers: some scores captured the "*need for security and privacy*" (the independent, input, variables); others the "*security-enhancing activities and interactions in the development team*" (the dependent, output, variables).

Figure 21 shows the processing to create these scores. The aim in each case was to create an ordinal score that approximated to linear across the range of raw data, so a higher score corresponds to more security (or more drivers towards security) and each increment represents a similar semantic increase. As shown, the Requirements Score reflects the *security need* as the arithmetic sum of the three Likert-style responses encoded as integers; similarly, to explore the *why*, there are Developer Knowledge and Expertise Support scores. The Security Update Frequency estimate was the product of the answers to two questions; this had an exponential (Poisson) distribution, so to make it linear [10] we used a transformation: $\log(x_i + 1)$ to create the Security Update Frequency Score. See Appendix J for details.

The calculation of the Expertise Support Score is based on an assumption that direct expert involvement is more effective than 'security champions'; the Requirements Score assumes that, for example, occasionally using two techniques is as effective as regularly using one; and the Assurance Technique Score assumes that, say, considering four techniques is as effective as consistently using one. Though reasonable as an approach, none of these scores are linear or even provably ordinal [161]; we anticipated that inconsistencies in the scoring would add to the statistical variance but not obscure overall trends. See Section 5.3.6 for a post-hoc justification.

In statistics, the usual relationship to look for is a linear one. In line with previous research in the field [51] we used the Pearson Correlation Coefficient ('Pearson R') calculation [44] to establish whether pairs of values had a significant linear relationship; this test is acceptable for Likert-style data [96,124].

In this analysis we treated the Security Update Frequency score as a dependent variable (output); and the Requirements, Expertise Support, and Developer Knowledge scores as independent variables (inputs)[21]. The use of Assurance Techniques is likely to be affected by the latter three variables but may itself in turn affect the Security Update Frequency and other security outcomes; in the analysis, therefore, we treated the Assurance Technique score as an independent and as a dependent variable in different tests.

As discussed in Section 3.4.4, since the analysis constituted multiple tests on the same data, we applied the Bonferroni correction [144], reducing the threshold for 'significance' accordingly to $(5\%)/5 = 1\%$. To validate the preconditions for the Pearson Correlation Coefficient test [44], we then constructed x-y plots of all the pairs of variables that showed significant correlation.

5.2 Application Analysis Methodology

In the second phase of the project, we downloaded and analysed the apps corresponding to the survey responses. For analysis, we used a selection of state-of-the-art vulnerability scanners. Each one focuses on a different problem category and produces a relatively low number of false positives. We chose mature tools that are openly accessible to Android developers.

5.2.1 Description of Analysis Tools

The tools covered three key areas: SSL Security, Cryptographic API Misuse, and Privacy Leaks. We selected these areas because these cover a representative range from the possible security and privacy vulnerabilities faced by application developers [128].

SSL Security: A key concern in the secure treatment of information is the correct use of secure transport mechanisms (SSL, TLS) when connecting to remote systems. To capture this aspect, we used two techniques. First, we used MalloDroid [56] to inspect the correct use of certificate validation in the apps code. Second, we extracted any HTTPS URLs from the constant pools of the classes contained in the app using the `OPAL` framework [52] and checked the corresponding server configurations and certificates using the command-line tools `curl` and `openssl`.

Cryptographic API Misuse: Many apps use cryptographic measures to improve data security and privacy, and a key concern in the secure treatment of information is the handling of cryptographic primitives (e.g., for persistence). We run CogniCrypt [100] to capture this aspect. CogniCrypt uses static inter-procedural static program analysis to detect misuses of the Java Cryptography API. The detected problems range from improper configuration of algorithms (e.g., use of AES with ECB) to incorrect order of calls to the API. As it is formulated as a static program analysis, CogniCrypt makes conservative assumptions (over-approximations) on the control flow of the program, which may produce false positive reports.

Privacy Leaks: To find possibly harmful data flow that can lead to privacy leaks, we used FlowDroid [11]. This tool is designed to find information flow in Android apps between defined information sources and information sinks. For example, the location APIs are considered as sources of private information, and the text message sending APIs as sinks. FlowDroid uses static inter-procedural data flow analysis to find evidence of

[21]Pearson's R does not distinguish dependent and independent variables, so this affects only our choice of scores to correlate with each other.

Table 9: App Binary Analysis Tool Versions

MalloDroid	Version Dec 30, 2013
OPAL framework	Version 1.0.0
curl	Version 7.64.0
openssl	Version 1.1.1b
FlowDroid	Version 2.7.1
LibScout	Version 2.3.2
CogniCrypt	Version 1.0.0

directed information flow between these methods. We configured the tool with the default sources and sink for Android provided by the tool authors, which had been constructed by manual inspection of common vulnerabilities in Android apps. FlowDroid is not able to determine if the found information flow is to be considered an actual leak as it might also be intended to use the information in the particular context (e.g. for location-based services).

Practical Approach: We downloaded the application binaries for at least one application by each of the survey respondents, wherever possible; we ran the full set of scanning tools on each and counted the *issues* (reports of possible vulnerabilities) generated. Table 9 lists the versions of the tools we used. In some cases, the tools failed (see Figure 20); where this happened the corresponding app and developer data were omitted from the analysis.

5.2.2 Application Statistical Analysis

As in the survey statistical analysis (Section 5.1.7), we used graphical tools to explore the data, and linear analysis to explore relationships between the data.

To investigate RQ 3.1, we defined further scores to represent the outcome *"fewer security defects"* in each app analysed. Figure 22 shows the processing involved. We anticipated that the issue counts would have a Poisson distribution; to permit linear analysis we used a log transformation[22]. As with the scores for developer behaviour, we wanted scores that increase with increasing app security and privacy, and we therefore negated the log value.

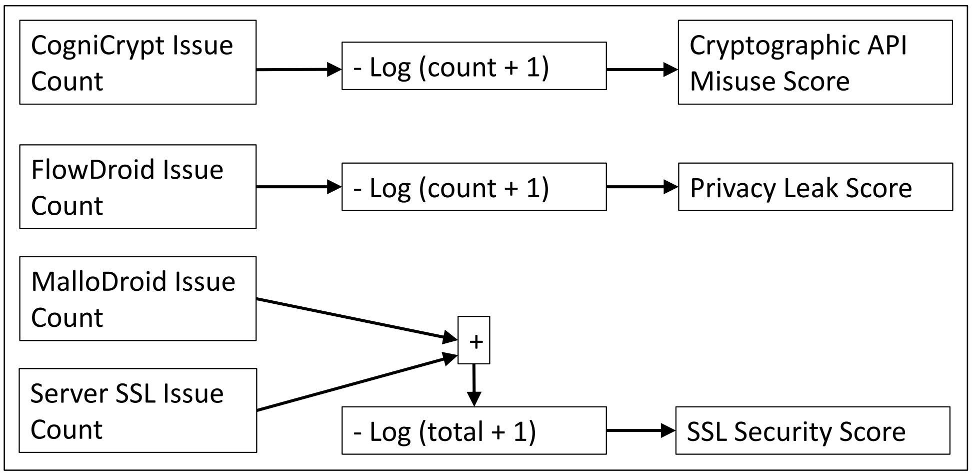

Figure 22: App Analysis Security Scores

[22] Specifically, $\log(x_i + \mathrm{k})$, where k is chosen to minimize skewness [10]; in practice we trialled different values of k, finding no difference to the results, so used the conventional research practice of k=1.

We used the same method as previously (Section 5.1.7) to look for relationship between these scores and the scores from Figure 21 covering the "*need for security, involvement of specialist roles, and use of assurance techniques in a development team"* in RQ 3.1.

5.2.3 Survey Limitations

As with most studies of this type, our work has limitations.

The response rate for our online developer survey was low, as might be expected from sending unsolicited emails to prospective participants. However, our recruitment approach was also used by relevant previous work [2,3,195]. The low response rate may show some self-selection bias, but since the invitations made no mention of security, we have no reason to believe a priori that those who responded differ meaningfully in terms of security or privacy behaviour from those who did not.

All the survey data—except download count and last app update date—is self-reported. Though we addressed this by keeping questions as fact oriented as possible, this is an important limitation.

In terms of the population, the survey reached app owners rather than all app developers; so, data about the respondents' own experience is not representative of all Android developers, nor of software developers in general.

5.2.4 App Analysis Limitations

The static analysis tools we chose each consider specific categories of vulnerabilities. This may disregard other categories of issues which may also be security critical. Indeed, many vulnerabilities—especially privacy ones—will tend to be in the intended app functionality rather than in the detailed implementation, and we have no way to estimate these. However, we used detectors for a range of implementation issues which may be found through other methods, and which developers who consider security or privacy important would be expected to address.

Static program analysis tools often report false positives, and the tools we used are no exception. Our approach for this survey, however, was to assume that the reported issue counts will correlate with the numbers of true vulnerabilities, and therefore that such counts can be used as a proxy for aspects of app security in statistical analysis.

We were able only to analyse 'free' and 'freemium' apps, not ones where Google Play Store charges for download; this may introduce a bias. In cases where respondents have more than one app, the app we downloaded may not be one requiring the security practices and priorities described in the survey.

We considered improving the app analysis by ranking vulnerabilities based on severity. However, the analysis did not identify vulnerabilities; it reported counts of 'issues' detected, where an 'issue' is a potential vulnerability. To determine whether an issue represents a vulnerability would require detailed analysis of the source code; this source code was not available to the researchers, and decompilation was infeasible due to the widespread use of obfuscation tools.

We also considered distinguishing issues in the source code from issues in libraries, or using vulnerability ratings for libraries. However, although there have been several worthwhile tools developed to analyse the libraries used by Android apps, including LibScout [16] and LibDetect [72], with the current state of the art they are not sophisticated enough to detect library versions reliably, nor are they integrated with other

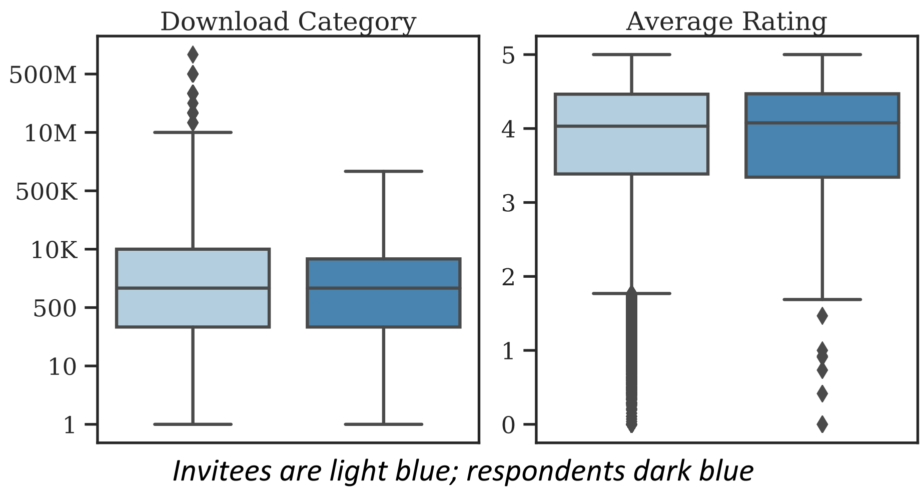

Figure 23: Comparing Participants' App Success with Invitees'

binary analysis tools to allow differentiation of issues in libraries from issues in the main code.

5.3 Results

This section describes our results, both from the survey and from the app analysis.

5.3.1 Sample Validity

Comparing the box plots for invitees with those for participants in Figure 23, we see that the average user rating and number of downloads for apps produced by the 345 developers who completed surveys are very similar to those for the 55,000 invited.

One survey question asked the respondent's years of experience in software development. Figure 24 compares the results with answers to a similar question addressed to the 21,000 Android developers out of the 89,000 developers who answered the 2019 Stack Overflow developer survey [158]. As will be seen, our respondents are generally more experienced than the corresponding general population (our median 12 years; Stack Overflow population median of 8 years; Mann Whitney $p = 10^{-21}$).

One concern was whether our app selection criterion (over 100 downloads and one update) was too lenient, since little-used apps may well have poor security. To test this,

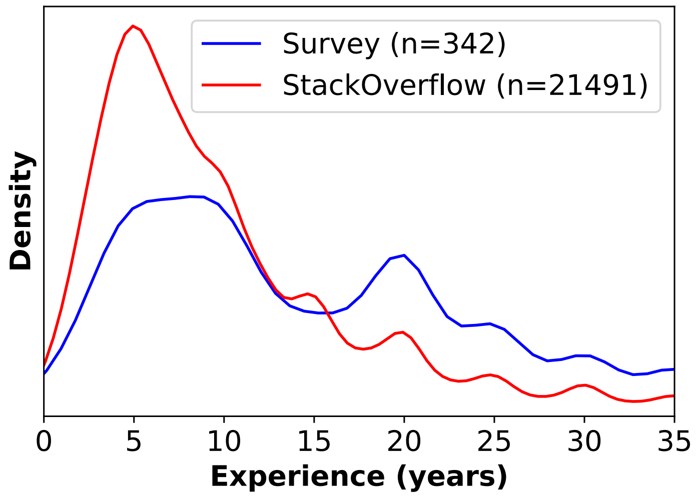

Figure 24: Participants' Experience Compared with Developer Population

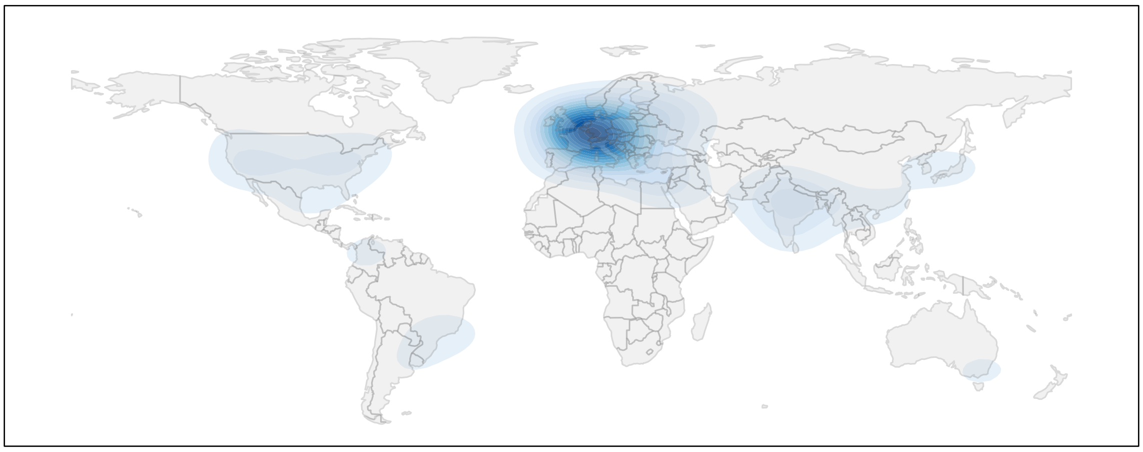

Figure 25: Geographical Location of Participants

we used the Mann Whitney test comparing developers of apps with less than 1000 downloads against the rest[23] (see Section 3.4.4). We did this for all of the scores (Sections 5.1.7 and 5.2.2) and for all the numerically analysable survey questions to see if the distribution was different for low-download apps. In the survey results and scores we found small p-values ($p < 0.003$) only for questions whose answers we expected to correlate with download counts: 'How many apps have you developed', 'How many Android apps have your developed' and 'Is developing apps your primary job', and we concluded that the populations were essentially the same. Doing the same Mann Whitney test on the App Analysis scores, we found low p-values only for the Cryptographic API Misuse and Privacy Leak scores ($p \sim 0.016$ for each). Though suggestive, these values are not significant after statistical correction. We concluded that there was no justification for changing our app selection criteria.

Finally, to check the accuracy of respondents' replies, we compared the respondent-stated app update interval with objective evidence. App update histories are not generally available from Google Play, but we did collect the last update date for each app we considered. We correlated the time since that last update with the participant-stated update interval using log scales: Pearson R=0.38, P=1e-9 (n=242). The tiny P value corroborates the assumption that the stated update frequencies reflect reality; the moderate R value reflects that respondents were asked the about updates to 'their most frequently updated app' and not the app we considered, plus the randomness of where each app was in the release cycle.

5.3.2 Geographical Location of Participants

Figure 25 provides an overview of the physical location of the participants. As highlighted, they are predominantly European.

Figure 26 shows the main countries involved. As will be seen, this top eight countries accounts for less than half the total participants; a total of 65 countries were represented.

[23] We specified this analysis after data gathering; accordingly, significance in any of the correlations should be considered suspect. However, a lack of significance in a wide range of correlation calculations is a valid finding.

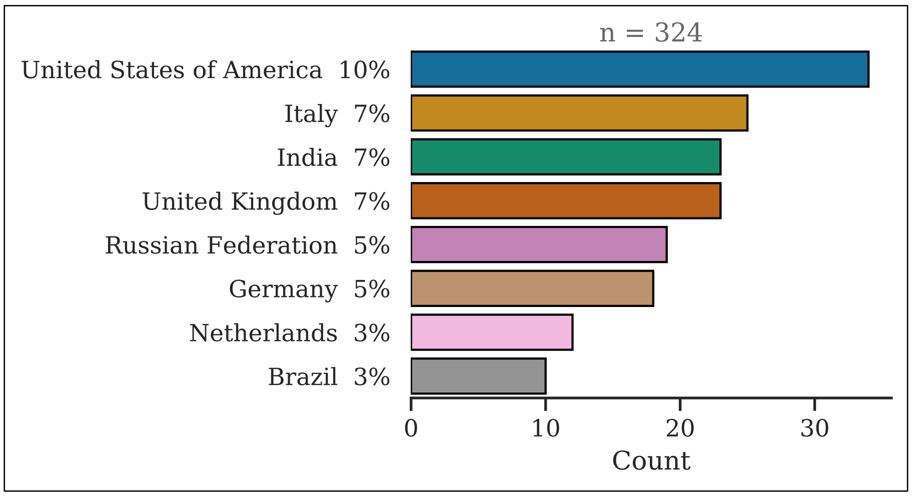

Figure 26: Countries of Participants

5.3.3 Findings on Self-Reported Developer Behaviour

The next sections describe the survey results for individual survey questions, without considering associations between answers[24].

Importance of Security and Privacy: Figure 27 shows respondents' ratings of the importance of security and privacy in their apps. For comparison, we also asked and show the importance of other functional and non-functional requirements. We were surprised how many developers considered security and privacy important. Over 40% of respondents considered each of security and privacy to be 'extremely important': ratings comparable with multi-platform support and higher than support for many features.

Team Structure: Only 42% of respondents were working in teams (95% confidence interval 36% – 46%), the remainder being solo developers. Of those working in teams, Figure 28 shows the distribution of team sizes; more than half had 4 or fewer members.

Figure 29 shows how many teams included particular roles (other than the respondent). Notably only half had tester roles, and only a third project managers.

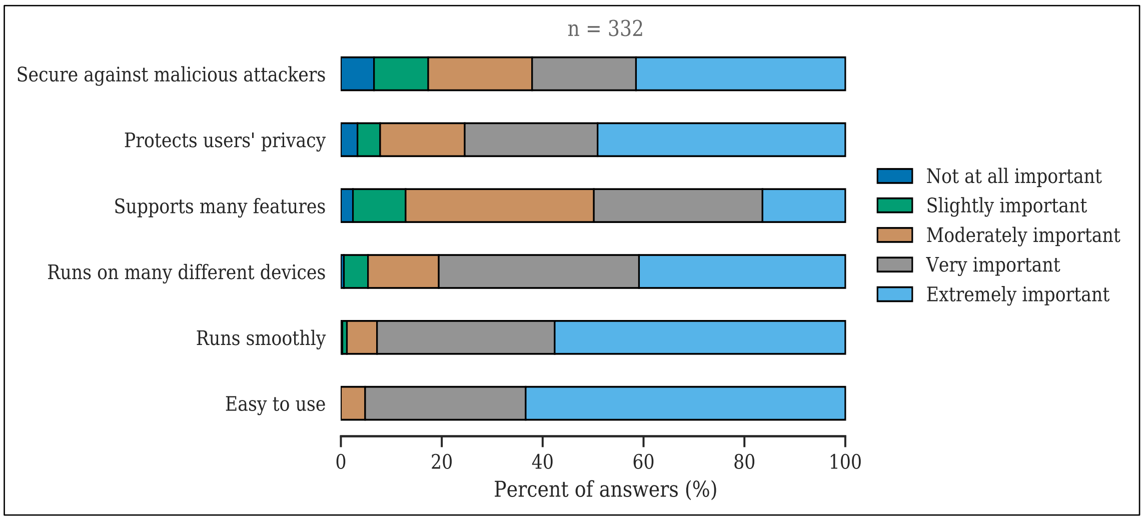

Figure 27: Importance of Different Requirements

[24] The number of answers varies to each question or set of questions, giving different values for 'n' in each chart.

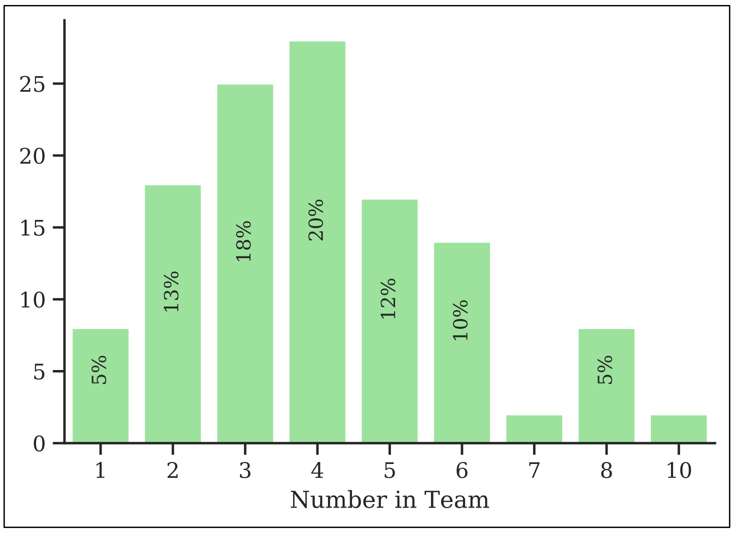

Figure 28: Distribution of Participants' Team Sizes

Security Expert Support: Only 17% of respondents reported receiving support from professional security experts. So, for **RQ 3.2** we calculate the ninety-five percent confidence interval for the proportion working with security experts in the Android app developer population as a whole as:

Lower bound = 14%, Upper bound = 22%

Of these few professional security experts discussed by respondents, 33% were part of the development team and the remainder external. Their most common function was Penetration Testing (44%), but they also provided Design Reviews (39%), Audits (33%) and Training (27%).

Some teams (18%) had a 'security champion', a non-expert providing security input to the rest of the team. Only 7% had both professional experts and champions.

Developer Security Knowledge: Figure 30 shows how survey participants rated their security expertise. Interestingly, very few considered themselves to have no knowledge; this is as we would expect given the level of development experience of participants (Section 5.3.1).

Use of Assurance Techniques: Figure 31 shows the reported use of assurance techniques. Unsurprisingly, Threat Assessment for every build is rare; possibly those respondents consider the list of threats every day. Penetration Testing for each build is

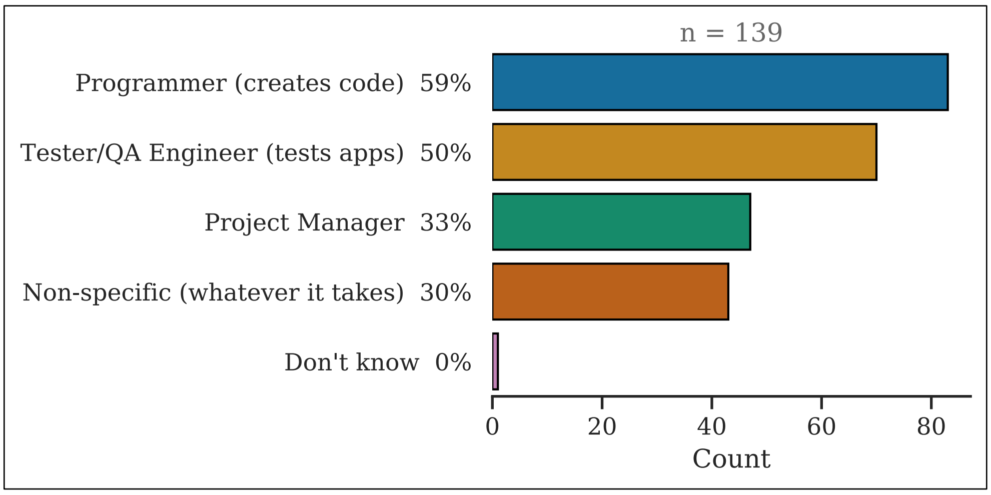

Figure 29: Other Roles in Participants' Teams

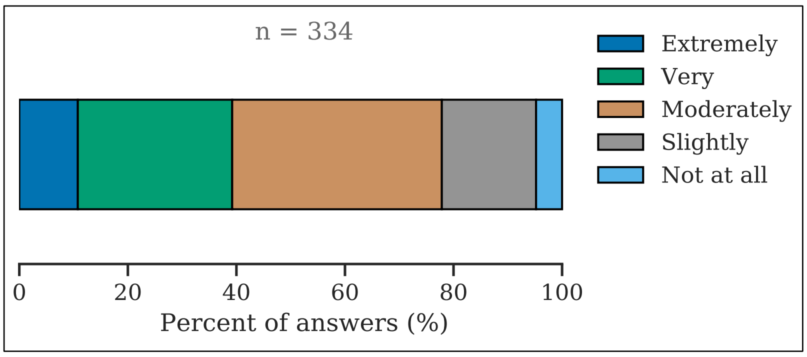

Figure 30: How Knowledgeable about Security

also rare; possibly they meant automated penetration testing. But otherwise the proportions using each are fairly consistent across all the techniques, with just under half not having considered each technique, and only a small percentage using it for every build.

Combinations of Assurance Techniques: We investigated the extent to which teams used combinations of assurance techniques. Figure 32 summarises how many and how often the techniques are used. It shows the cumulative proportion of respondents using each number of techniques, separated out to show how often they used them. Thus for example, the middle bottom dark blue rectangle shows that 4% used all five techniques every release or more often; the middle column shows that nearly 45% used at least one technique every release or more often; the left hand column shows that 76% had trialled at least one technique but only 17% had used all five.

So, for **RQ 3.3**, the 95% confidence intervals for the proportion regularly using one or more of the given assurance techniques in the wider Android developer population [99] are:

Lower bound = 38%, Upper bound = 49%

The figure for those who have at least tried one assurance technique is much higher:

Lower bound = 71%, Upper bound = 80%

We analysed which combinations of techniques were popular amongst the 15% (50) of respondents who only used two or three regularly.

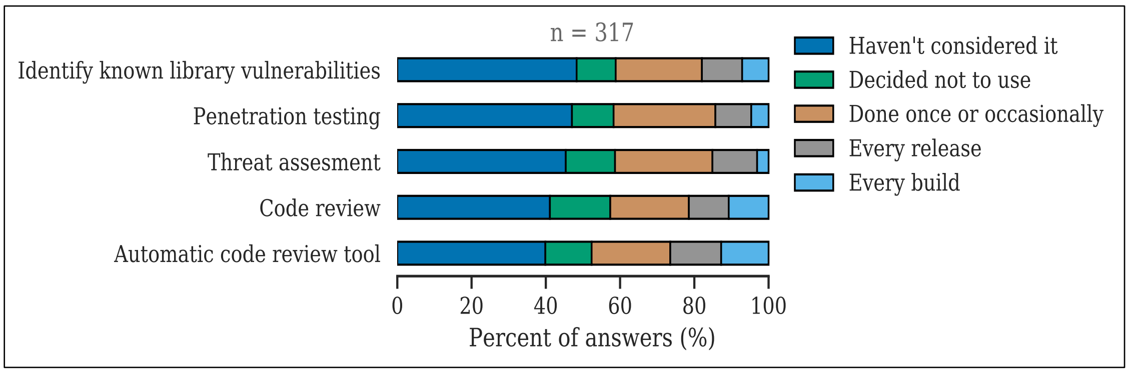

Figure 31: Use of Assurance Techniques

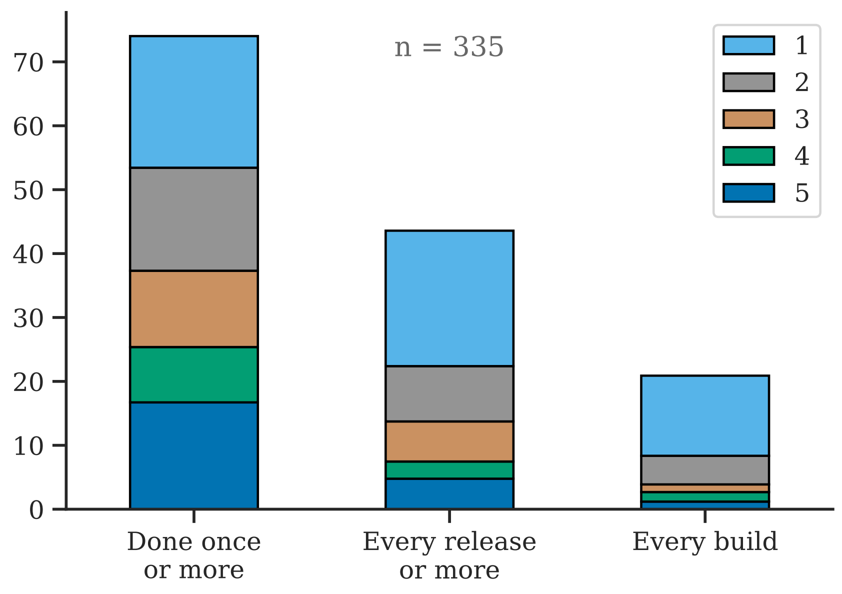

Figure 32: Percentage Using Each Number of Assurance Techniques

The most popular were as follows:

Combination:		**Proportion**
Automatic Static Analysis	Configuration Review	38%
Automatic Static Analysis	Code Review	32%
Code Review	Configuration Review	22%
Threat Assessment	Code Review	16%

Security Updates: Figure 33 shows the frequency of security updates, calculated as the product of the reported update frequency, and the reported proportion of security updates. The 95% confidence interval for the proportion with less than one update a year is $59\% - 70\%$.

5.3.4 Recent Changes in Team or Development Security

Given how fast moving the field of software security has become, it is also important to know what might have caused changes in the developers' perceptions or actions around security. Two questions in the survey addressed this: one listing possible reasons for security and privacy improvements and asking the user to select all that had affected app security; and for those who mentioned an impact from the recent European GDPR legislation [55], a further question asking what changes they had made as a result. Since the GDPR legislation affects any apps collecting data in Europe, it impacts developers worldwide.

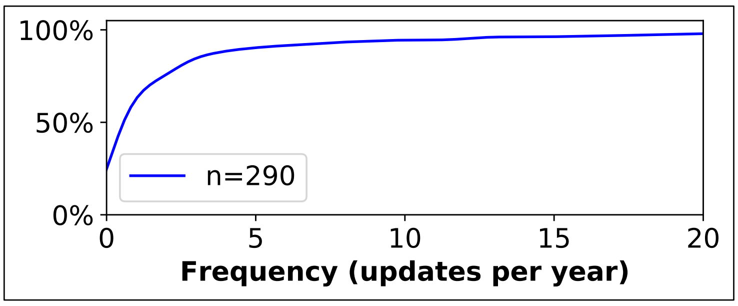

Figure 33: Cumulative Security Update Frequency

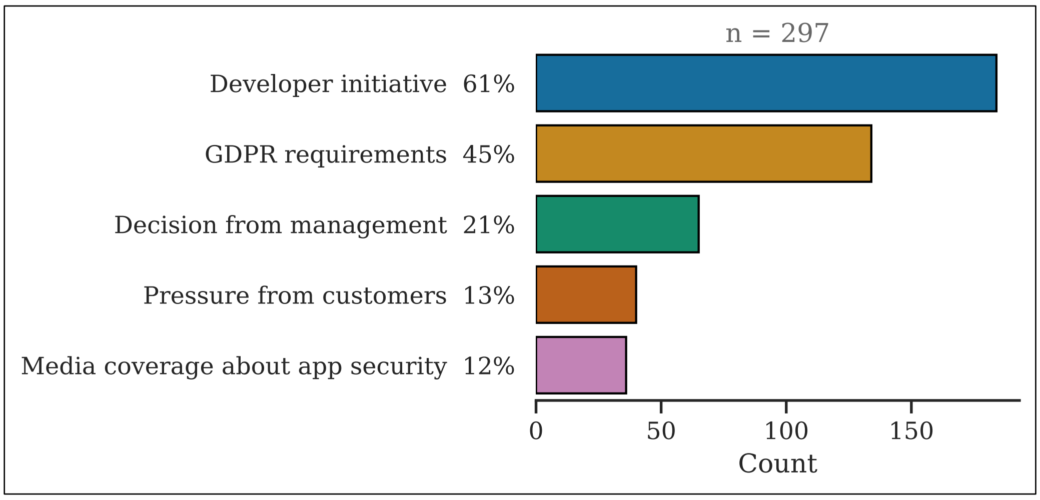

Figure 34: Top Five Reasons for Security Changes

Figure 34 shows the answers. Interestingly, the developers' perception is that, even more than GDPR, the main security driver has been the developers themselves. Encouragingly very few (3%) reported security improvements as a consequence of actual security issues affecting themselves, suggesting that this is still rare; a few more (7%) reported 'horror stories'—something bad happening to a competitor.

Of the 45% of participants (n=133) who reported changes as a result of GDPR, Figure 35 summarises the changes they made as a result. We observe that the majority of these changes were cosmetic as far as solving or mitigating security problems was concerned: changing privacy policies or adding pop-up dialogs. Only 33 made substantive changes to improve user security or privacy (giving 95% confidence limits of 8% to 15% for the wider Android developer population [99]).

5.3.5 Team-Based and UK-Based Assurance Technique Use

Since this thesis is primarily concerned with development teams, it is of value to see what we can deduce for this group of developers.

Figure 36 shows the use of assurance techniques by 139 respondents who described themselves as working in teams. Unsurprisingly, comparing this with Figure 31 we see that in teams a decidedly larger proportion are doing Code Review.

Specifically, answering **RQ 3.3** for team-based Android developers, 56% reported using one or more assurance techniques regularly, making the 95% confidence intervals for the proportion in the wider population:

Lower bound = 48%, Upper bound = 64%

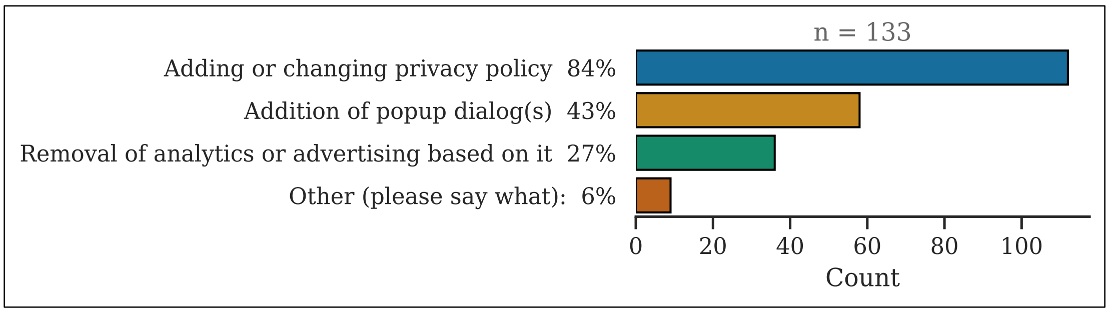

Figure 35: Changes Made Due to GDPR

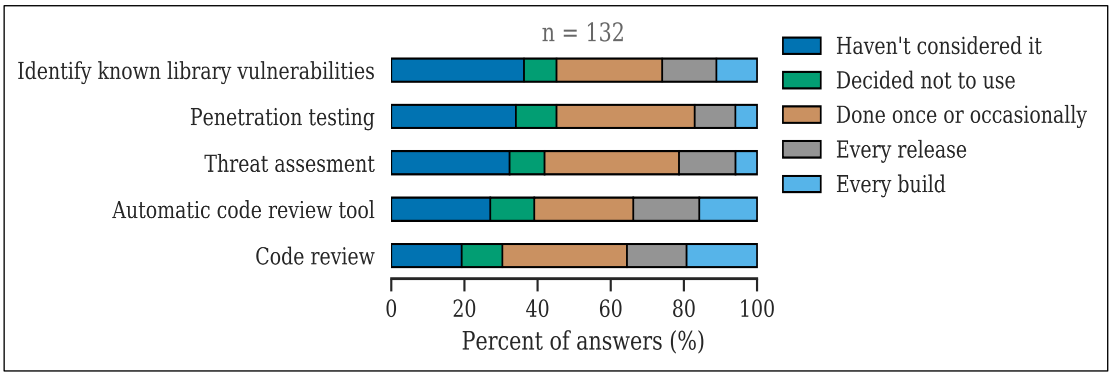

Figure 36: Use of Assurance Techniques by Developers in Teams

Turning to the figures for the United Kingdom, Figure 26 on page 99 showed the countries of the respondents. Unfortunately, the sample size from the United Kingdom is too small (n=22) for meaningful statistical analysis[25].

Instead, we observed that there was no a priori reason to believe United Kingdom developers are different from those in the rest of the world in development behaviour. We validated this using the Mann Whitney test to compare the UK sample with the full sample population (as Section 5.3.1). We found significant differences only in Q25, "For how many years have you been programming in general (not just for Android)", which we ascribe to the difference in developer populations.

Accordingly, we have no reason to believe that the results relating to developer *behaviour* would have been materially different in a UK-only survey.

5.3.6 Linear Analysis of Developer Survey Scores

Table 10 shows the results of the analysis described in Section 5.1.7. It correlates each of the two dependent scores representing "security-enhancing activities and interactions in the development team" against four independent "need and mechanisms for security and privacy" scores. Each cell in the table shows, for the corresponding test:

The R-Value, between -1 and 1, indicating how much of the variation in one measurement corresponds to the variation in the other.

The P-Value, indicating the likelihood that the result could have been observed by chance.

Table 10: Pearson R Results for Developer Survey Security Scores

Independent: / Dependent:	Expertise Support	Require-ments	Developer Knowledge	Assurance Technique Use
Assurance Technique Use	0.56, 3.9e-25	0.37, 1.5e-11	0.27, 8.6e-07	
Security Update Frequency	0.16, 0.0085	0.25, 2e-05	*0.03, 0.61*	0.41, 5.7e-13

[25] It would have been possible to select just '.uk' email addresses for the invitations; however, this would both have introduced bias (since many UK email addresses do not conform to this pattern), and lacked interest to the non-UK co-researchers.

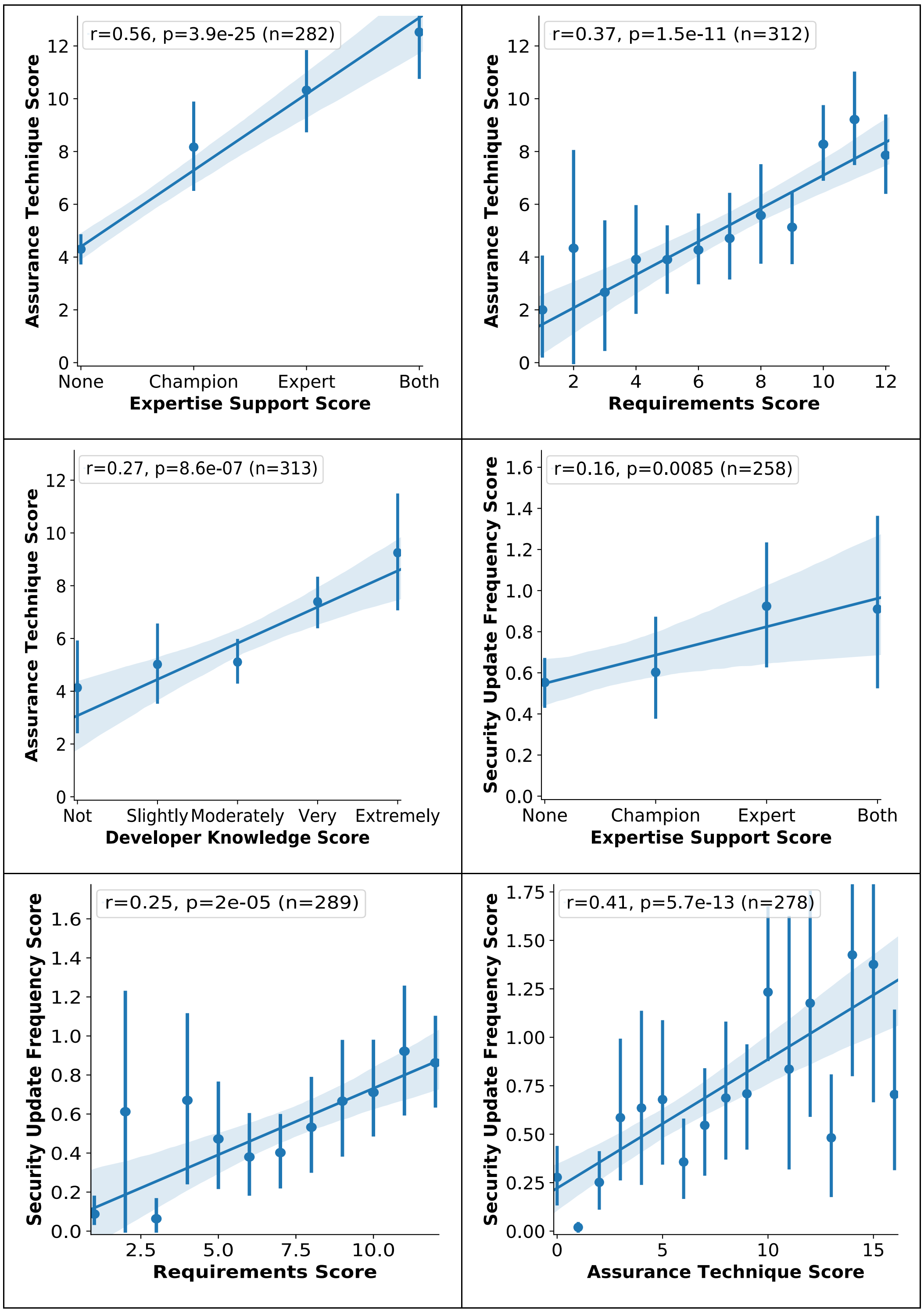

Figure 37: Cross-plots of the Scores with Significant Correlations

Non-italic figures highlighted in yellow indicate a statistically significant result ($p < 0.01$): results where we can be reasonably sure that higher values in one score are associated with higher values in the other.

Figure 37 shows x-y plots of these significant results. Dots and vertical bars show the mean and its 95% confidence interval for the y-readings corresponding to each x-value. The plots also show a simple linear regression line and its confidence limits. The graphs validate the preconditions for the use of Pearson R [130]: particularly homoscedascity and lack of outliers.

Table 11: Pearson R Results for App Security vs. Developer Security

Independent: / Dependent:	Expertise Support	Require-ments	Developer Knowledge	Assurance Technique Use
Cryptographic API Misuse	-0.17, 0.016	*-0.06, 0.37*	*-0.09, 0.17*	*-0.13, 0.047*
Privacy Leak	*-0.09, 0.20*	*-0.01, 0.85*	*0.02, 0.81*	*0.02, 0.81*
SSL Security	*-0.14, 0.049*	*0.01, 0.93*	*-0.02, 0.76*	*-0.08, 0.20*

We observe that the first two plots also justify our choice of the calculation for the Requirements Score and Expertise Support Score since the use of assurance techniques shows a strong linear relationship to both scores.

5.3.7 Findings on Application Security

In the Application Security analysis (see Section 5.2.2), of the tools used, CogniCrypt reported no issues for 32% of apps; FlowDroid for 35% and the Bad SSL/MalloDroid combination for 70%. Only 20% of apps analysed showed no issues from any of the tools.

Table 11 shows the results of the analysis described in Section 5.2.2. It correlates each of three dependent scores representing "fewer security defects" against the four independent "need and mechanisms for security and privacy" scores. Non-italic figures highlighted in yellow indicate a statistically significant result ($p < 0.01$).

Only one result achieves significance and bizarrely that result suggests a negative correlation: the involvement of security professionals and champions is associated with worse Cryptographic API misuse outcomes.

Figure 38 explores this odd finding. It shows that the effect is not large, and that both experts and champions seem to be associated with the negative correlation, though experts more so. We note, as well, that the p-value is only just significant given the Bonferroni correction (Threshold for significance $0.05/3 = 0.017$).

Disappointingly, in response to **RQ 3.1**, use of assurance techniques was not associated with better security outcomes, nor was developer security knowledge, nor was a user requirement for good security.

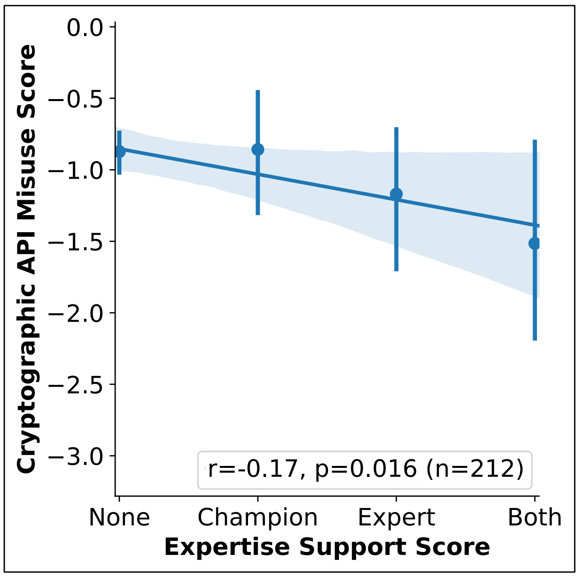

Figure 38: Worse Cryptosecurity with Expert Involvement?

5.4 Discussion

At first sight, the findings in Sections 5.3.3 and 5.3.7 give a depressing view of app security. From Section 5.3.7 we see that over 80% of apps had reported defects from our analysis tools. From Figure 33 we see that the majority of apps get security updates less than once a year. From the analysis of the app security measurements, Table 11 shows that security outcomes seem to have little correlation with an app's perceived need for security and privacy.

And Figure 35 shows that GDPR's new compliance rules for apps have had little positive impact. Certainly, in many cases cosmetic changes may have been all that was needed; but the finding suggests that GDPR has not been a strong force to improve app security and privacy.

5.4.1 Adoption of Security Techniques by Developers

However, there are positive aspects too. Considering the findings in Section 5.3.2, Figure 30 shows us that the vast majority of the respondents consider themselves to have at least some security knowledge, and thus are likely to be aware of security as a possible issue in their software development. Indeed, Figure 27 shows that more than 60% of the respondents consider security to be very or extremely important to their users, and even more put the same value on privacy.

Section 5.3.2's combinations of assurance techniques used are particularly interesting in suggesting how security improvement is happening. Though the analysis only covers a small fraction of the survey participants, those respondents it considers are the ones using only a proportion of the Assurance Techniques and it therefore offers an insight into which techniques are adopted first. One would expect teams whose security is driven by external experts to adopt the Threat Assessment/Penetration Testing combination, since both of these activities can be carried out by the experts themselves; actually, rather more teams adopt tool-only techniques (Automated Static Analysis and Configuration Review), or code-review based techniques (Automated Static Analysis and Code Review), perhaps because few have access to security experts (Section 5.3.2).

This suggests that the adoption of assurance techniques is being driven by the developers themselves, rather than by external security experts, and so what we are seeing is developer-led security. This tallies with the reasons given for app security changes in Figure 34, where the most common reason for changes was developer initiative. It also corresponds to the views of security experts, who emphasise the importance of developer initiative in improving software security [193].

5.4.2 Appropriate Use of Security Techniques

Using security assurance techniques usually has a cost, both in time and in financial terms [164], and therefore it is poor economics to adopt them in cases where they are not required. From Section 5.3.6 we see that this is correctly reflected in the Android ecosystem: the use of Assurance Techniques increases in line with the importance of security for the app. We suggest that the correlation with the involvement of security professionals/champions and with developer knowledge of security may be an effect (expert developers and security professionals will tend to work on products that need security) as much as a cause (their involvement causes increased assurance technique use).

Updating apps also has a considerable cost, and again we would anticipate having more security updates in cases where security is important for the app. Again Table 10 confirms this behaviour, and shows that, justifiably, there is no correlation between the security update frequency and the security experience of the developer.

5.4.3 Impact on App Security

It was disappointing that the use of assurance techniques did not appear to be a major factor leading to better security outcomes when we analysed the apps themselves. Even though the analysis tools can only detect a limited range of code level security issues, we expected more security-experienced developers and those using assurance techniques—especially Static Code Analysis—to generate fewer such issues.

We conclude that other factors must drown out this effect. We observe, for example, that most app binary code will consist of libraries, and even up-to-date libraries will differ enormously in the number of such issues they may have. We hypothesise that the scores generated by the tools we used depend more on the nature of the libraries needed to implement the app functionality than on any attributes of the non-library code created by the developers; current tools cannot verify this effect (Section 5.2.4).

More surprising is the finding that the involvement of professionals and champions seems to be associated with increased numbers of Cryptographic API issues. It seems unlikely that this is because they create the issues. Instead, we observe that our tools will not detect a failure to use cryptography in apps where it is required, whereas experts or champions will do so. We suggest that teams involving experts or champions will therefore tend to use cryptography more frequently, leading to more such issues[26].

5.5 Summary and Conclusions

This chapter describes the creation and deployment of a survey to Android app developers, in which we asked them a range of questions related to their approach to security and privacy in app development; and a second phase in which we compared the answers with the outcomes of running security analysis tools on one of their apps. The research addressed the following question:

RQ 3 *To what extent, and how, does a perceived need for security and privacy lead to security-enhancing activities and interactions in an Android development team and result in better software security?*

From the 335 survey responses analysed, we found a high level of reported security importance for the app development, but low use of practical security assurance techniques (Section 5.3.2). Where such techniques were used, this was in proportion to the perceived importance, as was the involvement of professionals and security champions. The frequency of app security updates followed a similar pattern (Section 5.3.6).

Considering the “how” of RQ 3: in the perception of respondents to the survey, app security improvements have been predominantly driven by developers themselves (Section 5.4.1); this is supported by the observation that the assurance techniques first

[26] We might speculate also that security professionals may tend to push teams to use extra cryptography without providing guidance on how.

adopted are those most easily available to developers. GDPR has also had an impact, though the resulting changes for GDPR have been mainly cosmetic (Section 5.3.4).

RQ 3.1: *To what extent do the perceived need for security, the involvement of specialist roles, and the use of assurance techniques in a development team lead to fewer security defects*?

The results of the app analysis showed little relationship with the reported security drivers and development process from the survey; we believe this reflects the inability of the current generation of binary analysis tools to analyse libraries effectively and separately from the main app code. We did however find the involvement of security specialists or champions to be associated with more Cryptographic API issues, probably since they correctly enforce much more Cryptography use (Sections 5.3.7, 5.4.3)

RQ 3.2 What proportion of Android developers have access to security experts?

Section 5.3.2 concludes that between 14% and 22% of developers work with security experts.

RQ 3.3 To what extent do Android developers use assurance techniques?

Only between 38% and 49% regularly use assurance techniques (Section 5.3.3). For the third to a half of the population who were working in teams, the proportion was higher: between 71 and 80% (Section 5.3.5).

Contrasting the high need for security with the low use of assurance techniques and low availability of security professionals, this suggests that there is an urgent need for means to support app developers in adopting security assurance techniques in the absence of security professionals. The following chapters explore one such means.

The author has released a privacy-preserving set of the survey raw results, along with the full questions and data description [186].

6 Intervention Package Creation

The purpose of this PhD project was to answer:

RQ 1 What is needed to make a cost-effective and widely applicable intervention to help UK software development teams achieve better software security?

Following the literature review in Chapter 2, the Expert Survey in Chapter 4 and the Developer Survey in Chapter 5, the next step was to investigate creating such an intervention.

6.1 Requirements for the Intervention

In section 4.14.2, we deduced from the Expert Survey that a cost-effective intervention would best:

- Motivate Active Developers to drive their own security improvements;
- Provide an Incentivisation Session to help do so;
- Encourage developers to adopt six key intervention techniques; and
- Deliver continued On-the-Job Training.

Stack Overflow's 2016 Developer Survey [157] suggests that a majority of developers work in teams, so we conclude an effective intervention should:

- Support developers working in teams

And the Developer Survey suggests that to have a wide appeal an intervention must:

- Not require security specialists, since few teams have access to them (Section 5.5)
- Support developers currently using few or no Assurance Techniques, since few are doing so (Section 5.5)

The six key intervention techniques were as follows (Section 4.4):

Threat Assessment	Identifying and ranking the threats to computer software, a component, or an IT system.
Stakeholder Negotiation	Discussion and negotiation with stakeholders, such as product managers, on security choices
Configuration Review	A review of the way a system or its software has been configured to see if this leads to known vulnerabilities, using manual checking software versions or automated build review scanners.
Automated Static Analysis	The process of using an automated scanner on a web application or network to identify vulnerabilities.
Source Code Review	The manual examination of source code to discover faults that were introduced during the software development process.
Penetration Testing	A simulated attack on a component or system, carried out by a security expert using similar techniques to that of a real- world malicious attacker.

6.2 Constructing the Intervention

The next step, therefore, was to construct such an intervention. The author had expected it to take the form of a website, a book or video [190]; or possibly a code analysis tool (Section 2.1.3), or training-based intervention (Section 2.2.4)

In practice, excellent implementations already exist of such interventions (Section 2.2), but the need for improved security remains. We observed that there were no interventions that both provided the incentivisation session required above and encouraged developers to drive the security improvement process.

6.2.1 The Consultancy Model

As a former consultant and trainer, the author had experience of interventions to support developer-driven changes: specifically, the adoption of the object-oriented paradigm, and later of the agile paradigm, for software development. This experience provided a tried-and-tested model for such interventions. The essence of this 'consultancy model' is as follows:

- A single external consultant facilitator engages on site with the team, leading training sessions, workshops or individual sessions, as required.
- All the work is focussed around the specific project being undertaken by the group.
- Involvement is not full-time, but over a period of weeks or months.
- Confidentiality is necessary, usually requiring Non-Disclosure Agreements or contracts.
- The consultant is paid based on their professional time spent.

Since this consultancy model is familiar to the managers and technical leads of software development teams who have the power to engage with the research team, it seemed a good one on which to base a new form of intervention. To ensure academic credibility, and because of the importance of the trials, no payment was involved. The other four aspects were retained; on-site facilitator, specific project, several-month involvement, and confidentiality.

Given the need for a practical and lightweight process, the target was for the intervention itself to require less than a day's on-site involvement. In practice, that meant the researcher spending most of a day on-site with the teams involved at the start of the intervention; then continuing over several months using teleconferencing or videoconferencing. For research purposes, where possible, the researcher also returned for the final session.

Returning to the requirements in Section 6.1, given the need to motivate 'Active Developers', we determined that facilitated workshops with the teams would be the best approach. We observed that two of the Assurance Techniques are suitable for such workshop sessions:

- Incentivisation Session, and
- Threat Assessment

Indeed, Section 6.1 already identifies the Incentivisation Session as an essential component of any intervention. The next sections explain their implementation as cost-effective team workshops not needing security specialists.

6.2.2 Implementing the Incentivisation Session

In the 'traditional' security specialist approach to inspiring developers, the Incentivisation Session involves an expert or trainer explaining all the bad things that may happen, and using the developers' fear of those events as a motivator (Section 4.11). Unfortunately, fear is only effective short-term as a motivator [98]. Frederick Herzberg in the Harvard Business Review [84] put it like this:

> *KITA [Kick in the 'Rear'] ... has been demonstrated to be a total failure….*
> *A negative KITA does not lead to motivation, but to movement*

Indeed, researchers who tried this approach found it ineffective in the longer term (see Section 2.2.4).

Accordingly, as an alternative approach to fear-based motivation, we wanted an Incentivisation Session that would help developers engage with security better and lose their fear of it. Devising such an Incentivisation Session was perhaps the biggest challenge of the project**.**

Fortuitously, while considering this challenge the author received an enquiry from a colleague working as a consultant 'Agile Coach'. He wanted to use a game, the 'Agile Security Game' [184], invented by the author as a fun workshop session for the AgileNorth 2016 conference.

This game was based on the 'Mumba' role-playing game invented by Frey et al. [65], to help elicit participants' prior experience of real-life security attacks. The 'Agile Security Game' variant, however, was designed simply to educate developers about security. In it, participants act as product managers, selecting security-enhancing product improvements with varying costs and learning whether their choices deter attacks.

The colleague wanted to use the game in Company A (see Section 7.2.1) to help motivate development teams towards security. The author realised it was being requested as an Incentivisation Session, and proposed delivering the full planned intervention in that company—a proposal that was accepted.

6.2.3 The Agile Security Game

The following inset describes the Agile Security Game.

The Agile Security Game

The facilitator arranges the room with separate tables with 2-6 chairs around each table, and a display screen visible to all the players. Each player gets a sheet of paper describing a product with poor security; each table of players becomes a team taking on the role of product manager. The facilitator also prints out separately a set of 'security story cards' to be given out to each table, each card describing and pricing, in story points, a possible security enhancement.

After an introduction by the facilitator, play proceeds in four rounds, representing development 'sprints'. At the start of each sprint the facilitator hands out security story cards to each table. The players then select the stories for 'their product developers' to implement, subject to a budget for each sprint expressed in story points. The discussion around security story selection is the main point of the workshop, and should not be hurried.

Once all the teams have chosen their stories in each sprint, the facilitator explains that the product developers have shipped the corresponding enhancements, and that subsequently there have been attacks on the software. The attacks for that sprint are shown on the screen, along with which security stories mitigate each. Based on that, some product owners will be 'damaged'; others not.

Following the last sprint there's a facilitated discussion, where the teams state what they learned; the facilitator may also explain some of the game workings, including some 'security stories' that are not recommended practice.

6.2.4 Implementing Threat Assessment

The Threat Assessment workshop was also challenging to implement. Much of the literature [114,154] describes a heavyweight process taking a while to set up and requiring considerable knowledge of possible technical threats, preferably with support from a professional with a detailed understanding of both the industry sector and current cyber threats to it. But such a process would be expensive in time and commitment, and the required professional knowledge was not available.

However, in this case the researchers' own experience was valuable. As technical lead for a major mobile money project, the author had faced this problem in a commercial project. With the help of Alec Muffett [208], a consultant security expert, he had developed a lightweight brainstorming process to identify threats and potential attackers [189]. While this may have lacked the rigor of the threat modelling approaches used by some large companies, it had served to deliver a product which the security analyst at their customer T-Mobile described as '*the most secure app they had seen that year'*. Accordingly, the author used the same approach here.

6.2.5 Threat Assessment Description

The following inset describes the Threat Assessment workshop as we implemented it in Developers Security Essentials.

Threat Assessment Session

Brainstorm Threats

Brainstorming requires one member of the team to act as facilitator. The team sits in chairs in a circle facing each other and a flipchart (or whiteboard). The facilitator writes a question to focus the discussion, along the lines of 'what threats do we face'. Then everyone suggests possible threats – without analysing each or attempting to filter out any of them. As they do, the facilitator writes down each threat (whether sensible or not) on the flipchart.

As ways of generating ideas, participants also consider also who might be the attackers – what would they want and how would they go about getting it. That may generate a different set of possible threats. Similarly looking at the architecture of the system in detail, concentrating particularly on interfaces between systems and components is a further excellent way to find threats.

From the flipchart, the team creates a document listing each of the threats: the attacker, what they might get from it, how they might get it.

Assess Threats

The second step is to assess the threats. It should be a separate session after a break, since the analytic type of thinking involved is different from the brainstorming in the previous section.

In this workshop, the threats are written up on a list – usually on a display screen. The participants address each in turn, perhaps by voting on which ones look most important to address first. For each, they estimate:

1. Likelihood: Low, Medium or High
2. Impact: Low, Medium or High.

Obviously, these will be relatively inaccurate assessments: the aim will only be for finger-in-the-air accuracy. If the workshop can involve a Security Specialist, they may have helpful knowledge about likelihoods of different threats, and possibly even typical impacts.

And then, taking the threats with high impact or likelihood first, the team identifies possible mitigations – possible things to do to deal with the threat. Some mitigations may be in code, or changes to functionality; others might be processes, discussions with other teams, or even preparing a plan for dealing with a successful attack. They then estimate development costs for each using the same process as estimates for any other piece of development (story points, perhaps).

During this workshop, participants consider the other five assurance techniques described at the start of the chapter as possible mitigations.

6.2.6 Implementing On-the-job Training

Finally, Section 6.1 required On-the-Job Training (Section 4.12). Specifically, we wanted a regular 'nudge' [168] to the team as a reminder to take action on what they had determined in the initial workshops and on what they had discovered since then. For this we used a monthly follow-up meeting, usually by video conference. The facilitator asked

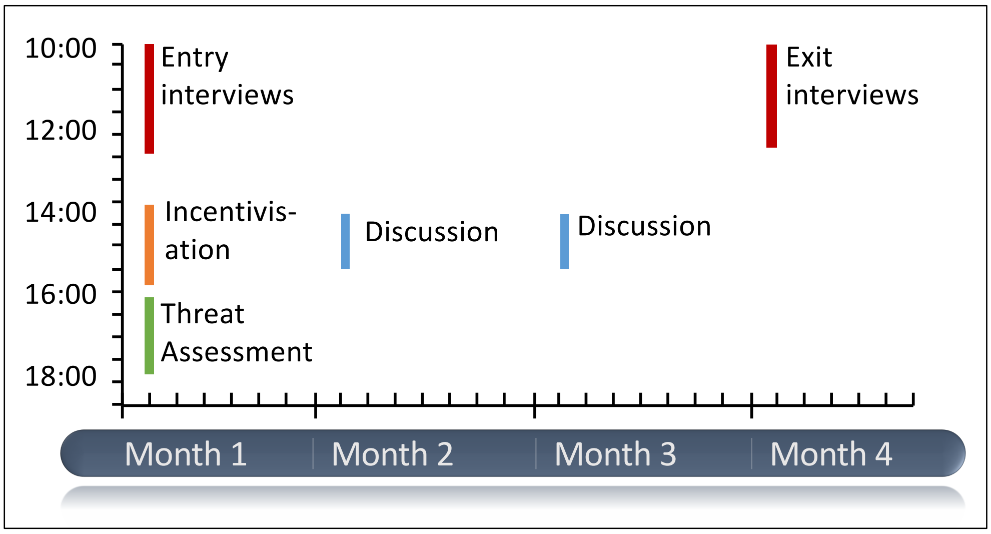

Figure 39: Typical Schedule for the Interventions

open questions with the team about their progress over the previous month, and discussed issues that came up.

6.2.7 Intervention Schedule

Figure 39 shows a typical schedule for the interventions. The work with each company spanned 3-4 months, with only two days on site at the start and end. The involvement time was limited to four months in order to get the feedback from the exit interviews reasonably quickly.

6.3 Facilitation Approach

Given the Active Developer model, we wanted to use language and approaches consistent with developers as the instigators of activity rather than the language and approaches of commands and formal processes (Section 4.3).

Therefore, at no point did the facilitator interact with the development teams using terms like *"you must"* or *"it's essential that"*. Also, throughout the workshops and game, the researchers allowed the developers themselves to drive the solutions; as facilitators they provided only guidance.

Furthermore, the researchers were aware that Source Code Review and Penetration Testing are relatively expensive for a team to adopt (Section 4.13.1) and therefore seldom within a team's power to achieve themselves. So, the decision was taken not to promote them explicitly. Instead the Intervener concentrated on promoting, when opportunity arose, the other main techniques identified in Chapter 4: Configuration Review, Automated Static Analysis and Stakeholder Negotiation.

An online book, video, and materials [182] supported the package.

7 Package Trials (Magid)

This project investigated having a consultant lead the Developer Security Essentials intervention in three different organisations.

We called the project 'Magid' after some troubled superhuman interveners in a novel by Diana Winn Jones [93].

Our research question for this project was straightforward:

RQ 4 *What security outcomes did the 'Developer Security Essentials' package have, and what aspects contributed most to those outcomes?*

This is an 'overview question', and difficult to answer with precision. Accordingly, we unpacked it to create several sub-questions. First, we wanted to know the short-term outcome from using the intervention:

RQ 4.1 *What security improvements and changes were made in the development teams' ways of working and developed products in the short term due to the intervention?*

Next, given that some previous interventions described in the literature had failed to have a long-term impact (Section 2.2.4), we wanted to know whether this was true for this intervention:

RQ 4.2 *To what extent did the changes made in the development teams' ways of working persist over a one-year timeframe?*

And finally, we wanted an insight into what was happening, to support modifying the intervention in future:

RQ 4.3 *What aspects of software development as practiced by the teams supported or hindered adoption of the various security techniques?*

7.1 Research Method

Section 3.2 justifies the choice of Action Research in this project. The choice of variant of Action Research was dictated by the situation. The interventions took relatively little time and did not involve discussion of individual problems, nor did they overtly emphasise organisational learning; that ruled out Action Learning and Action Science. The subjects did not influence the intervention design, nor participate in theory generation, making Participatory Action Research unsuitable. So, the method adopted was Canonical Action Research (CAR, see Section 3.5), with the author working as 'intervener', directly with the participants ('client').

We addressed the principles of CAR (Section 3.5.2) in the project as follows:

Researcher-client agreement: The lead researcher agreed the form of the intervention, the nature of the workshops, and the specific approaches and choices of participants with the team leaders in each case.

Cyclical process model: This was the hardest principle to address; indeed many Action Research projects in software engineering have had only a single cycle [147]. In practice all of our participating development teams were using forms of Agile processes, and in each case the three-month duration of the intervention covered several agile development iterations, so the monthly follow-up intervention session allowed retrospection and improvements. However, the full intervention was used only once, not in each monthly session, so the monthly sessions could not be considered cycles in the Action Research sense. In planning the project, we anticipated a later Action Research cycle with the same teams.

Theory use: The interventions were based on theory derived from the earlier projects; the research process itself also generated further theory for later use: specifically, in the Magid 2 project.

Change through action: In each case, the participating developers decided on possible changes and prioritised them themselves; the normal development prioritisation techniques (Kanban boards or task lists) ensured that a record was kept and that the timings of the changes was known.

Learning through reflection: This happened mainly though the final interviews and the discussions around them. Later reflection by the researchers was captured and fed back though the research papers and discussion with participants.

7.1.1 Practical Approach

On arrival, the researcher met with the main contact, and arranged the signing of the organisation consent form. He then, in a meeting room, interviewed in sequence four or five of the team members, each signing the individual consent form at the start of the interview. The interview protocol is given in Appendix F. The interviews typically took 20 minutes each.

The researcher then met back with the organiser, and together they set up the room for the first workshop, the Agile App Security Game, as given in Appendix D. The teams then arrived and, after a brief introduction by the team lead and the researcher, they played the game as in Appendix D.

After the game, and following a half hour break, they did the Threat Assessment session (Section 6.2.5). This used a simple brainstorming approach [61]. The researcher used a flipchart or whiteboard to capture suggestions from the group, encouraging as wide a scope as possible and discouraging criticism or selectiveness.

The facilitator and team leaders then selected the five or six of the most likely and damaging threats identified and discussed them in more detail, identifying possible ways to mitigate them. The teams also kept the flip chart sheets (or screenshots) and transcribed the full list of threats for reference afterwards.

During the discussion of the mitigation for each threat the researcher introduced suggestions of possible approaches where these were not forthcoming from the participants, mainly from the list of key assurance techniques (Section 6.1). The suggestions usually took the form of short War Stories (Section 7.3.6).

The monthly follow-up sessions were mostly by videoconference between the researcher and the team lead and a selection of the rest of the participants. The exit interviews were mostly in person, of the same people as the entry interviews, and took about 20 minutes each.

One point of importance related to:

RQ 4.2 *To what extent did the changes made in the development teams' ways of working persist over a one-year timeframe?*

was whether participants understood the aim of the intervention, so that they themselves might use Assurance Techniques later in future projects; accordingly, the exit interviews included an open question to elicit whether the participants appreciated the need for Threat Assessment and perhaps other interventions. The exit interview protocol is given in Appendix G.

7.1.2 Research Analysis

The focus of this research project was the improvements achieved by the subjects through their own efforts, as focussed by the intervention. Accordingly, in our evaluations of the results we concentrated on practical, objective, improvements in security as a result of the interventions; feedback on the interventions themselves and the way they worked was treated as a secondary outcome[27].

A main research sub-question for this project was:

RQ 4.1 *What security improvements and changes were made in the development teams' ways of working and developed products in the short term due to the intervention?*

To measure the intervention's security effects, we needed a baseline with no intervention. A-B testing, requiring a different team working in parallel, was not practical. Instead, we used a longitudinal approach, deducing a baseline ('no intervention situation') from the initial situation plus a knowledge of the original plans by the team leaders to improve security over the same timescale.

The author[28] then carried out the pre- and post- interviews and the Developer Security Essentials intervention with the development teams and transcribed and analysed the workshops and interviews as described in Section 3.5.3.

In this coding we looked for aspects of security improvement—including in learning and attitude—implied by statements from the speakers. To justify improvement, we coded the initial interviews to provide evidence of a baseline security activities and awareness before the start of the interventions.

We analysed the kinds of interaction involved in the workshops and looked for 'Motivators and Blockers' : aspects that helped and hindered such security improvements [14]. We coded signs of new knowledge in the team, new activities related to security, and evidence of improvements in the security of developed software.

Given that the teams were prepared to work with us, we knew that at least some of them had some prior interest in security. In the interviews and our analysis, therefore, we were careful to distinguish new security activities and enhancements attributable to the

[27] Note that in the later, 'Magid 2' project we changed focus, and we altered our research method accordingly. See Section 3.6.

[28] All the work and analysis were done by the author of this thesis; Ingolf Becker acted as second coder for the dual coding.

interventions from those that had been planned or contemplated before the trials and those due to other external factors such as customer demand or other security specialists.

7.1.3 Research Numbers

Three companies participated, generating a total of 19 hours of audio.

The final code book consisted of 5 families of codes, making a total of 41 codes, applied to 1405 quotations in total.

7.2 Participating Companies

This section introduces the three companies, with the projects and development teams involved. To preserve confidentiality, we have changed all names and the exact functionality of the products involved.

7.2.1 Company A

Company A is a small-to-medium company employing around 50 people in the UK. Set up about 10 years ago, it has a single product which is sold both web-based as 'software as a service', and as an installable system for clients' own sites. This product manages sensitive data, and is used by large multinational organisations, including several that are household names.

The product is a web application and is shipped or installed as a single codebase implemented mainly in Java. The 'software as a service' implementation is hosted on systems at a leading hosting provider; clients with their own installations manage them themselves, which means that the developers must provide support for older software versions. The nature of the app means there are complex rules and permissions as to which users may see what. These are typically implemented for each new customer installation by a separate team of configuration specialists.

7.2.1.1 A's Developers

Participating in Company A's workshops were developers from two teams. The teams worked on separate tasks on a shared code base, and each team had a technical lead, the 'architect'.

The company development teams show some of the enthusiasm and characteristics of a start-up. We observed a culture of technological improvement, and a willingness to embrace change. Both teams use an agile approach to development based on Scrum, with sprints, stories and a prioritisation process.

7.2.2 Company B

Company B is a tiny non-profit start-up, run on a part-time basis by two professionals: an educationalist and a software project manager. Other staff also assist on a part-time basis. The company purpose is to provide work experience for promising young people who would otherwise be unable to get initial jobs in IT. They undertake pro-bono software development projects for charities. During the interventions we worked on two projects: first a marketing website for Company B itself, and later a project for an art installation involving voice recognition and public interaction.

7.2.2.1 B's Developers

The development team constituted the educationalist (B1), a project manager (initially B2), and two student developers with very limited experience (B3, B4). Typical interactions were dominated by B1, but with contributions from the others.

7.2.3 Company C

Company C is a well-known and long-established multi-national organisation, providing information services mainly via the Internet to a range of companies and individuals. The department we worked with provides membership facilities, managing payments and controlling access to the company's services.

7.2.3.1 C's Developers

We were introduced to the company by C1, an experienced software tester, who had an interest in encouraging security. The team members involved were testers, managers and programmers. The membership system is a mature software, but there is a policy of continuous improvement; currently the system is migrating to a micro-services architecture. All the team were competent and experienced professionals, but in contrast to company A we noted more emphasis on inter-departmental politics.

During the interventions, C company changed policy on testing, disbanding the separate QA team; three of the staff we had been working chose to take redundancy. Two of these we managed to contact, and they agreed to exit interviews by video and telephone. As researchers, we found arrangements difficult to make (probably due to the reorganisations), and managed only one follow-up session after two months, and to ensure it took place we held that on the customer site rather than by video.

7.2.4 Interview Participants

To help identify the effects of the interventions, we interviewed team members in each company both before and after the process. We agreed up to six interviewees with each company, enough to provide a full range of roles, and requested accordingly a cross section of the roles and experience within each team. Table 12 shows the interviewees, with the role, gender and an estimate of the professional experience of each.

We have included quotations in the remainder of the paper, in *italics*. Where the speaker can be identified, we have cited the appropriate ID. In the recordings of group sessions, however, it was rarely possible to identify individual speakers, and quotations are cited with role and context accordingly, e.g. *'Developer, Threat Assessment.'* We have edited the quotations to protect confidentiality and indicate context: square brackets show additions and replacements; ellipses show removals.

Throughout the rest of this chapter we refer to the author, who carried out the interventions, as the **Intervener**.

For a variety of reasons (equipment failure, researcher's omission) three sessions were not recorded: the first follow-up Discussion for Company A, the first follow-up Discussion for Company B, and the Threat Assessment for Company C. As mitigations, the author kept notes and a copy of any outputs from those sessions. Where referenced, these are indicated in brackets as '**(not recorded)**'.

Table 12: Interviewees from Each Company Team

Organisation	Identifier	Role	Gender	Experience
Company A	A1	Architect	Male	17
	A2	Programmer	Male	2
	A3	Programmer	Male	14
	A4	Programmer	Male	3
Company B	B1	Manager	Female	25
	B2	Manager	Female	13
	B3	Developer	Male	-
	B4	Developer	Male	-
Company C	C1	QA	Female	7
	C2	Manager	Male	13
	C3	Programmer	Female	3
	C4	QA	Female	10
	C5	Developer	Male	10

7.3 How the Sessions Went

This session analyses the intervention sessions themselves, without considering their longer-term impact.

7.3.1 Intervention Time Requirements

Referring back to Figure 39 in Section 6.2.7, the timeline needed for the interventions, it will be seen that, despite the long-elapsed time, the total effort required from the intervener was relatively short: a total of two days (plus counting travel time). What is more, at least four hours of that were research interviews and not part of the intervention itself. So, the total effort spent for the interventions was less than one working day. Adding another day for preparation – scheduling, preparing materials for the workshops, etc. – the total time spent by the intervener on the interventions was less than two working days for each company. In terms of team effort involved, the participant numbers and times involved were roughly as shown in Table 13.

Table 13: Actual Participant Time Cost

	Time involved (h)	Company A participants	Company B participants	Company C participants
Incentivisation session	1.5	15	4	16
Threat modelling workshop	1.5	15	4	16
Follow-up 1	1	6	4	
Follow-up 2	1	6	4	8
Exit workshop	1	10	4	
Total participant hours		67	24	56

We can therefore summarise the cost of this set of interventions as shown below:

Participants	**Total time**
Intervention facilitator:	15 person hours
Development team:	20 - 70 person hours

The cost of the interventions, therefore, is relatively small, and is within the scope of a wide range of organisations.

7.3.2 Effect of the Incentivisation Session

All three groups engaged well with the Agile App Security Game (Section 6.2.3), with each group discussing the security choices at length. Participants reported different benefits, though. Some saw it as teaching about security decisions:

> *The game was fun, I did enjoy the game. And it was proving as per usual, that you can't... whatever you do, you are going to lose somewhere (A3)*

> *I think what it proved is how challenging it is to get this right. It actually hammered home that circumstances that we all work through every single day, which is this balancing act, between how much time you have got, vs what is being demanded from you, from some customer somewhere. (C2)*

Others as encouraging communication and knowledge exchange:

> *Yes, that was good. It really got everybody talking. And it could be linked back to things that were happening within [Company C]. 'Yeah, we do something like that, we do this and this' or 'yeah, no, we don't do anything about that - and we really should'. (C1)*

7.3.3 Effect of Threat Assessment

The Threat Assessment workshop generated some ways of thinking and conclusions that were unexpected for the participants:

> *I never really thought about 'who would', so much, until you put up 'why would somebody and who would they be' (A4)*

Company A, in particular, identified two kinds of threat that were totally different from the ‘anonymous hacker exploiting one of their coding mistakes’ that they had been envisaging: customers viewing each other’s data; and hackers exploiting out of date components. They also identified a range of further possible issues:

> *I find it a little concerning that there are so many attacks that we traditionally haven't mitigated against. ... Stuff like social engineering. (Participant, Threat Assessment)*

At the time of the first workshop, Company B were starting to work on a website for their company. Like many developers they had not considered it in terms of security, and were surprised to find when they thought about it that there were issues: the need to store personal details of applicants, for example. Indeed, the discussion prompted a significant change in their website software architecture and their rules for team members:

> *A big take-away for me, is that we started with a much grander idea of we needed to be doing, and there was all this personal data... and now we have*

> *said oh no, we just need email! And more important is things like making sure that people's software on their computers is up to date! (B1)*

The teams in Company C had a much greater a priori understanding of the security issues; indeed, C1 had been handling security alerts and issues as part of her day-to-day role. So, though the Threat Assessment session did identify some possible issues (such as physical access to developer workstations), the main impact of the workshop was the sharing of knowledge between people with different roles:

> *I don't think we came up with any extra, really. I think we were kind of… apart from us talking about it more, we have not really been able to influence a lot of other change. But it was good to get everyone talking again, and thinking about it. (C1)*

7.3.4 Effect of the Follow-up Discussion Sessions

The discussion sessions varied far more between companies than the other two sessions, because of the different needs of the projects.

With Company A, in the first Discussion the architects requested a prioritisation session (not recorded). Using a shared 'Trello'[29] board, the architects and Intervener prioritised a list of possible security enhancements derived from the Threat Assessment session. While the intervener contributed information about industry security decisions, the main point made was that these prioritisation decisions were for product management, with the role of security and development experts merely being to provide context for the decisions.

The second discussion session with Company A followed more of the pattern the researchers were expecting: a discussion of the new security activities the team were implementing, and a discussion of the advisability of disk encryption for the server:

> *What [are] the advantages of database encryption, and what wouldn't [it] give us, compared to application level encryption… and [what are the] risks with encryption itself? (Architect)*

With Company B, the first session (not recorded) was a similar discussion, discussing the security improvements arising from the first Threat Assessment session and comparing industry practice. By the time of the second session, however, the team had moved on to starting a new project, so the session became an ad-hoc Threat Assessment session, using a shared Google Doc[30]. The following are examples of the threats they identified relating to public voice input for their product:

> *Don't want swearwords to appear in the output.*
> *Microphone (voice recognition) or app overhearing someone else's talk.*
> *Someone dominating the [microphone]. Could set a time limit….*
> *Commands that might damage the database. NOSQL. (Shared document)*

While not all the items identified may need addressing, the concerns show a promising understanding of the wider nature of security threats.

With Company C the follow-up discussion sessions required considerable persuasion on the part of the researchers to organise; possibly with the considerable internal experience within their own teams, and the continuous reminder of regular security incidents to

[29] https://trello.com/

[30] https://www.google.co.uk/docs/about/

manage, they felt little need of further external input. The single session we arranged also felt less positive than any of the other sessions with any company – we now know this was probably an effect of the redundancies. The discussion served as a review of the current security improvements in action, but mostly covered relationships with other departments: improved security enthusiasm from product management:

> *[In a recent strategy meeting] those that ultimately decide what we do, and in what order actually quite happily went: 'well if it is a security issue, we should fix it', whereas normally it is a case of 'well, we have got these deadlines to hit, and we have got this stuff to deliver' (Participant)*

And a dysfunctional relationship with the security department:

> *There is a lot of stuff, security-wise, that doesn't seem to have a home, doesn't seem to have an owner. The Security Team is happy to shout about it when it suits them, but I wouldn't necessarily suggest that they own it, or help you particularly. (Participant)*

7.3.5 Additional Sessions

Company A at their request had an extra presentation by the intervener. This followed the Entry Interviews and described the 8 interventions, the process we were planning to use and the relationship to the OWASP Top Ten issues. This contributed to the success of the interventions with Company A – indeed the exit interviews from B and C suggested such a session:

> *[I would like to have included] more visual presentations. I mean, we used the cards, and the discussions were beneficial but maybe a visual element to that maybe a video or a presentation would help us explore in a different way. (B3)*

> *Maybe from my perspective it would have great to get a bigger picture of what the programme was about. "So, this is going to be across three different stages, and first we will have ..." (C3)*

Both Company A and Company B had an exit workshop (not formally analysed) when the lead researcher was on-site for the exit interviews. In the case of Company A, this was a discussion of how to evaluate risk for different security threats; for company B, a discussion of the education value of the sessions and possible security-based careers for the students involved.

7.3.6 Team Interactions

It is instructive to examine how the nature of the discussions in the workshops varied between companies. Figure 40 contrasts different styles of interaction (by both developers and Intervener) during the workshops and discussions with each company, showing the proportion of dialog devoted to some of the most important categories of discussion. The workshops varied considerably in the proportions of time devoted to the main activities: knowledge presentation, to finding issues and vulnerabilities, and to addressing the issues discovered. This reflects differences in culture, structure and projects between the teams.

As Figure 40 shows, Company A saw the largest proportion of time presenting knowledge – people stating facts and information about security and the products – probably because of the high level of software expertise and the presence of security knowledge within the team. Company B, with the least experienced team, had less knowledge to share and

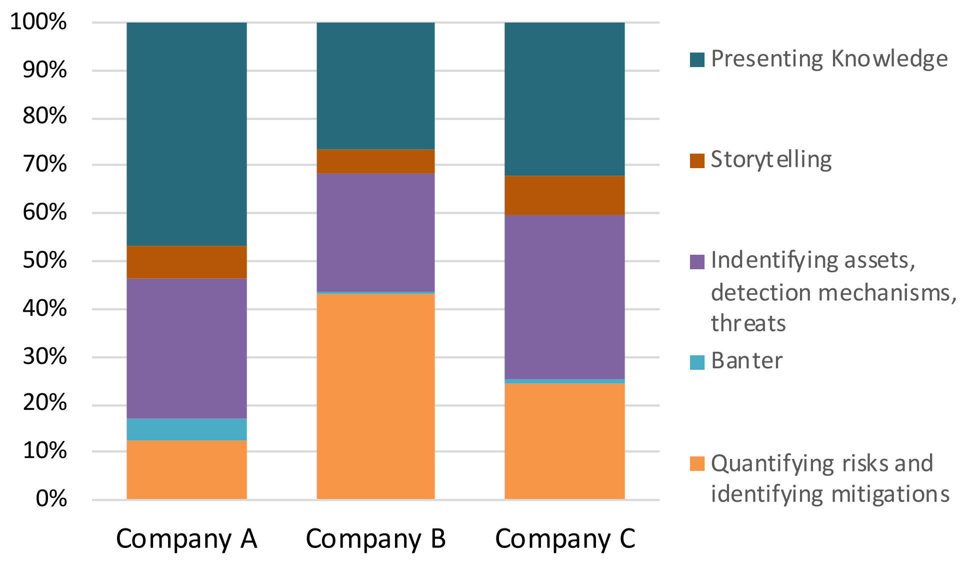

Figure 40: Styles of Participant Interaction

found it easiest to concentrate on possible attacks. Some of Company C's team had extensive knowledge both of their systems and of security aspects, but this knowledge wasn't well distributed across the team, so a good deal of the session constituted Presenting Knowledge by team members. Company C's group identified no detection mechanisms, perhaps because detection and handling intruders was the responsibility of a separate security department.

In all three companies we observed examples of an effective way of presenting knowledge, Storytelling (shown separately in the diagram), narrating how a participant addressed or was affected by security issues [82].

A particular revealing measurement in terms of culture is the amount of Banter, friendly joshing and jokes, involved: Team A's high performing and relaxed culture had a good deal; Team C's more formal culture evinced little, and Team B, with a large disparity in status between participants, had none.

The differing proportions reflect different emphasis in the workshops. For Team A, the novelty was discovering the true nature of their security threats, while addressing them would be business as usual and so required less discussion. For Team B, starting from virtually no security knowledge and working on less security-critical projects, it was more important to find ways to deal with the smaller set of risks they did identify. And for Team C, with good security expertise but poor communication between teams, most of the benefit was in pooling knowledge fragmented among the participants, and hence discussion was fairly evenly spread between the three main activities.

7.4 Outcomes

This section explores the objective outcomes in terms of identifiable improvements in the development process and product security for each team, addressing the first research sub-question:

RQ 4.1 *What security improvements and changes were made in the development teams' ways of working and developed products in the short term due to the intervention?*

7.4.1 Outcomes for Team A

There were at least two significant improvements in Team A's product and process security as a result of the interventions. Beforehand, the developers had been thinking of security improvements as line by line improvements in the code they themselves had written. Afterwards, they understood that their most effective security improvements were likely to be elsewhere:

> *I find it a little concerning that there are so many attacks that we traditionally haven't mitigated against. (A Workshop)*

Specifically, they made three changes. First, in a form of Configuration Review, they introduced a component security checker to their build cycle and embarked on a program of updating and replacing components according to their security vulnerabilities.

> *We [have built] the OWASP dependency checker into our build process, ... and established a process for how we deal with new vulnerabilities in existing libraries or adding new libraries or upgrading libraries. (A1)*

Second, they identified their own existing customers as competitors with each other, and therefore potential 'attackers', and identified that the permissions functionality was therefore a major privacy issue; making fixes in this area was likely to give security wins:

> *I have a ... task to check user permissions, and check that a user has access to that specific entity or a set of those entities (A2)*

Thirdly, they introduced a monthly focus on the OWASP 'Top Ten' vulnerabilities, one at a time. This approach had been mooted prior to the interventions but was only carried out after the initial workshops:

> *[A team architect] puts out a 'we're working through this one this week', and he puts up a link and it has got everybody's name next to it, and you read through it, and then there is more information if you want. You can ask questions, and we have got a good internal issue tracking board. Any kind of potential thing, big or small, goes on there, and it can get prioritised into our work properly. (A4)*

And in one of the discussion sessions they established that the prioritisation of security features required product management, not development, decisions:

> *That is where the priority call would come from. I think [Product Management] do understand it, ... but there is always going to be that element of weighing up (Group Session)*

An unusual and intriguing approach they also tried was having one of the team be a covert 'saboteur', occasionally introducing security defects to see if the review process would find it; in practice, though, they found it problematic:

> *One team did the saboteur exercise ... It was a bit mixed. The saboteur didn't enjoy being a saboteur... (Group Session)*

7.4.2 Outcomes for Team B

Team B, with little prior security experience, had more potential improvements in process and in product security. As a result of the first Threat Assessment process they made several changes. They abandoned plans to store personal data in a website server database:

> *We said about the form, that it would send an email [instead of saving personal data on the server]. (B1)*

In addition, they introduced improved security and backup for development workstations and code repositories, against the threat of malicious code modifications or access to personal data:

> *[We did] an audit on our computer systems: on our laptops… and the laptops that the students are bringing. We do scans, and make sure that the antivirus and anti-malware protection is all up to date. (B1)*
>
> *I also update my data a lot more, I back it up, not just to a file server but with a USB. (B2)*

Later, as they started further projects, they introduced their own Threat Assessments:

> *We developed a threat model at the start of our [later] project, and it is used in the code reviews and testing. (B1)*

These seem to have been effective; for example, they identified a need to secure their API key management, an example of Configuration Review:

> *We need to make sure that … those [API] keys don't become public, and that all students know that we have to do that. (B1)*

7.4.3 Outcomes for Team C

There were no identifiable improvements to Team C's process or product directly attributable to the interventions. The primary reason for this is that their security knowledge and practice as a team were already good: better than they may have realised:

> *I'm not sure too many changes were made. (C1)*

While some changes were made as a result on ongoing security improvements:

> *I'm much happier because we started working with Two Factor Authentication… for our client… admins… (C5)*

the participants did identify improved communication and understanding as resulting from the interventions:

> *I think it got everyone talking about security a bit more, especially within our team... There was a lot of security things going on that I didn't know about. (C1)*

7.4.4 Security Learning as a Result of the Interventions

As discussed in Section 7.1.1, we looked for participants' appreciation of the importance of Assurance Techniques, especially Threat Assessment. Table 14 shows the results of the corresponding analysis, along with brief descriptions of each participant. The top lines (A1–C5) consider the exit interviews for each participant and identify how many statements indicated internalised understanding of each assurance technique. The bottom three lines consider group discussions towards the end of the process and show the

Table 14: Evidence of Learning by Interviewees

ID	*Role*	*Experience (years)*	**Automated Static Analysis**	**Product Negotiation**	**Configuration Review**	**On-the-job Training**	**Threat Assessment**
A1	Architect	17	1	4	3	3	3
A2	Programmer	2				2	2
A3	Programmer	14		1	3	2	
A4	Programmer	3			2		1
B1	Manager	25		1			2
B2	Manager	13					
B3	Programmer	<1					
B4	Programmer	<1					
C1	QA	7	1		1	1	1
C2	Manager	13		1			
C3	Programmer	3					
C5	Programmer	10		1			3
A	Team discussion		6	1	11	6	2
B	Team discussion			2	3		4
C	Team discussion			4			1

number of participant statements that showed similar understanding. Deeper shades of blue highlight higher counts.

Since the Intervener was not promoting Penetration Testing and Code Review, there was no attempt to analyse the discussion for them. Surprisingly, only A1 showed appreciation of the Incentivisation Session. So, none of these are shown in the table.

As Table 14 shows, though both teams B and C implemented many of the assurance techniques, many of the individuals we interviewed did not evince a strong understanding of the reasons and approach to do so for future projects. Note however that since this understanding was estimated from participants discussion and there were no explicit interview questions about each technique, the omission may not reflect the true understanding of the participants involved.

However, members of the Team A gained a good understanding of the techniques; we can conclude they did not implement Automatic Static Analysis as a positive decision based on the value of using it—an example of good security process. The leaders of teams A and B indicated they had learned aspects of future Product Negotiation:

> *I guess, one challenge, as always, is playing what we, as architects, believe are the most pressing security concerns, against what customers are asking for in terms of dealing with security concerns. (A1)*
>
> *I would …feel confident to be able to talk to people about our security policies and how we manage security (B1)*

And that they appreciated the need for Threat Assessment:

> *[If I was advising a team on security] I think brainstorming threats and vulnerabilities and assets is really helpful. (A1)*

> *And one of the things that I think we probably are doing, as a result of being part of this process, is that auditing, that thinking things through first, what are our security issues, what are our risks, and how we are going to deal with those, in terms of the design. (B1)*

7.5 Outcomes after One Year

The additional interviews after one year for A and B allowed us to address:

RQ 4.2 *To what extent did the changes made in the development teams' ways of working persist over a one-year timeframe?*

7.5.1 Outcomes after One Year for Team A

A year later, secure development had become increasingly important to sales (Product Negotiation):

> *With every sale we will get stringent questions around security. ... I think, increasingly, there are more questions around development processes, and application security. And clearly, without being able to answer those questions satisfactorily, we wouldn't be able to sell. (A1)*

Configuration Review is now part of their development process:

> *The OWASP dependency checker is very much embedded in our process. We have never yet got it to the point where all the dependencies are green! But we do now appear to be at the point where it is a regular part of our process to check for new vulnerabilities that have been found, and to add upgrades for those libraries that contain known vulnerabilities, within, either, the next release, or the release after that, depending on how much other pressure there is on our road map. (A1)*

Disappointingly, the two innovative forms of On-the-job Training instigated by the teams independently of the workshops had not continued:

> *Sprint by sprint [we were] picking up one of the OWASP Top Ten, and getting all the developers to review it, and identify issues where we weren't meeting those things. That, sadly, has fallen by the wayside ... [because] we didn't have the bandwidth on our road map to deal with the things that people were highlighting. (A1)*

> *[The secret saboteur] carried on for a few sprints, I think it didn't work out quite so well, when somebody got accused of being the saboteur, when actually it was just a genuine mistake they made! It then became very embarrassing for that person. I think that fell by the wayside, slightly! (A1)*

They were, however, considering using an Automated Static Analysis tool:

> *One of the things on our backlog is bringing in SonarQube which might potentially identify security issues in the code. (A1)*

Table 15: Summary of Techniques Adopted after One Year

	Team A	Team B	Team C
Incentivization Session		In regular use	
Threat Assessment		In regular use	
On-the-Job Training	Introduced, but abandoned	In regular use	Already in place
Product Negotiation	In regular use		Already in place
Configuration Review	In regular use	In regular use	Already in place
Automated Static Analysis	Planning introduction		Already in place
Penetration Testing	Already in place		Already in place
Code Review	Already in place	In regular use	Already in place

7.5.2 Outcomes for Team B after One Year

Since the emphasis for Team B is on training, it was encouraging to find after one year that they had continued finding ways to instigate development security. Specifically, they now had Incentivisation Sessions, and they were teaching Threat Assessment:

We are just starting a new project, so part of the induction, and part of the on-boarding for all the students, is that we do a little bit of security training, and we do a threat modelling exercise. (B1)

They now had security Code Reviews:

And then, as far as our code reviews are concerned, [we are] actually looking at security aspects, at every stage. So, each time we are doing a code review, security is one of the things on the form to tick. (B1)

And they had various forms of On-the-job Training:

We had a few other students who come in at a later stage, and [B4] did a nice 'Brown Bag' talk on security. And we are passing that on. (B1)

Particularly gratifying for us was that the intervention helped identify an aptitude in one participant for security work, and to inspire a choice of career:

And [B4] who was going to be a struggle because of his Maths and English, his options going forward are quite limited, but he is… about to start a Level 2 Traineeship in Cybersecurity! Something that came out of your research was really how interested he is in it. (B1)

7.5.3 Techniques Adopted

Table 15 summarises the above outcomes: shaded cells indicate new assurance techniques in use as a result of the intervention process. As discussed in Section 7.1.2, throughout the analysis we were careful to distinguish changes arising from the interventions from those due to pre-existing plans or other external factors; so, for example, Team A's trial of On-the-job Training and plans for Automated Static Analysis are not credited to the interventions.

7.6 Blockers and Motivators

The previous two sections show the outcomes from using the interventions, but provide little indication of what was happening to lead to those outcomes. The large amount of data available in the form of transcripts of the workshops—including the follow-up session—allow us to address:

RQ 4.3 *What aspects of software development as practiced by the teams supported or hindered adoption of the various security techniques?*

Using the same open coding as before, we analysed the interviews and workshops to identify 'blockers', problems that threatened to prevent adoption of the practices; and 'motivators', incentives for practicing secure software development. In total, we identified 44 mentions of blockers and 27 mentions of motivators.

7.6.1 Problems and How They Were Overcome

Analysing the workshops and interviews in more detail, we identified several problems encountered in carrying out security enhancements. We have termed these 'blockers', and against them we have identified corresponding 'motivators' – benefits or practical solutions – that helped team members to overcome them. Table 16 shows a selection of the most important such blockers and their corresponding motivators.

Table 16: Blockers and Motivators

Blocker	Motivator
The significant work involved in upgrading a range of components, and modifying the code to support revised APIs where required for the upgraded versions. *We haven't necessarily got to as much of it as we would have liked to. Hopefully, the architect guys… [will] try and feed some of those stories in. (A3)*	The satisfaction of seeing 'red lights' turn green as the components were updated: *You've got lights that you can turn green - it becomes relatively straight forward to go through turning them green, one after another until they are all green (A1)* Also, the improved support and documentation in later versions of the components: *Usually the APIs are clearer. ... The older versions of documentation are now extinct (A3)*

Blocker	Motivator
More generally, the additional work involved in implementing and prioritising security enhancements: *[Only] a certain amount of our road map time is given to architecture. And we have ended up diverting most of that time to addressing security vulnerabilities in one way or another, since you first came. And there is still more to do. The downside of that is, obviously, that we don't address other architectural concerns like performance, or code quality. (A1)*	The benefits of security as a feature, whether tick box support for the audits of potential customers, or actual unique selling propositions when compared with others. *I think it has come at just the right time for us, because … the world is moving forward in terms of expectations around security… [and] we are getting more customers for whom security is a bigger concern (A1)*
The difficulty of learning from existing security sources. *I still find reading the OWASP stuff difficult. (A3)*	Learning as a group. *We've adopted this idea of focussing on a particular one of the OWASP Top Ten each release. I think that went pretty well in the first release. (A1)*
Certain security services not being available to a target user base. *We could use … Facebook logins… but it could be blocked by the firewall by proxy … and a lot of our target audience is people in colleges and schools, who wouldn't necessarily be able to get to that. (Participants, B Threat Assessment)*	Alternative providers *No college … would block access to Google… And … you can set up a Google account with any email address as well. (Participants, B Threat Assessment)*
Unhelpful company policy on tool provision. *So AlertLogic is your classic enterprise solution for something like this, where … it takes 5% of all the resources on the machine, and is really hard to set up, doesn't work when containers are involved, and takes six months to roll out a patch'... So, I was like 'why don't we use something new' and [management] were like 'nooo'... 'We have already paid for this other one'! (Participant, C Follow-up)*	Using additional tools to the company-specified ones. *We ended up using two: an open source one, Falco, which sends us slack alerts if it detects any weirdness on [our microservices architecture], and Alert Logic. (Participant, C Follow-up)*

Blocker	Motivator
Friction with other departments over security issues. *So, people would end up doing the work, and then having to get sign off from the Security Team after it, or having to make changes to it, and everything else! So, between Development and Security Team there is a lot of friction which obviously doesn't help. (C1)*	Proactive interaction over security issues: *[In my new company] a member of the Security Team is involved, when we are planning the development work, and then after we have deployed it, it goes into our [release testing] environment, and the Security Team have a week with it to test it, from their side, to check there is no vulnerabilities. (C1)*

7.6.2 Categorising Blockers and Motivators

The blockers and motivators fit broadly into the following categories: organisational aspects, supporting tools and the product/business themselves. Figure 41 shows to what extent each was referenced by participants from each company: blockers are shown to the left; motivators to the right. The following Sections 7.6.3 to 7.6.5 explore each category.

7.6.3 Organisational Blockers and Motivators

Under organisational aspects we found blockers in management issues such as no clear ownership of security in the organisation, and time and workload management. This is essentially the key scarce resource in organisations, and poor management of employees' time and workload will override any personal, positive factors [152]. Participants from Team C described a dysfunctional relationship with a security team that was required to sign off on products but gave no guidance to developers and was not approachable for

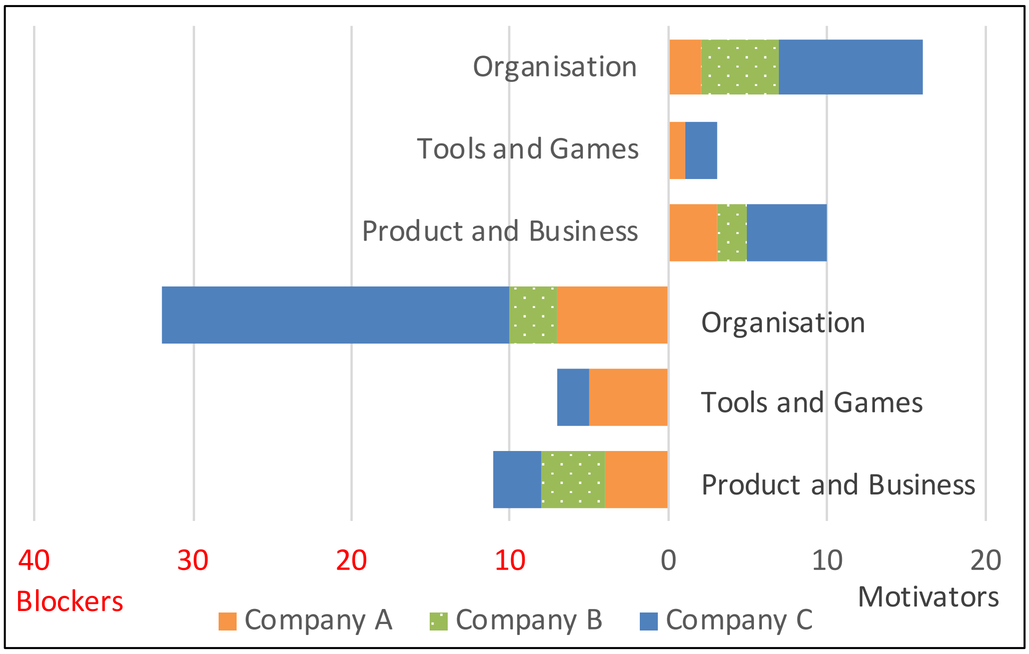

Figure 41: Mentions of Blockers and Motivators, by Company

help. Their security team apparently practiced an internal 'security through obscurity' approach, which makes learning from security issues difficult for developers:

> *It was almost as if this information was kept confidential, on a need to know basis, and unfortunately it means that [development] teams will find it difficult to learn from the event. (C2)*

This is reflected in the substantial number of organisation blockers identified by Team C in Figure 41. Two participants from Team A also noted that while the security education received in the organisation was interesting and helpful, it was frustrating that there was no space for reflecting on it or practicing it when developing.

At the same time, management can also provide a motivator. In Team B security aspects were integrated into the development processes and acknowledged in planning:

> *"We needed to put it into our procedures, not just into our thoughts, but into our ... you know, 'this is the way we work; this is what we do'. This is what I have got out of it." (B1).*

One participant in Team C reiterated this point by considering holistic thinking about the product to generate security understanding and motivation. Another participant suggested that security targets should be part of performance indicators for employees in order to motivate work on security.

7.6.4 Supporting Tools as Blockers and Motivators

In terms of supporting tools, Teams A and C used a large number of tools, games and procedures to support their secure software development processes. A few of these approaches were abandoned due to their poor design: one game in Team A caused embarrassment to individuals, and made employees feel uncomfortable (see Section 7.5.1). Two participants noted the complexity of their infrastructures and the difficulty of integrating and configuring off-the-shelf security solutions, especially when legacy software is involved. In particular, encryption, key management and cloud computing platforms were mentioned as aspects where achieving security was unreasonably difficult.

Yet at the same time, outside influences were also perceived as motivators for security. In Team C, compliance checks, certification and recent relevant legislation have all caused an increased interest in security in the organisation, and this has driven security improvements. The participants also mentioned changes in architectures as motivators for security improvement, as in the participant's opinion improved features and improved security often co-occur. The developers are also keen to release their code to the public, motivating a greater focus on security:

> *Our problem, I think, here is that we have a tendency to want to make our code repositories public, as a means of helping the wider world. The problem with that is that you automatically put yourself in a vulnerable position. (C2)*

7.6.5 Business Function Blockers and Motivators

The third category of issues centre on the team's business function. Team A noted that customer's security policies and requests for customisation of the product are significant barriers to maintaining secure code and good policies. In Team C security was reported to be difficult to sell. Yet in Team A security customers were actively requesting security, making them a motivator for secure software development. The recent increase in news coverage on security is also seen as a motivator in both companies:

> *Some of it could be good old-fashioned scaremongering due to what has happened in the press, but if that is what works, then fine, we'll take that. Because the reality is, it was the stuff that needed to be done." (C2)*

7.6.6 Tension Between Blockers and Motivators

All three organisations had motivators in the categories where blockers were present. But these motivators where not created in response to the blockers, but rather as independent encouragements for secure software development.

The implication for development teams is the need both to encourage the motivators, and to resolve the blockers. Since most were outside the immediate control of the developers, this is an organisational, rather than a developer, opportunity for improvement.

7.7 Discussion and Next Steps

This section returns to the research question:

RQ 4 *What security outcomes did the 'Developer Security Essentials' package have, and what aspects contributed most to those outcomes?*

It explores how the answers to RQ 4.1, RQ 4.2 and RQ 4.3 in the previous sections provide a basis for the next stage of the research.

7.7.1 Estimate of Impact

While it is early to know the long-term impact, it is unlikely that that Company A's teams will remove their component security evaluation and upgrade process, nor lose the benefit of the permissions improvements. Whether the remaining security enhancements discussed in the first follow-up session get incorporated is, correctly, a business rather than a technical decision; it is promising that the architects are asking the right questions (about risk evaluation) to support business in making that decision.

Similarly, we know from feedback (Section 7.5.2) that Company B's project leaders now include security in their project planning, instructions for students, and general company management; and their students are more aware of security issues.

Some key learning points, especially that security is more than avoiding the OWASP Top Ten in your code, will remain with all the participants from Companies A and B.

In the case of Company C, the most we can say is that the interventions contributed to an existing trend of increased security awareness.

7.7.2 How the Interventions Worked

The impact of the interventions differed between teams: not only in the nature of the security issues addressed; but also, in the teams' responses to the interventions and in how they benefitted. Team A introduced better development processes; Team B gained an awareness of several specific security improvements and the need for Threat Assessment; and for Team C the interventions prompted better communication and understanding. Sections 7.4.1 to 7.4.3 explored the differences in the ways the teams responded to these interventions.

The successes identified came through the developers' choices. As the Expert Survey concluded (see Section 4.14), to be effective a program needs to motivate rather than simply direct the teams involved. And, indeed, the interventions were successful to the extent that they could change the developers' thinking, understanding and motivation. The interventions involved, predominantly, conversations between developers, allowing them to learn mainly from each other, and to motivate themselves rather than respond to outside pressures. Table 15 and Table 14 suggest that this was an effective motivation and learning approach.

Indeed, by contrast to the results of earlier studies based on interventions using Penetration Testing as a motivator [134,172], in this study Table 15 shows that for both Teams A and B the long term impact after one year was still important.

7.7.2.1 Interaction with Other Stakeholders

The analysis of blockers and motivators identified during the interventions (section 7.6) found that a large majority of both involved interactions with either the business function or other aspects of the organisation. It follows to improve development security it will help to work explicitly on these interactions.

Of the key assurance techniques identified by the expert survey (Section 4.4), two are related to such interactions: Product Negotiation, and to a lesser extent, Threat Assessment. We conclude that exploring ways to enhance these two techniques by addressing blockers and encouraging motivators has the potential to deliver further improvements.

7.7.2.2 Learning Points for Developers

Table 17 highlights three learning points for software developers from the above discussions: the effectiveness of team activities, key assurance techniques, and the importance of organisational issues.

7.7.3 Impact of the Active Developer Model

Several aspects of the interventions suggested the effectiveness of the Active Developer Model, as follows.

Company A discovered security and privacy threats and acted as 'customers' for security enhancements independently of their company's security specialists (Section 7.4.1), only becoming demotivated when they were disempowered from carrying out the mitigations they suggested (Section 7.5.1).

Company B lacked security specialists to provide an alternative to the Active Developer Model; yet participants B1 and B2 adopted security principles enthusiastically (Section 7.4.2), and the discovery of B4's aptitude for software security thinking even led him to a career in the area (Section 7.5.2).

Though the impact of Developer Security Essentials in Company C was limited, the fact that their software security practices were effective was itself a vindication of the Active Developer Model; since their relationship with the security specialists in the company were dysfunctional (Section 7.6.1), the success of the software security can only be attributed to being driven by the developers.

Table 17: Advice for Software Development Teams

Apply Team Activities to Security	All the workshops derive their effectiveness more from discussions between participants than from any information provided by the intervener, and as Section shows the nature of these discussions was different for each team. So, the success of these interventions can be attributed to the team nature of the activities, and on the participants bringing their own unique range of expertise and knowledge to them. Whether or not a given team uses the specific workshops described here, we conclude that there is benefit in regarding software security as a team, as much as an individual developer, process.
Focus on Key Assurance Techniques	In an example of the Pareto Principle, that 80% of the benefit often derives from 20% of the input [42], the results of this chapter show that introducing three assurance techniques that are within the scope of most development teams, out of out of twenty in use by industry, are together capable of delivering a large impact. We conclude that teams will benefit from concentrating first on these techniques, namely **Threat Assessment**, **Automated Static Analysis**, and **Configuration Review**.
Address Organizational Issues	As Figure 41 shows, some 40% of mentions of issues by participants, both of Motivators supporting security improvement, and Blockers discouraging it, are ascribable to organizational issues. Whilst this finding will not surprise any security professional, it emphasizes the need to regard the promotion of software development security as a systemic, rather than purely a development team, matter.

7.7.4 Future Work on Assurance Techniques

The importance suggested by Section 4.4 of a small number of assurance techniques provides an incentive for further research on those techniques. While Automated Static Analysis and Penetration Testing have received a good deal of research attention, participant comments, especially Blockers and Motivators, suggest areas for inquiry for others:

Threat Assessment	Participants requested example expert assessments for different domains and types of software, to act as a basis for their own assessments.
Configuration Review	Several Blockers suggest a need for improvements to tools and to vulnerability databases to support more fine-grained component analysis.
Code Review	Traditional line-by-line code review may not be optimal for security issues: one participant, for example, described instead asking developers to show in their code how they addressed specific security issues. There is a need for experimentation investigating the merits of different approaches.

Product Negotiation	Participants requested methods to express specific security improvements as organisation benefits; and ways to identify the probability of different security breaches. From Section 7.7.2.1, we also conclude a need also to find ways to identify and address blockers and motivators during the process.
Incentivisation Session	Alternatives to the Agile Security Game include Capture-the-flag games, Penetration Test-based sessions, and case study-based training. While this work proves the success of the first, research would be valuable to compare other approaches in differing situations.
On-the-job Training	The interventions provide only a one-off security improvement. Games such as the 'covert saboteur' in Team A offer opportunities for developers to develop their skills further. However, as we saw, the effectiveness of such approaches depends on personal aspects and team dynamics (Section 6.2.1). Research is needed to provide low-time-cost ways to continue the team security improvement process.

7.7.5 Viral Distribution?

For the interventions to have a longer-term impact, we wanted participants with the intervention work not only to improve their own projects, but also to understand how to take the interventions and use them themselves in other contexts.

As discussed in Section 7.4.4, the results were generally disappointing, though the leaders of Teams A and B did show understanding of Threat Assessment and Product Negotiation.

7.7.6 Next Steps

We identified three key areas for future work on the interventions. First, the participant-driven nature of the workshops meant that not every technique was covered for every team: Team B did not discuss Automated Static Analysis, Penetration Testing, nor Code Review, for example. One participant suggested a checklist or take-away sheet after the first day's presentation:

I think maybe some sort of tick sheet in terms of "have you got these things in place?" to take away, that might be a good addition (A1).

Second, for the program to scale to a wider number of participant teams, we needed intervention leaders who appreciated the aims of the different sessions, such as the importance of an Incentivisation Session to achieve team motivation. Yet Table 14 suggests that this knowledge was not successfully conveyed to many of the participants. Nor did any participants learn how to use the Developer Security Essentials intervention themselves. We identified a need to motivate and teach some participants to lead the intervention.

Third, an important area for improvement was in Product Negotiation: both in methods to express specific security improvements as organisation benefits (Section 7.7.4) and to gain the time and 'mind-space' to use effectively the security learning the team had gained (Section 7.6.3). We considered that an extended workshop might be a suitable way to support developers in doing this.

In terms of the methodology used for the trials, we identified two improvements to make:

1. There were too few companies involved for us to deduce much about the effects of different contexts on the effectiveness of the intervention. Involving more would give more scope for such deductions.
2. The changes resulting from the interventions were discovered at haphazard; the analysis approach derived example improvements but did not necessarily provide consistency between companies. For further work, we wanted a more consistent analysis of the understanding gained and techniques implemented.

These improvements were incorporated into the next project.

8 Further Trials (Magid 2)

This chapter describes a further project, Magid 2, involving a second round of trials with an improved Developer Security Essentials package.

This project reflected an emphasis on making the intervention potentially scalable for use by large numbers of organisations. Its research question was therefore:

RQ 5 *Which aspects of the 'Developer Security Essentials' intervention are effective at improving security when used independently by teams from a variety of cultures and different types of organisation, and why?*

The three most salient changes from the previous project are:

Independent Facilitators: We trained one or two facilitators from each organisation, and they then managed the intervention. The purpose of this was to transfer the ability and confidence to run the workshops to the facilitators, potentially allowing them to run further workshops without the support of the research team.

Variety of Cultures and Types of Organisation: In this project we carried out the interventions with many more groups: eight as compared with three in the previous project. In addition to the aspects of organisation, such as size and security capability, we also identified organisation culture types for each group.

Support for Product Negotiation: A further 'Threat Sales' workshop was added after the 'Threat Assessment' workshop, to help developers in representing security and privacy issues as business factors.

Section 8.4 gives more details.

8.1 Choice of Methodology

Whilst Canonical Action Research (CAR) had worked well as a basis for the previous cycle of trials, in attempting to use it for a second cycle we came across a 'showstopper': CAR focusses on the changes to the 'clients', and does not support changes to the list of clients. Specifically, in CAR, the invariant across cycles is the client (occasionally, a list of clients). Here the invariant was different: we were using the same *intervention* with a different *set of clients*. CAR could not handle the Magid 2 project.

While the techniques of CAR—recording the interactions involved, double coding them and analysing the results—still remain suitable for evaluating the intervention, the philosophy and overview guidance do not.

Accordingly, we moved to using Design-Based Research with CAR techniques, as described in Section 3.6. Sections 3.6.3 to 3.6.5 describe the practical aspects of doing so to support designing an intervention.

8.1.1 Ethical Issues

We obtained Ethics agreement for this project from the Lancaster University Faculty of Science and Technology Research Ethics team approval as a continuation project from the previous one.

Two of the companies involved required the deletion of transcripts and audio at the end of the research project (for confidentiality reasons). Because the coding record within NVivo itself has possible future research value, we want to do this without destroying the codes assigned. We therefore plan to extract and anonymise any quotations, delete the relevant audio files, and overwrite all non-space characters in the corresponding transcripts in the NVivo tool with 'x's to preserve the coding while satisfying the companies' requirements.

8.1.2 Data Analysis in the Magid 2 Project

Section 3.6.4 describes the generic approach we used to evaluate the Developer Security Essentials intervention; and Section 3.6.5 describes the evaluation of that approach. The 'Aspects' were assurance-related techniques, such as 'Pen testing'. We started with the list from the Expert Survey (4.4) and added further techniques as they were mentioned in the interviews.

The 'Levels' were the degree of engagement for each technique as follows:

No mention	The technique was not mentioned (though it might have been known or used).
Aware	Knowledge and understanding showed of the technique.
Planned	Existing plans to incorporate the technique into development.
Using	The team have used the technique.
Established	The team use (or consider for use) the technique in each new project they take on.

None of the participants nor teams indicated in any of the discussions that any of the assurance techniques were dropped or used less as a result of the interventions—even though if this had happened it could have been a positive result, representing better using of resources. We therefore concluded that in cases where the After engagement value was apparently less than the Before one, this only meant that the interviewees had not happen to discuss that level of engagement as much in the After interviews[31]; and therefore that

[31] In this project, this happened in less than half a dozen cases out of over a hundred, and in each case only by one level of engagement (3 rather than 4, for example).

it would be misleading to show it as a decrease in engagement. We therefore removed such decreases by increasing those After engagement values to the corresponding Before values.

As in the previous project (see Section 7.1.2), we were careful in coding to distinguish new assurance technique engagement attributable to the interventions, from engagement due to other external factors such as customer demand or other security specialists. We did not code these latter increases in engagement levels at all.

8.2 Participating Developer Groups

The eight different organisations were selected opportunistically. As in the previous project, the individual members we interviewed are identified using the team letter and a number: 'D1'. However, in this case not all of the organisations involved are companies, and in many cases the workshop participants came from several different teams. In this discussion, therefore, we refer to the set of participants in a single Developer Security Essentials intervention as a 'group'. We use an identifying letter for each group, and the sequence of letters continues from the previous project, hence are D – K.

Several recruitment methods were used, as follows:

University: used the support of the Lancaster University 'Academic Centre of Excellence in Cyber Security Research' (ACE-CSR) team with its connections to government, and the Lancaster School of Computing and Communications Business Engagement Team and their contacts with companies near Lancaster.

Personal: used business contacts developed by the author in previous work in industry.

Conference: used leads from running the 'Agile App Security Game' (section **7.3.2**) as a workshop at several UK developer conferences: AgileNorth, Agile Cambridge and Agile Manchester.

As in the previous project, a selection of the participants was interviewed, including the facilitator and usually a team leader, both before and after the interventions.

All of the developers interviewed were male, as were all managers and testers; only the three product managers were female.

From the previous project with teams A-C, it was known that the culture of the teams would have a definite impact on the nature of their interaction in workshops. Accordingly, we have added a categorisation of the organisation culture. This was a subjective estimate by the author based on the team's behaviour during the workshops, using a taxonomy defined by Charles Handy [80] to describe the power of individuals' roles and functions within an organisation. Table 18 summarises that taxonomy; it derives from the Provenmodels website [116].

8.2.1 Summary of Participating Groups

Table 19 summarises the groups, showing the organisation sizes (Small for less than 100, Large for greater than 5000); the culture of each team as discussed in the previous sections. It indicates the number of participants, the way in which the team was recruited. It also shows 'Security Maturity', an estimate of their 'secure software capability

Table 18: Organisation Cultures (after Provenmodels [116])

Zeus	Power is concentrated in the hands of one individual, the top boss. Control radiates from the centre's use of personal contacts over procedures. The most powerful person dominates the decision making process.
Apollo	A strong role culture places a premium on order and efficiency. Power is hierarchical and clearly defined in the company's job descriptions. Decision making occurs at the top of the bureaucracy.
Athena	Power is derived from the expertise required to complete a task or project. The work, itself, is the leading principle of coordination. Decision making occurs through meritocracies. Employees move frequently from one project or group to another.
Dionysius	Organisations exist for individuals to achieve their goals. Employees see themselves as independent professionals who have temporarily lent their services or skills to the organisation. Management is considered an unnecessary counterweight and given the lowest status. Decision making occurs by consent of the professionals.

maturity' [90] by the author based on public information about the organisations and the groups' discussions during the workshops, as follows:

Low:	Little or no awareness or activity related to software security
Medium:	Aware of and addressing security issues, typically including some developers with good security knowledge.
High:	Experts at software security, within an organisational culture that assigns it a high priority.

For each intervention, the group designated one or two 'facilitators' (D1, E1, etc), and it was these facilitators who arranged and led the interventions. Since the skills and aptitudes of these facilitators were likely to be important, the table shows their roles and indicates, by the use of the plural, how many were involved.

The table shows how the different means of recruitment introduced groups with corresponding cultures. The author's prior contacts with companies tended to be with the CEOs of Small to Medium Companies (SMEs); the workshops would only happen where such CEOs have influence over technical training for their teams; accordingly, we find the groups recruited tended to be Zeusian in nature. Similarly, the conferences which led to recruitment were in the 'Agile Cambridge', 'Agile Manchester' series: conferences targeted at senior technical development professionals, particularly 'scrum masters': a

Table 19: Participating Developer Groups

	Org. size	Culture	Security Maturity	Size	Facilitator(s)	Recruited
D	Large	Athena	Low	10	Managers	University
E	Large	Apollo	High	12	Security specialist	University
F	Small	Zeus	Low	3	Manager	University
G	Medium	Zeus	Med-High	10	Managers	Personal
H	Small	Zeus	Medium	8	Developer	Personal
I	Medium	Athena	Medium	14	Manager + Developer	Conference
J	Large	Apollo	High	14	Security specialists	Conference
K	Medium	Athena	Medium	16	Developers	Conference

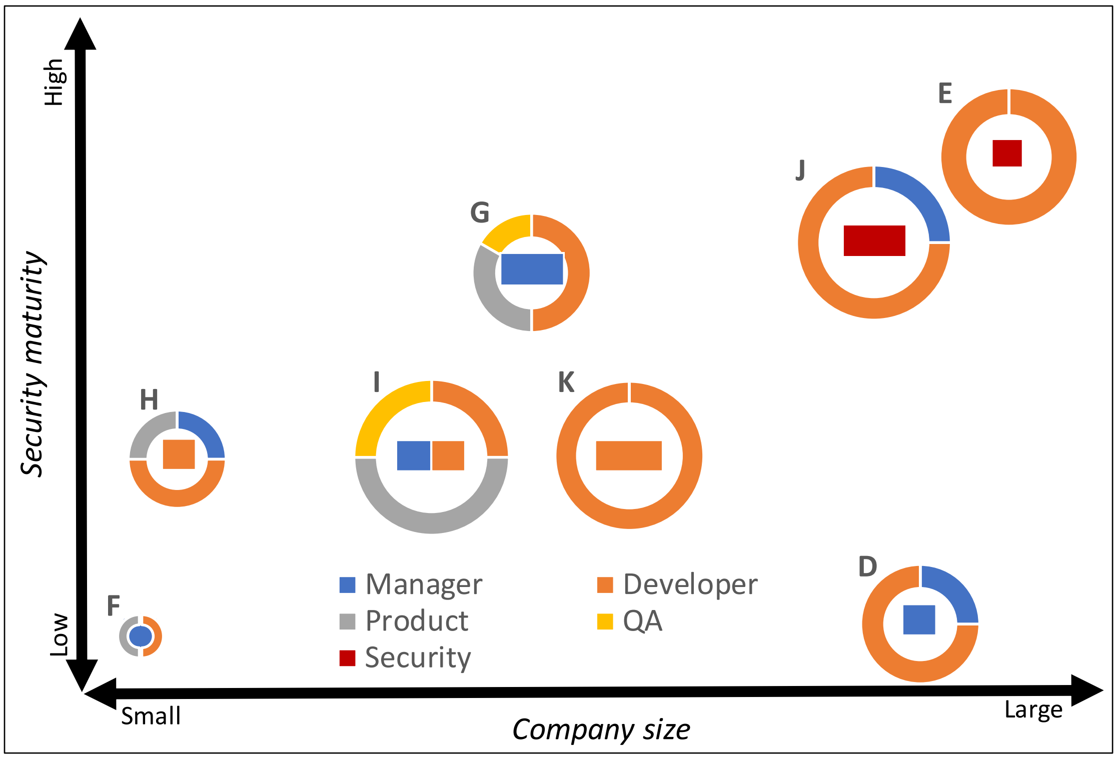

Figure 42: Composition of the Participating Groups

role involving the facilitation of software developers. We speculate that such professionals tend to favour Athenian cultures, since these give experts the most power in structured organisations (see Table 18); though Group J turned out to be marginally Apollonian rather than Athenian.

Figure 42 visualises some of the information from Table 19, adding the roles and numbers of interviewees for each workshop. Each group is represented by a ring, with an area proportional to the number of participants. The colours of segments in the ring indicate interviewee roles. The leaders and their roles are indicated by the squares (for a single leader) or rectangles (for two leaders) inside each ring; the colours show their normal roles. The locations of the rings indicate the size of the organisation involved (x axis) and the Security Maturity of the teams prior to the interventions (y axis).

Recall that we interviewed only a subset of the participants in each group. The colours of the ring segments represent the interviewee roles, not the overall participant roles. Since, however, we requested a representative sample of participants for the interviews, it is probable that the proportions of roles of the participants are similar to those shown here.

The visualisation shows the wide range of group sizes, organisation sizes and security maturity involved in the project. Unsurprisingly, only groups in large, security-adept companies had access to security professionals, and thus only groups E and J had these as facilitators. We had requested that product managers and testers join the groups; as Figure 42 shows, this happened only in the smaller companies, possibly because in small companies product managers and testers are closer to the development team, and more amenable to taking part in group activities. The smaller companies F and H didn’t have a separate QA function; so, as shown, only the medium sized companies G and I had testers join the workshops[32].

8.3 Research Sub-Questions

As discussed in Chapter 3, Section 3.6, the method used for the research was Design-Based Research. Specifically, we used DBR for the overall philosophy, and used the data gathering and analysis techniques of Canonical Action Research (Section 3.5).

The two cycles of the DBR method (Figure 9 in Section 3.6.2) require separate research questions for the Design Practice cycle and the Design Theory cycle, as discussed in the following sections.

8.3.1 Design Practice Questions

The Design Practice cycle in DBR requires measurement of the effectiveness of the artefact—in our case, the intervention:

RQ 5.1 *To what extent did the groups learn about and adopt different security-enhancing activities as a result of the Developer Security Essentials intervention?*

We also wanted an indication for which kinds of teams the intervention is likely to be most useful. Though the small sample size precludes any statistical validity, such an indication is better than nothing to suggest where to focus effort in encouraging teams to adopt the intervention:

RQ 5.2 *How does the impact of the intervention vary with different company sizes, team cultures, facilitation styles, security expertise, and kinds of participants?*

Specifically, one aspect we can influence, even if not fully control, is the facilitation style used with the intervention in future:

RQ 5.3 *How does the impact of the intervention vary with different facilitation styles?*

8.3.2 Design Theory Questions

Second, the DBS Design Theory cycle requires theoretical hypotheses to test. The previous project had taught us that improving Product Negotiation between developers and Product Management was important for improving security (Section 7.7.6). Returning to the positive approach derived from Appreciative Inquiry that had been successful in the Expert Survey (Section 4.1), we concluded that identifying the benefits to the organisation of mitigating security and privacy issues might be a good way to do this.

As researchers, we were not in a position to quantify any actual resulting security improvements, but we could establish whether process improvements were made or additional resources allocated. This gives a research question as follows:

RQ 5.4 *Can having developers consider the positive benefits of security and privacy mitigations lead to security improvements in the development process?*

[32] Company K did not mention a separate QA function. We believe much of their testing was through automated tests created by developers.

In order to identify the positive benefits of security and privacy mitigations, we needed participants to do two things:

- Estimate the risk and impact associated with each identified threat
- Identify positive benefits to the organisation (or customers) from mitigating the major threats.

Originally we considered teaching participants how to estimate risk and impact; and approaches to identifying such benefits, possibly using examples in a 'patterns' form [68]; but then we wondered if developers might be able to do both tasks without specific teaching. The corresponding hypothesis, then, was:

> *Teams of developers can produce both adequate risk and impact assessments and benefit analyses with minimal guidance.*

The research question, therefore, was simply, whether this is true:

RQ 5.5 *Can teams of developers produce both adequate risk and impact assessments and benefit analyses with minimal guidance?*

8.4 Method Implementation

8.4.1 Changes to the Intervention Package

From the previous round of trials, we had identified changes to make (Section 7.7.6):

1. A checklist or take-away sheet after the first day's presentation;
2. Intervention leaders to appreciate the aims of the different sessions;
3. Participants to learn to facilitate the sessions;
4. Improvement in Product Negotiation to express specific security improvements as organisation benefits and to gain the time and 'mind-space' to use effectively the security learning the team had gained;
5. Involving more companies; and
6. Improved analysis approach to provide consistency between companies, analysing the understanding gained and techniques implemented.

To achieve this, the major changes to the Intervention Package, Developer Security Essentials, were:

1. An initial training session for the facilitators by the researchers.
2. Improvements to the instructions for the Agile Security Game, so it could be set up and run without researchers present.
3. A checklist of assurance techniques for discussion towards the end of the main workshops.
4. A further 'Threat Sales' workshop after the 'Threat Assessment' workshop. In this the facilitators asked the participants to take the three or four highest priority issues and justify them in terms of positive impact for the organisation.

8.4.2 Threat Sales Workshop

In this workshop participants split into groups, and each group addressed a different threat from the most important five or so identified in the previous workshop (Section 7.1.1).

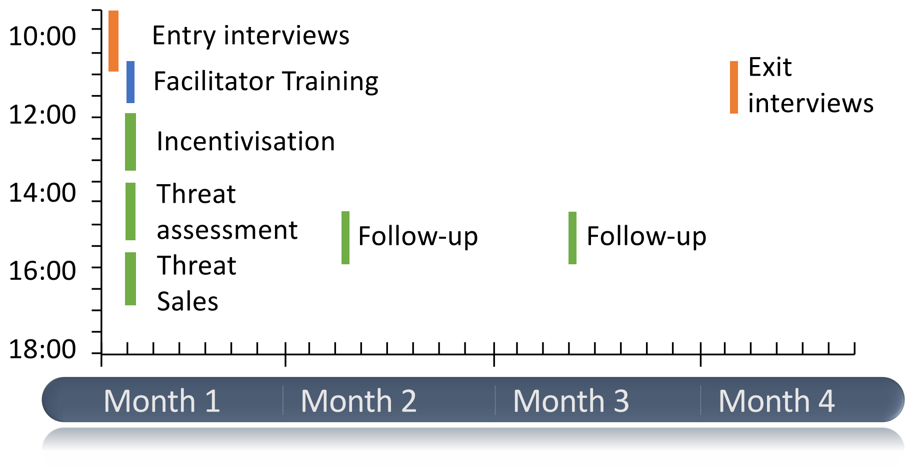

Figure 43: Intervention Timeline

We made the task approachable for developers by avoiding discussion of 'sales' or 'persuasion', two activities that developers tend to avoid[33]. Instead the participants were split into groups, each looking at one instruction for the participants was:

> *Choose one of the threats and work out positive ways in which addressing that threat will benefit the organisation.*

The thinking behind this was straightforward. Product Management professionals deal predominantly with competing enhancements to products; their mental map is less geared to handling risks to the business (which is a senior management role). So, for security improvements to be assessed by product managers, they must be expressed in terms of benefits.

Each group discussed the threat they had chosen and wrote notes on a whiteboard or flipchart page. A representative from each group then presented their conclusions to the other participants.

8.4.3 Practical Approach

The sessions were similar in structure to the previous round of interventions (Section 7.1.1). The entry and exit interview protocols were the same (Appendix F, Appendix G)

After the entry interviews, however, we added an extra session of 30 – 60 minutes with one or two people from the organisations involved who were prepared to learn to be facilitators. In that, we discussed the motivation for each of the Incentivisation Session, the Threat Assessment and the Threat Sales workshops, and instructed the facilitators how to lead each one.

The Threat Sales workshop followed the Threat Assessment workshop after a suitable (typically half hour) break.

Figure 43 shows a typical timeline for the intervention. The 'Facilitator Training' session also included entry interviews for the facilitators.

[33] The author spent 10 years at Penrillian trying to get professional developers to do sales activities; few were willing despite pay and other incentives.

Table 20: Inter-Rater Reliability Results

Description	Metric (All cases)	Metric (Active ratings)
Initial Krippendorff's Alpha	0.67	0.18
Krippendorff's Alpha after Discussion	0.73	0.46

8.5 Research Numbers

The eight interventions generated 21 hours of interviews, which were transcribed manually; and 47 hours of training, workshop and follow-up sessions, which were transcribed mechanically using Sonix. The final code book contained 2859 references to 51 codes.

The Inter-rater Reliability metrics, calculated as discussed in Section 3.6.5, were as shown on the first row of Table 20. The initial 'Active ratings' metric of 0.18 indicate only slight agreement. On discussion and investigation, the coders identified several causes for this:

- In the Before interviews, and some of the After ones, there was deliberately no attempt to ask interviewees about their use of assurance techniques, in order to avoid bias in the responses. This meant that each coder had to carefully interpret what was said for evidence of Assurance Techniques, and different interpretations were often possible.
- The most common cause of discrepancies was one or other coder overlooking a single indication of Assurance Technique use. For example, an offhand comment by an interviewee while discussing something else, *"…Which goes in the standard tool chain after Fortify…"*, might be picked up by one coder as Incorporation for Static Analysis Tools, and not by the other coder who would thus produce a rating of Not Mentioned for Static Analysis Tools
- There was a lack of shared understanding of some Assurance Technique definitions. For example, should an intention to repeat the workshop with the same team in a year be coded as 'Further Workshops'?

The coders addressed the last point above by agreeing more precise definitions for several Assurance Techniques; and addressed the first two points above by comparing coding on the cases where there was a substantial disagreement. They then independently amended their coding based on that discussion, giving the results shown on the second row of Table 20.

In the amended coding, the 'Active ratings' cases now show 'moderate' agreement. This is as good as can reasonably be expected, given that the issues identified above. On investigation, discrepancies appeared mostly to be assignable to one coder missing something, so the combination of the coding from both is a valid approximation to the ground truth.

8.6 Practical Results

Recall from Section 3.6.2 that Design-Based Research has two sets of outcomes: first the practical outcomes in terms of measurements of the effects of the artefact in the trials and deductions in terms of future improvements for the artefact; and secondly new theory derived from the trial data.

Table 21: Assurance Techniques

Vulnerability Finding	Process Improvements	Education
Automated Pen. Testing Automated Static Analysis Configuration Review Code Review Penetration Testing	Threat Assessment Product Negotiation Contingency Plan Security Champion Standardisation	On-the-job Training Further Workshops

This section explores the first set of outcomes: practical ones, addressing the first research sub-question:

RQ 5.1 *To what extent did the groups learn about and adopt different security-enhancing activities as a result of the Developer Security Essentials intervention?*

It describes first the resulting list of techniques based on the full analysis; then each of the groups undertaking the Developer Security Essentials intervention, and the results from each[34]. Each section introduces the organisation and teams involved and discusses the group's reasons for wanting the intervention. It then describes the intervention as it happened for that group, then provides the results of the analysis of the transcriptions of all of the sessions.

8.6.1 Resulting List of Techniques

Table 21 shows the full list of Techniques derived from the coding. It divides them into categories: Vulnerability Finding, techniques to find specific vulnerabilities in created software; Process Improvements, to create an environment to better support the creation of secure code or reduce the impact of security issues; and Education, to teach participants and stakeholders about the previous techniques. Most of the Techniques were introduced in Chapter 4, but the coders identified five further techniques for security improvement from the discussions and interviews, each used by several of the teams; these are highlighted. They also found that that one previously used Assurance Technique, Incentivisation Session, did not reflect the way participants discussed their secure development improvement; instead participants discussed Further Workshops. Incentivisation Session was therefore omitted.

The resulting full set of assurance techniques is described below:

Automated Penetration Testing	Using an automated tool to look for common, easily exploited, vulnerabilities in a website or web service.
Automated Static Analysis	Using automated tools to look for common vulnerabilities in source or binary code.
Configuration Review	Choosing secure components and frameworks, and keeping them up to date
Code Review	Scheduled meetings or pair programming to analyse code for security defects
Penetration Testing	Having a Security Specialist look for vulnerabilities accessible via the web.

[34] In other chapters the participant descriptions and results were separated. Given the large number of groups here, it is easier for the reader if they are kept together.

Threat Assessment	Design-level analysis of possible attackers, motives, and vulnerability locations (a.k.a. Threat Modelling).
Product Negotiation	Empowering product management to make security decisions.
Contingency Plan	The advance creation of a plan to handle security incidents.
Security Champion	Having a development team member, not usually a security expert, with a particular interest in security. They act as the go-to person for security issues within the development team.
Standardisation	The creation of standard security configurations, ways of working, or 'Secure Development Lifecycles', plus auditing processes to validate these.
On-the-job Training	Mentoring or informal workshops, used regularly with the development team
Further Workshops	Using the entire Developer Security Essentials package with other teams, or for the same team in a new project or with new members.

8.6.2 Group D Results

Group D are a project team within a university, funded by a government grant to promote business innovation. Group D's role is to developing proof of concept products and support applications, according to the requirements of specific businesses. The group had been in existence for only a year at the time of the intervention. Although members worked on several different projects at a time, all worked together as a team.

Unsurprisingly, the culture bears some resemblance to a start-up, but its situation within a university, the grant-based funding and perhaps the fact that the staff were on fixed-length contracts gives a more formal and disciplined feel to their ways of working. Though academic organisations are often Dionysian, within the team the culture appeared Athenian, with a respect for technical expertise and management experience.

The group were aware of the importance of software security but had little practical knowledge; hence their request for the workshops. There were two projects considered in the D Threat Assessment workshop. First was a data collection app to package grassland information for remote analysis by an agricultural specialist, their client. Second was an app to support managing personal and medical information for patients of a medical services company. Unusually, the Threat Assessment ideation session found almost no concerns with the first project, so in that session the team moved on to discuss the second project.

Figure 44 shows the levels of adoption of the various kinds of assurance technique (see Section 3.6.4) before and after the intervention.

Since the group's role was creating proof of concept applications, they realised that the actual implementation of security features and of security hardening was not important for them; what was important was that they supported their clients in implementing secure solutions when the clients came to implement the deployable applications. Thus the two techniques most important to team D were Threat Assessment (determining the security requirements) and Product Negotiation (conveying the analysis and importance of security to their clients).

Group D's projects tended to be relatively small, typically a few developer months, and so the three months of the intervention provided time for several new projects starting;

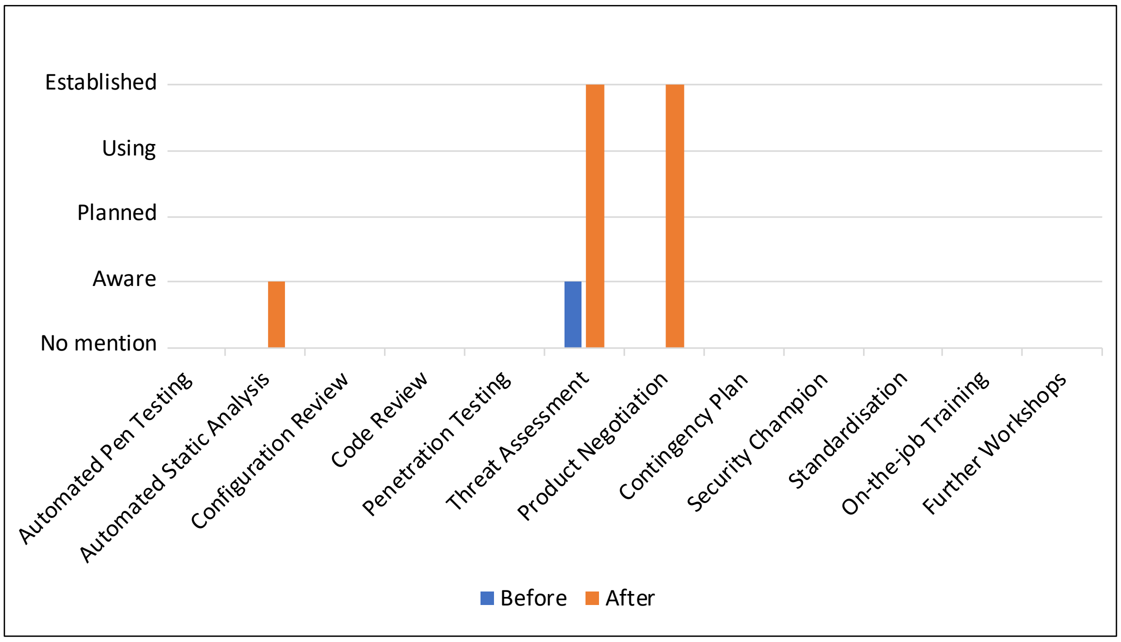

Figure 44: Group D Intervention Results

the 'Established' rating for those two assurance techniques reflects that both were incorporated into the new projects and into the group's process for further projects.

8.6.3 Group E Results

Group E work for a government department delivering software for sensitive government applications. The group was set up in the past couple of years, and many of its developers are less experienced than the average for the industry. The session leader E1, by contrast, was a highly experienced security specialist, now working on software development security. His role was to provide security support to a number of groups like group E, so his contribution to each project was relatively limited. E1 was keen to find ways to help the development teams he supported with security, and he requested the Developer Security Essentials Workshops accordingly.

Confidentiality restrictions limited what could be discussed in the presence of researchers. E1 chose this particular group because their product was a form of data store, and though it was designed to handle sensitive data, its functionality and design could be discussed without compromising confidentiality. Indeed, to avoid the need for security clearance for the researcher, the workshop took place in an insecure 'public' space: a meeting room in a serviced office building.

The group was divided into two teams, working on distinct aspects of the product; 15 people attended the workshop. The group culture was professional and fairly formal. Since the group's raison d'être was their ability to deliver a security product, E1's role gave him considerable authority. The group at the time had nobody in the product management role. The culture appeared Apollonian, with a clear sense of hierarchy.

The workshops showed a particular style of facilitation for the Threat Assessment session best described as 'directive'. The conversations were dominated by the facilitator and other participants tended to listen respectfully. This was a big contrast to the previous team, D, where everyone participated relatively equally.

Figure 45 shows the impact of the intervention. The largest difference between Before and After was in Product Negotiation: prior to the interventions, both development teams had regarded security as an absolute; every security feature and requirement had to be

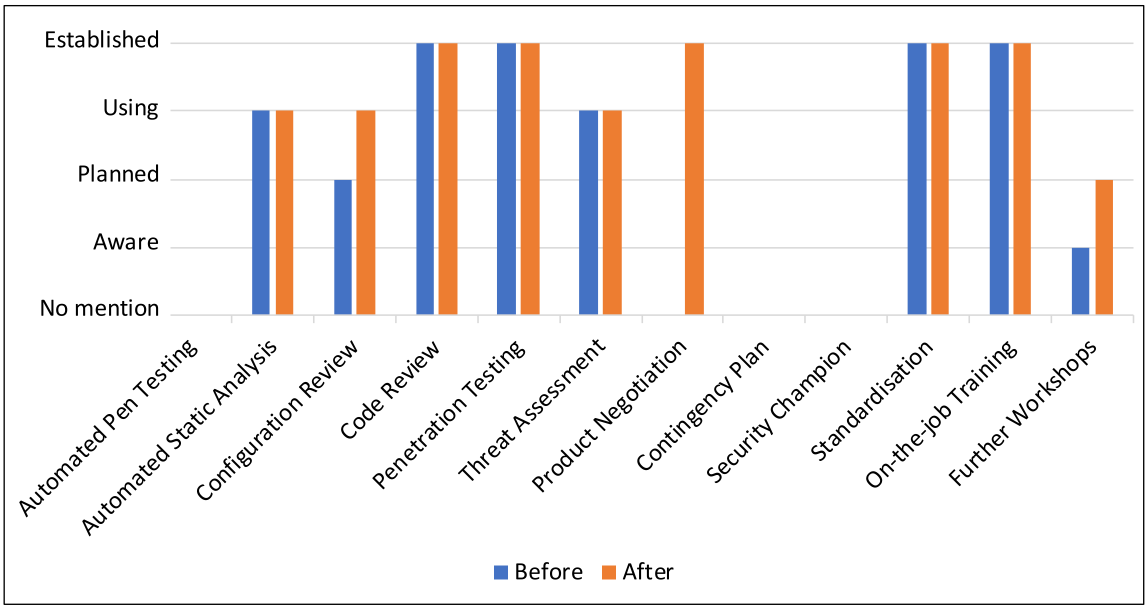

Figure 45: Group E Intervention Results

implemented without question. That attitude remained after the intervention; what changed was that the teams realised that some security aspects were needed *earlier* than others. The teams were using an agile development process, and now negotiated with their clients about the order of delivery of security features relative to other features.

E1 was pleased with the results of the intervention, and planned further uses of the workshop with other teams.

8.6.4 Group F Results

Group F were from a small surveying company that had created a specialist mapping product used by a large number of clients. The company's role is to carry out the mapping, store the maps in a Geographical Information System (GIS) database, and provide the database entry and reporting facilities for clients to store their data associated with the maps. They provide this through a web-based front end, and use servers hosted on their office site.

The company and product are long-established, and the company remains dominated by the Managing Director, who did not participate in the workshops. Group F have a manager, F1, with development skills, and a single full-time developer, F2. The product manager, F3, also took part in the workshops. The culture was fairly relaxed and informal, though it was clear that the managing director was involved in all significant decisions (a Zeusian culture). The team have been working together for several years, and seemed easy with each other.

Several years before, one of the developers involved with creating the web-based version of their product had had a doctorate in software security, and the underlying design and implementation reflected an understanding of security issues. However, none of the current team had any knowledge of software security. At the time of the workshops they anticipated a future sale to a government organisation, with possible contractual requirements for security.

Figure 46 shows the impact of the workshops. F1 took trouble to extract benefit from the discussions, creating a table of the threats identified from the Threat Assessment session

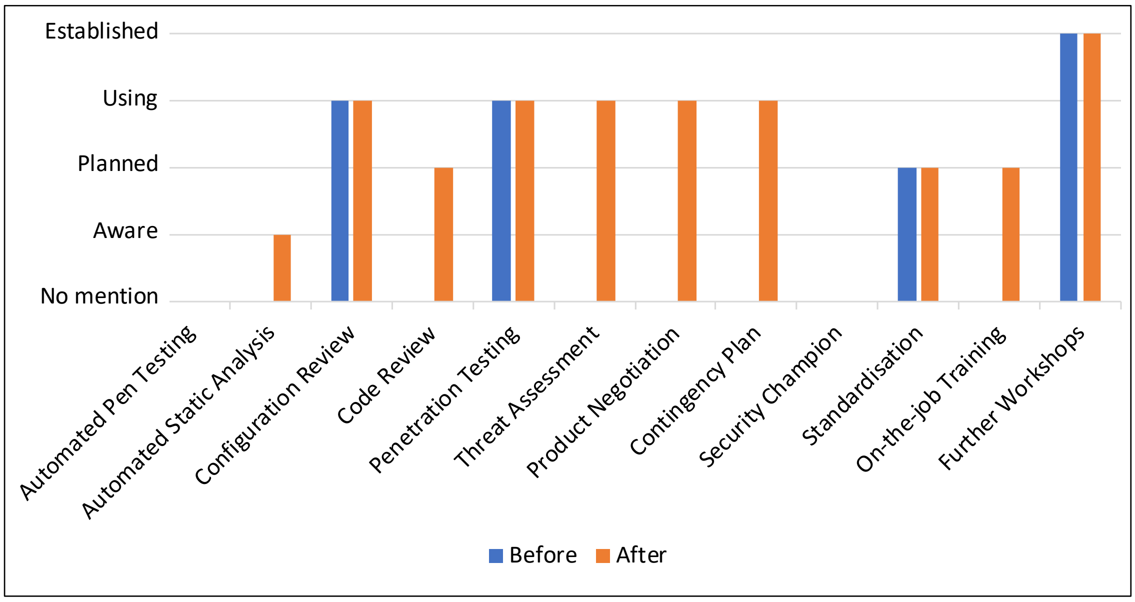

Figure 46: Group F Intervention Results

and returning to it in follow-up sessions. The Threat Assessment session identified several issues with the manual creation process of new customer accounts.

The government sale did occur, and the team used that Threat Assessment as a basis for Product Negotiation concerning which mitigations were now required for the new government customer and therefore got implemented; this Using of Product Negotiation is reflected in the figure.

8.6.5 Group G Results

Group G are from a web applications developer employing some 50-100 people, mainly development and product management staff. They produce a variety of applications for clients, ranging from stand-alone websites to front-ends supporting complex back end systems owned by the client. The two leads were G1 the group CTO and G2 the Development Manager.

The group came from several teams and included two product managers and a couple of Quality Assurance (QA) staff. Most of the participants were relatively experienced and correspondingly senior in the organisation. The CEO (not a participant) was still responsible for strategy and clearly involved in most aspects of the company: a Zeusian culture.

G1 and G2 had a particular reason for wanting the Developer Security Essentials Workshops. They were each personally expert in software security, and had a good knowledge of what was required in the websites their teams developed. However, increasingly they were finding that the effort their teams needed to ensure security was not reflected in the financial rewards the company received; neither was normally involved in pre-sale negotiations, and they had had recurring problems with being expected to provide costly security enhancements ‘for free’. They wanted to address this problem.

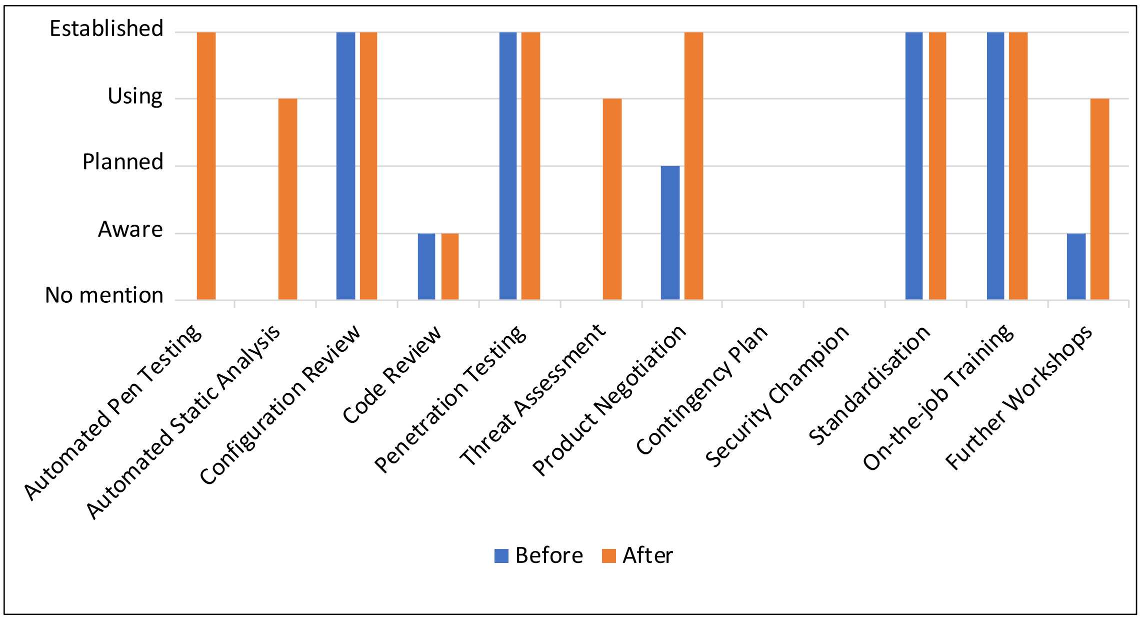

Figure 47: Group G Intervention Results

Figure 47 shows the impact of G's workshops. Given the problem being addressed, the Threat Assessment and Threat Sales workshops did not go into much technical detail.[35] Instead, they looked at ways to reflect security requirements through the pre-sales and contracts processes. They came out with an impressively simple way to discuss security cost-benefit with a client:

> *The problem here is that it is a difficult conversation to have once a client's signed off on a project… It should it should be done at the business development stage. People in Biz Dev should be coming to you .. and saying "Right, I've got this client on board. Here we go. There's three packages: our gold level hosting package, our bronze and our silver. Do you think they fit into any of these categories?" So, then we can go back to them as an additional add-on as we do with other little bits like maintenance … and just say "we've got these three packages and we reckon you fall into the bronze package" or "you fall into the silver package. It will be an extra blah blah blah, but we also do blah as standard and then these are the additional extras that you can buy…" [G Product Manager, Threat Sales Workshop]*

This idea of Gold, Silver and Bronze packages was taken up enthusiastically. G1 later expanded it to five options to include other aspects of security; at the time of the exit interviews he was creating a guidance sheet for the sales and marketing team.

8.6.6 Group H Results

Group H work for a small company selling a range of Internet of Things (IoT) devices and their associated infrastructure. Though the company has been established providing research services for some 30 years, it is only recently that they have moved to be a product-based company, so the company has some of the attributes of a successful start-up.

[35] Nevertheless, Automated Pen Testing and Static Analysis were introduced as a direct result of the workshops.

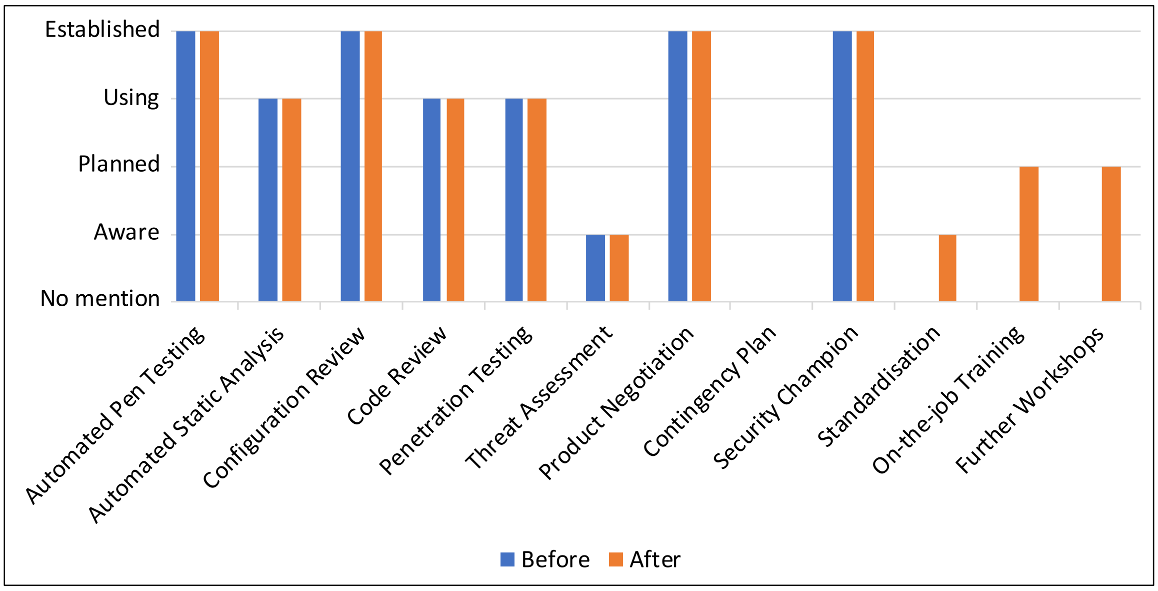

Figure 48: Group H Intervention Results

The group all work as a team; product management is effectively shared between H1 the CEO, who decides strategy (so another Zeusian culture), and H3 the CTO, who makes the day-to-day decisions on what functionality to implement. The workshop lead H2 was a developer with a particular interest in security.

The group justifiably considered themselves good at software security; the service they provide has a strong security component and security is part of their Unique Selling Proposition (USP) against Asian competitors capable of providing cheaper hardware. They viewed the workshops as a team-building style exercise to provide a measure of reward and entertainment to the participants.

Figure 48 shows the impact of the intervention for group H. They planned further training and possible workshops.

8.6.7 Group I Results

Group I are a well-established company providing the infrastructure required to allow a commodity trading market to function. The market uses internationally-agreed standards for the encryption and signing of trading messages, and this company provides systems with information on prices, bids and offers, and supports trades between users and with other participants in the market. It is a mature and respected company in its field, and the participants showed a relaxed confident attitude to development. Though they have considerable internal expertise in security, much of their security requirements have been satisfied in the past through perimeter security; only relatively trusted people have had access to the software. Now they have the possibility of delivering cloud-based services as well, where perimeter security will be less relevant.

Of the two workshop leaders, I1 has a largely managerial role; I2 has the most technical (and security) experience of the development teams in the company. Their purpose in running the workshops was to help developers and product management engage more effectively with security. Current versions of the product exist in a firewalled 'virtual community' where all systems owned by the trading companies are connected by VPN and are inaccessible from the wider Internet; future versions may be cloud-based.

The culture of the teams appeared relaxed and professional; there was no reference at any time to senior management, suggesting an Athenian culture.

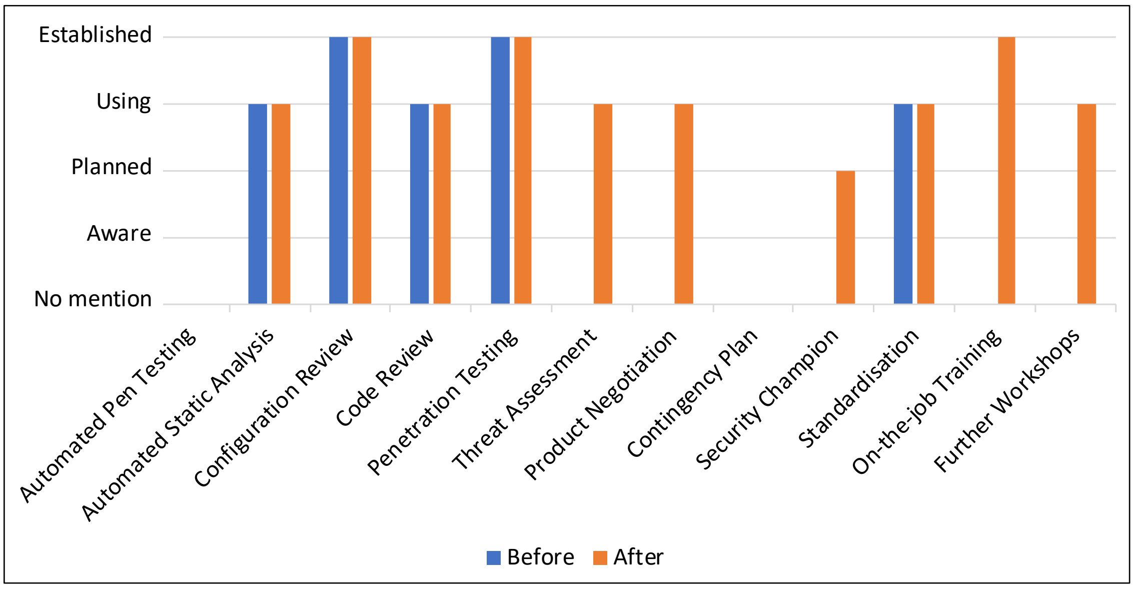

Figure 49: Group I Intervention Results

The group engaged enthusiastically with the workshops. Indeed, the day after the workshops at which the researchers were present they decided that the outcomes from the Threat Assessment and Security Sales were insufficient; they re-run both workshops to gain a more complete idea of the threats and possible impact on customers. They also ran the full suite of workshops separately with further development teams.

Figure 49 shows the impact of the workshops. As discussed above, they found value in the Threat Assessment and Product Negotiation, and both were incorporated for the future, along with further training and workshops.

8.6.8 Group J Results

Group J were from a well-established large company providing web interfaces for retailers. The particular group involved had the responsibility of creating tools and services to support deployment, and comprised about a dozen staff. The company has development sites in Eastern Europe, and a policy of moving capable staff around, so the group had more staff originating from outside the UK than any of the others involved in this project. Unlike any of the other organisations except Group E, this company has a separate security function employing professional security experts; it also has a policy of assigning the security experts directly to the teams, and of training developers as security experts. The culture of the group was serious, with little banter among the teams except in the breaks, and an awareness of hierarchy in terms of both management and technical experience. That suggests an Athenian or Apollonian culture; there seemed emphasis on management control, so this text assigns the culture as Apollonian.

The two assigned leaders for the workshops were both security specialists: J1 was the security specialist assigned to the group; J2 was more recently a developer and is in training as a security specialist. The group manager, who arranged our involvement, did not participate in the workshop, though J5, who manages one of the teams, did so. Their purpose for the workshops was to help the communication between security specialists and developers. There was a further technical expert, a programmer/system architect not introduced specifically to the researchers, who joined and contributed largely to the Threat Assessment and Security Sales Sessions.

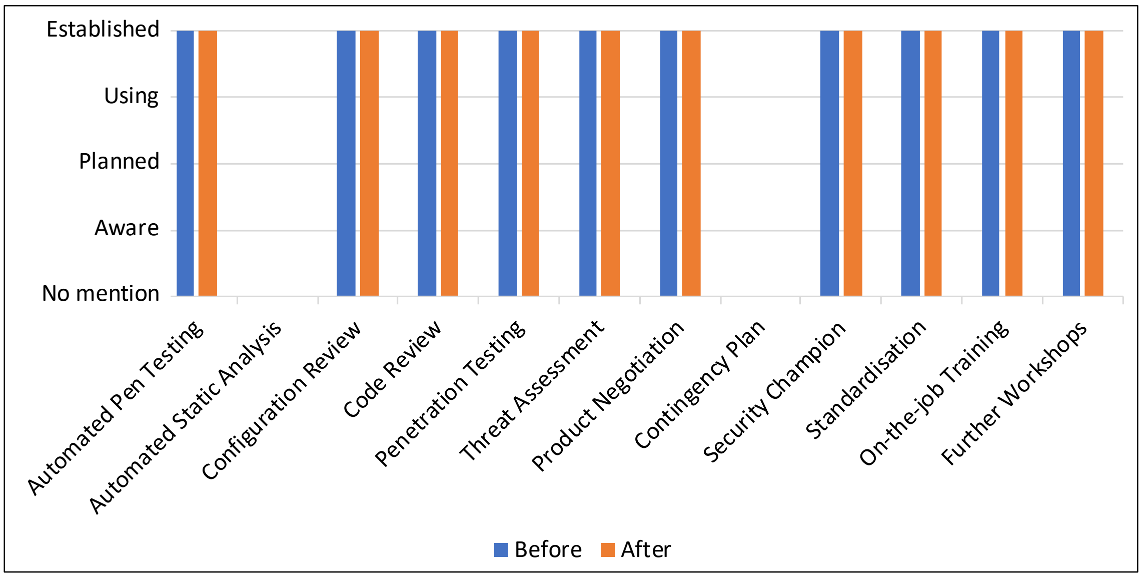

Figure 50: Group J Intervention Results

Figure 50 shows the effects of the intervention. As shown, the teams were expert at software security; while some of the participants may well have learned from the workshops there is no measurable improvement in the development process as a result.

Both groups E and J, the ones led by security professionals, showed a particular style of facilitation for the Threat Assessment session best described as 'directive'. The conversations were dominated by the facilitators (and in the case of J, one participant who had not been interviewed, and who appeared to have a system architect role). With Group E, the other participants tended to listen respectfully; with Group J, we observed several of the participants, unable to participate, getting on with their email instead.

8.6.9 Group K Results

Group K were from a well-established company with a few hundred employees providing tools for developers: some such tools are installed locally on client sites; others are cloud-hosted. The group had a strong emphasis on agile development processes, team interaction and structured workshops. The team style was correspondingly Athenian, with managers providing a coordinating role and professionals given considerable autonomy. The atmosphere was cheerful with some banter and an enthusiastic approach from the participants.

All the participants were developers; the two workshop leaders K1, K2 were amongst the most experienced in the group. Both had considerable experience in facilitating workshops, and brought their own approaches to brainstorming and analysing the results of brainstorming sessions. In particular, the techniques they used to engage participants were considerably more effective at getting participants' active engagement than those that had been used by the researchers or facilitators in most of the other groups, as follows.

For the Threat Assessment, they had each participant *independently* write down threat ideas on Post-It notes; then discuss them in groups. Participants then brought the notes they had created over to a whiteboard, where they sorted them together by type of threat and attacker, using location on the whiteboard to indicate similarity.

For the Risk Assessment, they had each participant annotate the post-it notes with coloured dots: dots of one colour for the ones they deemed most likely, and of another colour for the most impactful, as in Figure 51. During the break between workshops their

Figure 51: Whiteboard with Dots by Post-it Notes

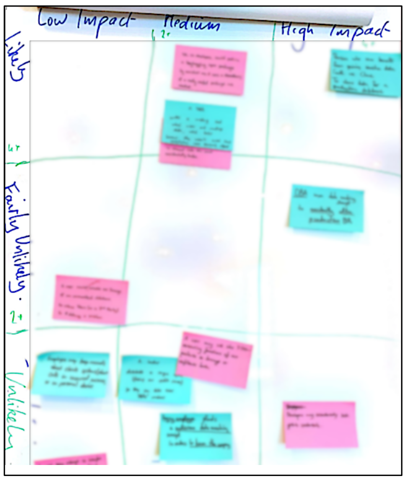

Figure 52: Whiteboard Showing Risk and Impact

facilitators then used this 'crowd-sourced' calculation to assign the threat post-its to a 3x3 matrix on a different whiteboard, as shown in Figure 52. This had 'low', 'medium' and 'high' impact on the x-axis; and 'likely', 'fairly unlikely' and 'unlikely' on the y-axis. Based on that, the facilitators then chose several threats to consider for the Threat Sales workshop—in this case the threats in the rightmost two squares in the top row.

For Security Sales, they took the stickers with the four most important identified threats, put them at different corners of the room, and had the participants group themselves next to their preferred topic; the participants self-organised to have similar numbers for each. Each group then worked out their 'pitch' for that threat and presented it to the others.

As shown in Figure 53, the researchers' analysis shows an increase in awareness in the teams of some of the assurance techniques.

8.6.10 Summary of Results

To gain an overview of the effectiveness of the interventions, this section analyses the changes in engagement levels seen in each group as a result of the analysis. By assigning ordinal ratings to the engagement levels as shown in Table 22, one can calculate an indication of the 'Impact' of the intervention—the extent to which the intervention affected the group's use of the technique.

Table 22: Engagement Levels

0 No mention
1 Aware
2 Planned
3 Using
4 Established

Note that this 'Impact' calculation is merely an indication: a two-unit Impact (change in engagement) might be from No Mention to

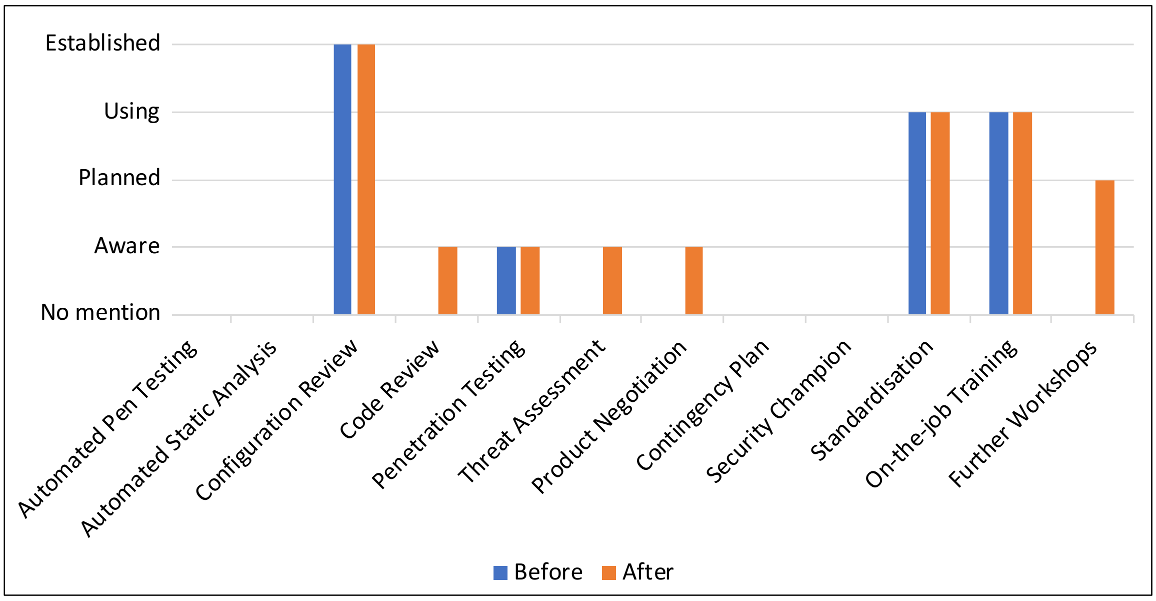

Figure 53: Group K Intervention Results

Planned, or from Planned to Established; these changes are not semantically equivalent. Equally, the number of samples (8) is insufficient for meaningful statistical analysis. However, we can reasonably assert that a large average Impact value represents more change than a small one.

Figure 54 summarises the results discussed in Sections 8.6.1 to 8.6.9, providing a summary answer to:

RQ 5.1 *To what extent did the groups learn about and adopt different security-enhancing activities as a result of the Developer Security Essentials intervention?*

The size of each bubble indicates the engagement level; the colour shows the change attributed to the intervention: amber for a change of 1 to 2 levels; red for 3 to 4 levels. As in earlier diagrams, the more concrete Assurance Techniques are nearer the top.

As the figure shows, the use of Threat Assessment and Product Negotiation was dramatically improved in a majority of groups; use of Penetration Testing and Use of Checklists were not affected at all. Group J showed little change as a result of the intervention; all the others did see at least some changes.

The results showed enhancements in the security activities and understanding in all the teams, albeit only a minor enhancement in the case of highly experienced group J. Groups I, J chose to carry out further workshops independently from the researchers, and D, E, F, G and I all showed major improvements in their use of Threat Assurance and Product Negotiation.

8.6.11 Technique Adoption by Different Categorisations of Group

This section addresses the second research sub-question:

RQ 5.2 *How does the impact of the intervention vary with different company sizes, team cultures, facilitation styles, security expertise, and kinds of participants?*

Table 24 calculates average impact values for different categorisations of the groups. The deeper shadings show the higher values in each categorisation; the red-green colours have no significance except to distinguish different categorisations. The line separators in the

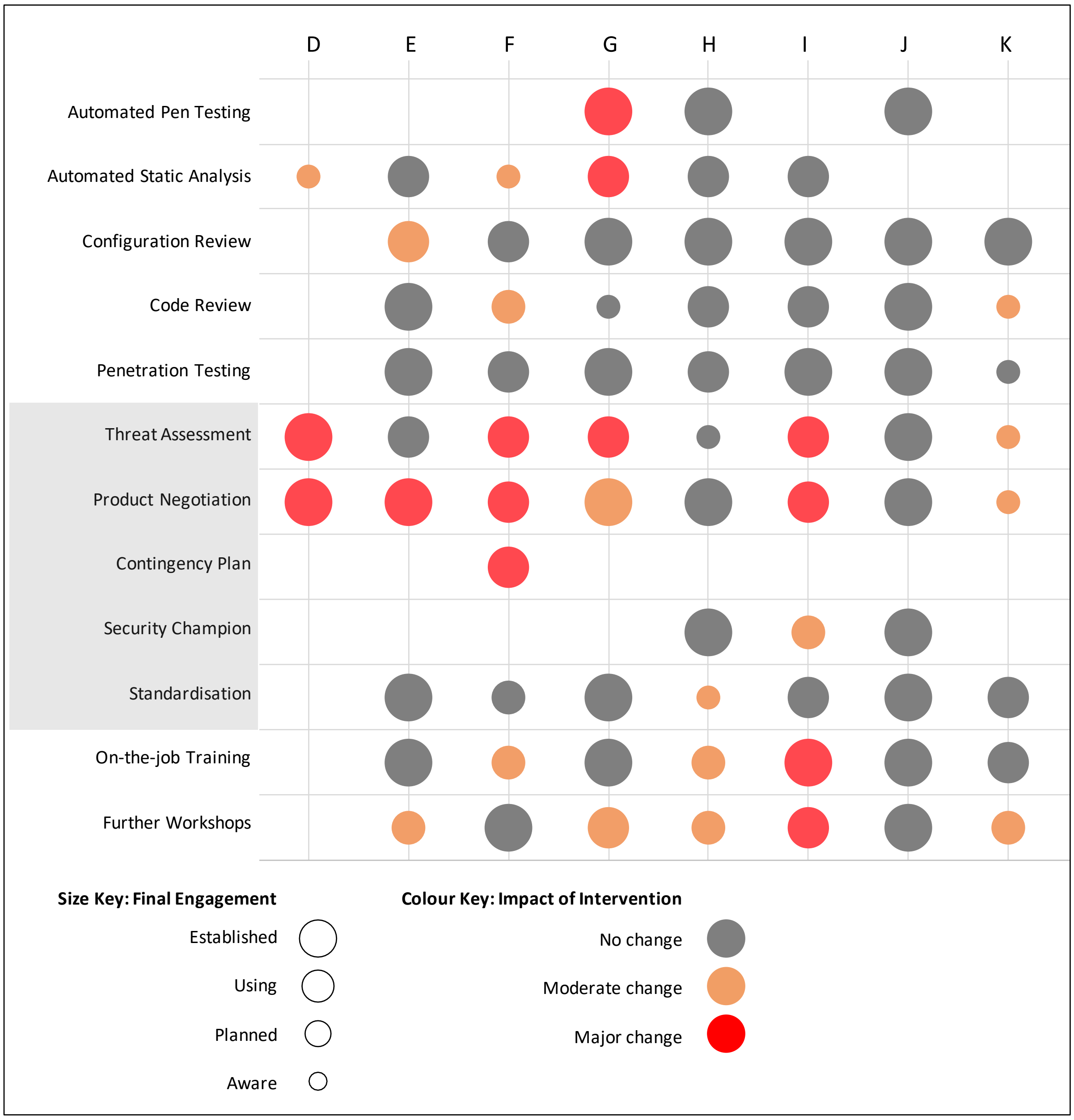

Figure 54: Changes in Assurance Technique Usage for All Groups

first column delineate the types of assurance technique (Vulnerability Finding, Process Improvements, and Training). The figures on the bottom line show the average increment over all assurance techniques for each category.

As shown, medium-size companies were most likely to plan further workshops; large companies were more likely to adopt Product Negotiation. Groups with Athena cultures (Section 8.2.1) were the most effective in adopting Threat Assessment and Product Negotiation.

Table 23 does the same as Table 24, but for a different set of categorisations: the security maturity of the group; whether or not product management was present; and the job title of the lead facilitator. Unsurprisingly, we see the groups with a low and medium initial security maturity achieved the biggest average impacts. Interestingly, though, we see that groups with product managers showed notably higher average impacts than those without; and those facilitated by managers achieved better impacts, especially in the process-related assurance techniques, than those facilitated by technical staff.

Table 24: Impact Averaged by Organisation Size and Team Culture

	Overall	Organisation Size			Team Culture		
		Large	Medium	Small	Zeus	Apollo	Athena
Count in each category	**8**	**3**	**3**	**2**	**3**	**2**	**3**
Automated Pen Testing	0.5		1.3		1.3		
Automated Static Analysis	0.6	0.3	1.	0.5	1.3		0.3
Configuration Review	0.1	0.3				0.5	
Code Review	0.4		0.3	1.	0.7		0.3
Penetration Testing							
Threat Assessment	1.6	1.	2.3	1.5	2.		2.3
Product Negotiation	2.1	2.7	2.	1.5	1.7	2.	2.7
Contingency Plan	0.4			1.5	1.		
Security Champion	0.3		0.7				0.7
Standardisation	0.1			0.5	0.3		
On-the-job Training	1.		1.3	2.	1.3		1.3
Further Workshops	1.3	0.3	2.3	1.	1.3	0.5	1.7
Average	0.7	0.4	0.9	0.8	0.9	0.3	0.8

Table 23: Impact Averaged over Group Descriptions

	Security Maturity			Product Manager Present?		Lead Facilitator		
	High	Medium	Low	Yes	No	Manager	Security	Developer
Count in each category	**2**	**4**	**2**	**4**	**4**	**4**	**2**	**2**
Automated Pen Testing		1.		1.		1.		
Automated Static Analysis		0.8	1.	1.	0.3	1.3		
Configuration Review	0.5				0.3		0.5	
Code Review		0.3	1.	0.5	0.3	0.5		0.5
Penetration Testing								
Threat Assessment		1.8	3.	2.3	1.	3.		0.5
Product Negotiation	2.	1.5	3.5	2.	2.3	3.	2.	0.5
Contingency Plan			1.5	0.8		0.8		
Security Champion		0.5		0.5		0.5		
Standardisation		0.3		0.3				0.5
On-the-job Training		1.5	1.	2.		1.5		1.
Further Workshops	0.5	2.3		1.8	0.8	1.3	0.5	2.
Average:	0.3	0.8	0.9	1.	0.4	1.1	0.3	0.4

Table 25: Impact of Different Categories of Intervention

		All	Vulnerability	Process	Training
All		0.7	0.3	0.9	1.1
Organisation Size	Large	0.4	0.1	0.7	0.2
	Medium	0.9	0.5	1.0	1.8
	Small	0.8	0.3	1.0	1.5
Team Culture	Zeus	0.9	0.7	1.0	1.3
	Apollo	0.3	0.1	0.4	0.3
	Athena	0.8	0.1	1.1	1.5
Security Maturity	High	0.3	0.1	0.4	0.3
	Medium	0.8	0.4	0.8	1.9
	Low	0.9	0.4	1.6	0.5
Product Manager	Yes	1.0	0.5	1.2	1.9
	No	0.4	0.2	0.7	0.4
Lead facilitator	Manager	1.1	0.6	1.5	1.4
	Security	0.3	0.1	0.4	0.3
	Developer	0.4	0.1	0.3	1.5

Finally, Table 25 shows a similar impact comparison, but this time showing how all of the categorisations in Table 24 and Table 23 compare when averaged over each of the three different kinds of technique: Vulnerability Finding, Process Improvement and Training. It shows that the training ones were adopted to a greater extent than the process ones, and both much more than the vulnerability collecting ones. We observe those led by managers were more likely to adopt process-based techniques; and that those with Athena cultures appear more likely to adopt training-based techniques, and relatively unlikely to adopt (additional) vulnerability-based ones.

We considered also investigating the different effects of the interventions on different kinds of participants: developers, testers, managers and product managers. However, the interviews and workshop transcripts did not distinguish the impact on individuals, but only on the entire team, which made this investigation impractical.

8.6.12 Which Forms of Facilitation Worked Well

Considering the final Design Practice research sub-question:

RQ 5.3 *How does the impact of the intervention vary with different facilitation styles?*

It was particularly clear to the researchers was that two of the three workshops, namely Threat Assessment and Threat Sales, varied very widely in effect, even though the recipe was fundamentally the same for all the groups.

Appendix K provides brief descriptions of the two workshops for each group, including assessments by the researcher for the 'energy level' in the room for each session, and brief reasons for why there was that energy level. Figure 55 summarises these descriptions. It shows the average impact (as in Table 25 in Section 8.6.11) plotted against

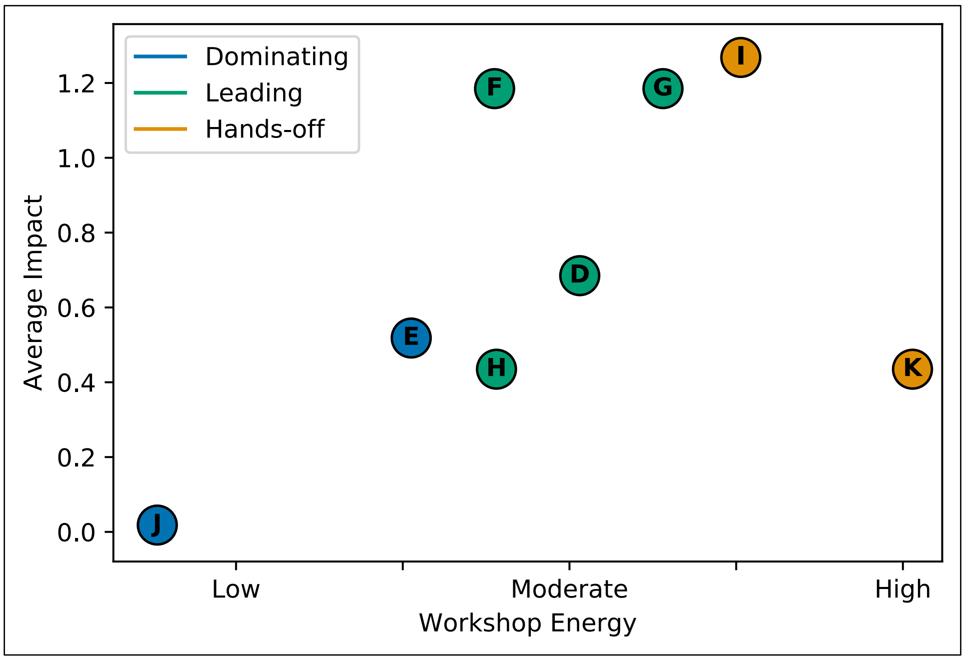

Figure 55: Impact vs. Energy, Categorised by Facilitation Style

the average energy in the room for the two workshops, categorised by the approach used by the facilitators. These approaches were:

Dominating	The facilitator(s) wrote up a board of 'ideas', but did most of the talking themselves, with interjections from the most experienced participants.
Leading	The facilitator(s) ran a more traditional brainstorming session, writing ideas up on a board but not contributing much otherwise themselves.
Hands-off	The facilitator(s) specified a format for the workshop and enforced procedure, but otherwise stayed out of the discussion or acted as equal participants themselves.

As the diagram shows the workshop energy did depend on the facilitation approach. Interestingly, though, the average impact (in terms of increases in engagement with assurance techniques) also rises with increasing workshop energy.

Both sessions led by security specialists involved a dominating style, and surprisingly, both were at the lower end of the impact; this was mainly because both groups had already adopted most of the techniques (Sections 8.6.3 and 8.6.8), but may also reflect the low level of energy achieved in the workshops.

8.7 Theory Results

Turning to the theoretical outcomes from Design-Based Research (Section 3.6.2), this section explores the two Design Theory research questions, RQ 5.4 and RQ 5.5 (Section 8.3.2).

8.7.1 Positive Benefits of Security and Privacy

Consider the first theory-based research question:

RQ 5.4 *Can having developers consider the positive benefits of security and privacy mitigations lead to security improvements in the development process?*

To address this, we looked for cases where the Threat Sales activity (Section 1) did lead to ‘security improvements in the development process’.

Group D identified in the Threat Sales discussion that the threat and risk assessment itself was a valuable asset to their clients as part of their Proof of Concept developments. They now incorporate a security discussion into every project they do in their ‘handover document’[36].

> *Now, after the workshop I think it was, we redesigned our handover template, which is where we now have a specific section for security.*
>
> *Even in the case of the project that I've just finished, ... even though it wasn't live code, we wanted to make the client aware that there was some security implications just with the data. (D4)*

Group F realised that they could ‘line up’ security improvements to be incorporated in the enhancements when new clients wanted them:

> *Yes, we are in a promising looking situation at the moment in terms of we have picked up some new contracts, and these are local authority contracts as well, so they will require us to implement pretty much everything that we had listed... I think the good thing is that if everything is detailed in the specification before we have even started implementation, then at least we are aware of what we need to do, how much effort it might be, rather than trying to deliver something for a fixed cost, and realise towards the end that “oh, we need to do all these things that we didn't cost or anticipate” (F1)*

Group G identified the ‘Gold, Silver, Bronze’ approach to selling security enhancement costs to their clients.

> *To make that process a lot simpler for our sales team, [G1] did a lot of the leg work and setting up a Gold, Silver and Bronze package to say “right, answer these 10 questions”, and then you would get a points score, and 'you fit in within this bracket, and this is the package that you need'. The work that Mike has done to start that process going has been brilliant. From then, [it’s] up to the company (G6)*

Group I subsequently included security requirements in discussions with new clients.

> *I feel a bit more confident, perhaps, if someone asks me a question saying “why aren't you doing this?” I can go: “I don't think we should because the threat is this, and there is these things, and these things, so therefore it is not worth it”. So, we are giving the Product Owners some more insight into why you would do this stuff, and where the value is. (I1)*

While we do not have evidence that the Threat Sales activity generated value in every case, the experience of Groups D and F, in particular, indicate that the activity of getting developers to consider the positive benefits of security can help get resources allocated to security improvements. We conclude, therefore, that the answer to RQ 5.4 is yes, having developers consider the positive benefits of security and privacy mitigations *can* lead to security improvements in the development process.

[36] Source: discussion in September 2020 between the author and D3.

8.7.2 Skills Not Associated with Developers

Consider the second Design Theory research sub-question:

RQ 5.5 *Can teams of developers produce both adequate risk and impact assessments and benefit analyses with minimal guidance?*

The Threat Sales workshop (Section 8.4.2) required the participants to establish the three or four highest priority threats; and for each, to find mitigations and then work out *positive ways in which addressing that threat will benefit the organisation.* That required two forms of analysis not normally associated with developers:

- Risk assessment: to identify the highest priority issues required an assessment of the probability of each threat identified in the Threat Assessment workshops; developers are not normally risk assessors.
- Sales promotion: Developers are not normally expected to promote enhancements; nor indeed to influence product management, other than through effort estimates.

Like every step in the development of these workshops, therefore, the Threat Sales workshop was an experiment; while the researchers had *hoped* that developers in the workshop would be able to assess risk and find positive ways to represent security to their stakeholders, they had no a priori reason to *believe* that they would. Indeed, other security researchers had assumed that the risk assessment process would require interaction with professional risk assessors[37]; and we are not aware of other work investigating the interaction of product management with security requirements.

The next two sections explore what happened related to each skill.

8.7.3 Risk Assessment

For the risk assessment, we suggested to the facilitators that they used a low-granularity approach, classifying the risk of each threat as low, medium or high.

Surprisingly, none of the teams had any trouble doing this. Even Group D, who are producing proof of concept apps for companies and are therefore not domain experts for their products, had little difficulty:

> *We've identified huge risks that they need to consider before they ever get anywhere near an actual working product. (Participant, Group D)*

Team E, whose security considerations prevented them discussing details of their product during the workshop, learned and took away the prioritisation process:

> *We had a follow-on session afterwards where we took everything away, ... and sat down and thought "what do we need to do next". (E3)*

For Group F, F1 produced a table of risks and impacts based on their discussion. Group G had no problem with risk assessments, since G1 and G2 were familiar with the likelihood of attacks on the websites they managed. Group H simply had their most expert members (H1, H3) identify the most likely by placing asterisks on the flipchart of identified threats. Group I did similar. Group J had J1 and J2 (facilitators and also security experts) do the assessment.

[37] Source: discussion with NCSC researchers.

Perhaps the most interesting approach was Group K as described in Section 8.6.9. Their facilitators had the team write the threats on post-it notes and had each participant annotate them with coloured dots: dots of one colour for the ones they deemed most likely, and of another colour for the most impactful. The results were notably consistent.

In summary, all the groups found effective ways to allocate a working risk probability to each of the threats they identified.

8.7.4 Threat Sales

Still more difficult for developers, one might have thought, would be devising ways to persuade product management professionals to allocate appropriate development resource to security improvements.

In larger groups, this process happened in several sub-groups, with each sub-group tackling a different threat.

The outcomes were surprisingly satisfactory:

- Group D identified that the threat and risk assessment itself was a valuable asset to their clients.
- Group E realised that while every security enhancement was essential, the ordering of their implementation could be altered to suit the client's needs.
- Group F 'lined up' security improvements to be incorporated in the enhancements when new clients wanted them.
- Group G identified the 'Gold, Silver, Bronze' approach to selling security enhancement costs to their clients.
- Group H identified that their security story was a major Unique Selling Point against competitors.
- Group I subsequently included security requirements in discussions with new clients.
- Group J devised several functionality and process improvements for their (internal) customers.
- In Group K, each of four subgroups delivered a convincing sales pitch for a security improvement.

It seems reasonable to conclude that developers generally do have the necessary skills and insights required to devise ways to persuade product management professionals to allocate resource to security improvements.

We conclude that the answer to RQ 5.5 is yes, teams of developers *can* produce both adequate risk and impact assessments and benefit analyses with minimal guidance.

8.8 Discussion

8.8.1 Security Process Improvements

The workshops concentrated on two aspects of security: using Threat Assessment to help participants focus their security effort on the appropriate threats; and encouraging Stakeholder Negotiation by finding ways to present security requirements as positive opportunities to product management. It was therefore encouraging, if unsurprising, that the results in Figure 54 (Section 8.6.10) show that all the groups completed the intervention with an understanding of the two assurance techniques, and, in a large majority of cases, the incorporation of those techniques into the groups' ways of working. Even more encouraging was that for half the groups involved this represented a large improvement over their previous practice.

Indeed, the majority of groups ended up incorporating, or at least using, all of the assurance techniques identified as important from the expert survey in Chapter 4: Automated Static Analysis, Configuration Review, Code Review, Penetration Testing, Threat Assessment and Product Negotiation.

Given the purpose of the new intervention package was to encourage others to use the package and lead sessions, it was encouraging that two companies did so; it was disappointing that a larger number had not got around to it after several months, even if they expressed intentions of doing so.

8.8.2 Variation of Results with Different Situations

The categorisations of Impact in Table 24 and Table 23 (Section 8.6.11) show several points of interest:

Sessions facilitated by managers appear more effective than those facilitated by developers or security specialists: This may reflect better training in facilitation-related skills given to managers; it may also reflect greater power amongst managers to introduce new techniques.

The presence or absence of a product manager in the group had negligible effect on the use of Stakeholder Negotiation: This was a surprise. The author had expected a product manager would encourage emphasis and therefore improvements in this, but the results do not show that effect. Surprisingly, though, looking at the bottom line, we see that the presence of a product manager did encourage the incorporation of other assurance techniques. This suggests there is good reason to encourage product managers to attend such sessions if possible.

Different cultures may favour improvements in different types of assurance techniques: The Zeusian cultures (Section 8.2.1) showed most improvement in Automated techniques and Threat Assessment; the Athenian ones in Threat Assessment and Product Negotiation. One might attribute the latter improvements to 'agency': Athenian cultures give the most 'agency' to the developers themselves as individuals, and thus once they accept the need they are the most able to implement Threat Assessment and Product Negotiation; assurance techniques like Automated Static Analysis, though they may well be less effective in practice, will appeal more to Zeusian cultures: as the 'magic bullets' supposedly popular with management.

Table 25 shows a point of interest when it comes to approaching other companies to use the interventions:

Medium sized companies, with moderate security knowledge, showed most tendency to adopt the interventions: The explanation is probably that such companies cannot afford a team of security specialists, but also will none the less have demanding customers who do need security, and also have 'latent' security knowledge amongst the team members that is brought out by the Developer Security Essentials workshops.

8.8.3 Best Facilitation Approaches

From Section 8.6.12, Figure 55, we see that, overall, the less participation by the facilitator in the workshops the more impact the intervention tended to have. The best facilitators, those who created the most energy in the workshops, were those who used a hands-off approach.

From observation of the workshops, we believe that the developers are learning and changing most when they are speaking, and discussing the security issues, not when they are told answers by a facilitator, nor even merely faced with a leader with a whiteboard marker. Best therefore is for a facilitator to create a good format and let the workshop delegates get on with it, only intervening with occasional information or in response to requests. This may also explain the finding in Section 8.8.2 that groups facilitated by managers tend to be particularly impactful, if the manager has the skill to use hands-off styles of facilitation, and is accepted as part of the team.

We conclude that, for maximum effectiveness, future versions of the Developer Security Essentials workshops should encourage facilitators to avoid taking a dominating role, and instead to provide a framework for the workshops, and support the developers themselves in driving the outcomes.

Particularly useful, then, is the approach used by Group K, as described in Section 8.7.3. Their session involved virtually no input from the facilitators other than coordination of the workshop process, and indeed, little security knowledge input from the researcher present. This approach, therefore, offers a possible basis for a prescriptive set of instructions for future facilitators, avoiding the need for the 'train the facilitator' sessions with the researchers.

8.8.4 Skills Not Associated with Developers

Sections 8.7.3 and 8.7.4 show that, surprisingly, developers found it easy both to assess the impact and likelihood of successful threat activities; and to think up ways of 'selling' security improvements to Product Management.

While we have no way of validating their assessments of either, we observe that there is good reason to believe that their assessments will be sufficient for the purpose:

- In the case of risk assessment, the consequence of getting one risk assessment wrong is much less than the consequence of not doing it at all. Since for all the teams, except perhaps Group E, there were no professional risk assessors available within the organisations to carry out the risk assessment, there is a strong argument for using what they produced.
- In the case of sales, in groups (F, G, H and I) where Product Managers were present, the Product Managers engaged very well with the process and found it valuable. This suggests that others may also find the results useful.

We conclude that there is no need for future versions of Developer Security Essentials to provide more sophisticated training in either risk assessment or sales; most teams will be able to carry out both workshops without it.

8.8.5 Impact of the Active Developer Model

A further implication of Figure 55 (Impact vs. Energy, Categorised by Facilitation Style) in section 8.6.12 concerns the Active Developer Model.

The diagram shows that the most effective workshops were those where developers acted virtually independently of the facilitators, generating their own conclusions and actions as a result; the least effective were those where the facilitator did most of the talking and the developers had a passive role.

While the small sample of 8 results does not permit statistical proof, this result is supportive of the Active Developer Model, which predicts that independent, self-activated developers will deliver better security than those ‘told what to do’.

8.8.6 Limitations

As with any research based on what are effectively multiple case studies, there are limitations in what can be deduced from this. Another series of trials might produce different results, since there are many factors outside the control of the researchers.

The coders did identify two specific limitations in the analysis process: identification of assurance techniques, and identification of the target group.

It was difficult in the coding process to identify assurance techniques and their level of adoption. Section 8.5 discusses some of the reasons for this; probably the most important issue was that the Before interviews did not ask explicitly about techniques, in order to avoid ‘priming’ the participants. In future research, we would recommend that, given such priming would be relatively small and short-term, it would be of more value to ask explicitly about assurance techniques beforehand.

The identification of the target group, *developers*, was an issue in the situations where security experts were also involved: groups E and J. Security experts are likely to be knowledgeable in all the assurance techniques mentioned here; they are also unlikely to change behaviour or knowledge as a result of the workshops. This research, however, is interested only in the impact on software developers. In Group E, the security expert E1 was separate from the developers, so it was straightforward to identify the extent to which the developers and the development team changed. However, in Group J, the security experts J1 and J2 were themselves experienced software developers, and were themselves working in the teams, so it was difficult for researchers to distinguish knowledge and intentions expressed by the experts from knowledge and intentions in the rest of the team; it may be that the intervention provided more benefit to the other Group J developers than Figure 54 suggests.

8.8.7 Future Work

The Developer Security Essentials package used in these trials has a practical limitation: it requires time input from a researcher to train the facilitators. This severely restricts its scalability to a wider audience of development teams, and hence the academic impact it can have.

The findings in the previous sections are helpful in addressing this:

1) The workshops are effective even when the contribution of the facilitator is limited to enforcing the procedures; they may not require them to have security expertise (Section 8.6.12).
2) One of the groups devised an effective template for the Threat Assessment and Threat Sales workshops suitable for step-by-step instructions (Sections 8.6.9, 8.8.3).

This opens the possibility of a new version of the package that needs no direct input from a researcher, and therefore can scale without limit.

The next step for the researchers is to create and trial such a package. However, a 'remote' trial of this kind offers several challenges to address, as follows:

Self-sufficient Support Materials: Supporting materials, such as instructions and the cards required for the Agile App Security Game, will need to be understandable and usable without the support of a researcher.

Participant Recruitment: to reach a larger number of development teams without the personal involvement of the researchers may require collaboration with marketing experts.

Data Collection: Without researchers present at the workshop in person the pre- and post- intervention interviews become problematic. In some cases these interviews might be carried out by phone or videoconference, but simpler and more scalable would be an online questionnaire based approach.

8.9 Conclusions

Recall the research question for this work:

RQ 5 *Which aspects of the 'Developer Security Essentials' intervention are effective at improving security when used independently by teams from a variety of cultures and different types of organisation, and why?*

The Magid 2 trials showed that Developer Security Essentials intervention led to security process improvements with all the groups who used it (Section 8.6.10). In particular, there was a strong improvement in the use of Threat Assessment to help participants focus their security effort on the appropriate threats; and Stakeholder Negotiation to encourage Product Management to allocate resource to threat mitigation (Figure 54).

All three workshops of the Developer Security Essentials were effective at helping improving security; developers proved adept even at risk assessment and creating positive representations of security improvements (Section 8.8.4).

The intervention had most impact where the workshops were facilitated by managers (Section 8.8.2) or in a 'hands-off' manner (Section 8.8.3). The Active Developer Model (Section 4.3) explains this: these are the arrangements that give the development team most 'volition'—that empower the development team most (Section 8.8.3).

The workshops were adopted most by medium sized companies; those that will have latent security expertise but no formal security function. They had least impact with very security-expert companies and when led by security specialists (Section 8.6.11), though this may reflect the lack of scope for improvement (Section 8.6.12).

The findings from this project promise a new version of the package that can scale without limit, and pave the way to the creation and trial of such a new package (Section 8.8.7). The author is currently working on a project to do exactly that.

9 Discussion and Conclusion

This chapter discusses conclusions from the research, both in terms of practical results and improvements to the package and in terms of wider theory gained. It contrasts work by other researchers, and outlines a range of further work to take the research forward.

9.1 Research Summary

At the start of the work on this thesis, the author expected that the answer to the research question

RQ 1 What is needed to make a cost-effective and widely applicable intervention to help UK software development teams achieve better software security?

would be some kind of tool or knowledgebase: a code analysis tool to alert the developers to possible errors, a website or book to teach them appropriate process tools, or a training course to transfer security skills.

This proved incorrect. While there is no doubt that all of these approaches are excellent ways to support software developers in improving their security, they all failed on the 'widely applicable' criteria; the industry survey described in Chapter 4 showed there were barriers stopping them from reaching the developers who needed them. Yet the online survey described in Chapter 5 showed there is definitely a need to overcome these barriers; fewer than two thirds of developers in teams were using any assurance techniques regularly.

Instead, therefore, what proved necessary before any of these tools could be of value to the developers in a team was something to 'sensitise' the developers to the importance of security and privacy. Chapter 6 describes the original creation of the package to do this and the thinking that led to it being a series of workshops.

Chapters 7 and 8 establish that the particular package we created, Developer Security Essentials, is remarkably effective. There is of course no reason to suggest that it is the only, or the best, such package that can be created. What we can say, is that it has a definite impact on the teams that use it, and a small cost, which justifies encouraging a large number of teams to use it.

The rest of this chapter summarises the detailed findings from the research, and suggests ways in which this impact might be realised.

9.2 Research Questions Revisited

Returning to the original research questions introduced in Section 1.6, the main thesis question RQ 1 led to a range of subsidiary questions RQ 2 to RQ 5. This section explores the answers to each of these subsidiary questions in turn, leading to an answer to the main question.

RQ 2 *What interventions can change the environment for members of the development team to achieve good security, considering cost-efficiency, motivational factors, choice of tools, supporting processes, culture, awareness, training and skills?*

Chapter 4 identified that effective interventions are those that conform to Active Developer Model: that developers must drive the security improvements themselves. Most of the effective such interventions are known by Security Specialists as 'Assurance Techniques', and of these, five are particularly cost-effective: Threat Assessment, Configuration Review, Automatic Static Analysis, Code Review and Penetration Testing. A further change to development process, Stakeholder Negotiation, also contributes largely to improved security; and two interventions, Incentivisation Session and On-the-Job Training, are effective in helping change the environment (Section 4.14).

RQ 3 *To what extent, and how, does a perceived need for security and privacy lead to security-enhancing activities and interactions in an Android development team and result in better software security?*

A survey of successful Android developers (Chapter 5) found a high level of reported security need for the app development, but less use of practical security assurance techniques such as the five discussed above, with less than 50% regularly using them; though of the third to a half of such developers working in teams nearly 80% used them regularly.

The use of such techniques was in proportion to the perceived need, as was the involvement of security specialists (though less than a quarter had this), and the frequency of app security updates. We found little relationship, however, between these factors and the density of security defects in the resulting apps; we believe this reflects the inability of the current generation of binary analysis tools to analyse libraries effectively and separately from the main app code. Surprisingly, we did find more Cryptographic API issues in apps whose development team worked with security specialists or champions, probably since the specialists and champions correctly enforce much more Cryptography use.

Reportedly, app security improvements have been largely driven by developers themselves; GDPR has had a minor impact (Section 5.5).

RQ 4 *What security outcomes did the 'Developer Security Essentials' package have, and what aspects contributed most to those outcomes?*

In the first year's trials of the Developer Security Essentials package (Chapter 7), two of the three organisations' teams achieved substantial improvements both in the security of their products and in their development process with respect to security. In particular, after a year or more, the first had identified that the choice of security improvements to make was a commercial, not a technical, question and had moved to a risk-based decision process; the second had incorporated security throughout their development training and process. The third company, a much larger organisation with a dysfunctional relationship between developers and security specialists, found that the interventions contributed to security awareness, but made no objectively detectable improvements as a result.

The main aspects that made the package effective were the extent to which it increased awareness rather than simply directed the teams involved. The effective teams were those who were able to do the following (Section 7.7.2.2):

Apply Team Activities to Security: The workshop effectiveness came from discussions between participants rather than information from the intervener.

Focus on Key Assurance Techniques: Specifically, Threat Assessment, Automated Static Analysis, and Configuration Review provide a large benefit for relatively low cost.

Address Organisational Issues: Nearly half the Motivators supporting security improvement, and Blockers discouraging it are ascribable to organisational issues, so addressing these can make a large contribution.

RQ 5 *Which aspects of the 'Developer Security Essentials' intervention are effective at improving security when used independently by teams from a variety of cultures and different types of organisation, and why?*

A second round of trials with 8 further organisations showed that the Developer Security Essentials intervention led to security process improvements with all the groups who used it (Chapter 8), especially in the use of Threat Assessment to help participants focus their security effort on the appropriate threats; and Stakeholder Negotiation to encourage Product Management to allocate resource to threat mitigation. All three workshops were effective at helping improving security; developers proved surprisingly adept even at risk assessment and creating positive representations of security improvements.

The intervention had least impact with very security-expert companies and when led by security specialists, though this may reflect the lack of scope for improvement. They had most impact where the workshops were facilitated by managers or in a 'hands-off' manner, probably because these arrangements empower the development team most.

9.2.1 Main Research Question

The previous questions were all derived from considering the original research question:

RQ 1 *What is needed to make a cost-effective and widely applicable intervention to help UK software development teams achieve better software security?*

From the above findings, we conclude that what is needed in any such intervention are the following properties:

- It sensitises a development team to the importance of security and privacy issues in developers, and enables them to make their own choices as to the best ways to address them.
- It helps the developers understand that every project has unique requirements related to security and privacy; and teaches them Threat Assessment approaches to establish and assess those requirements.
- It introduces them to the four further assurance techniques that are most effective in removing security issues during development: Configuration Review, Automatic Static Analysis, Code Review and Penetration Testing.
- It supports the developers in what we have termed 'Stakeholder Negotiation': finding ways establish the business value, or lack of it, for addressing security

requirements; and hence supporting good business decisions about where to spend effort and money on security and privacy improvements.

From the experimental work described in this thesis, we can add the observation that the following two techniques provide a good basis for such an intervention:

1. Workshops, in which the developers carry out structured discussions.
2. Regular On-the-job Training sessions afterward to act as a 'nudging' reminder.

9.3 Main Contributions

As introduced in section 1.5, the main new contributions of this research are the following:

1. The Active Developer Model
2. Favourite Intervention Techniques
3. Measurement of Assurance Technique Use
4. Proof of Concept of an Intervention Package
5. Pioneering Use of Design-Based Research
6. Business Benefit of Security

The following sections explore each of these in more detail.

9.3.1 The Active Developer Model

While Chapter 4 could only propose the Active Developer Model as a candidate theory, as discussed in Section 3.2, given our pragmatic approach such a theory can be accepted for future use *if it helps understand a complex situation, if deductions made using it are all confirmed, and if using it to plan research or interventions leads to desired outcomes.*

Section 4.3.2 shows how the Active Developer Model helps understand the complex situation of interventions to support security, by helping explain which interventions will be successful in terms of their Sensitisation, Support and Affordability.

All the deductions of the Active Developer Model explored (post hoc) in Sections 7.7.3 and 8.8.5 were borne out in the Magid 1 and Magid 2 projects, also corroborating the theory.

Finally, Sections 6.2.1 and 6.3 show how the theory was used to plan the design of the Developer Security Essentials: in particular, emphasising the need for an Incentivisation Session. Since the intervention was generally successful (Section 8.6.11), this supports accepting the theory.

It is reasonable to conclude that the Active Developer Model theory is valuable for intervention design, and to recommend its use in future.

9.3.2 Favourite Intervention Techniques

Section 4.14.1 identified eight intervention techniques favoured by security experts working with development teams:

Incentivisation Session	to motivate security improvement;
Threat Modelling	to identify the risks and benefits to the organisation from security issues;
Product Negotiation	to prioritise and justify effort and expense on mitigations;

Component Choice	to use tools to identify weak or out-of-date components;
Auto. Static Analysis	to facilitate the removal of certain classes of code-level vulnerabilities;
Code Review	to find more sophisticated vulnerabilities, where the culture permits;
Penetration Testing	to view the software from the point of view of an attacker; and finally,
On-the-Job Training	to keep the team actively considering software security

9.3.3 Measurement of Assurance Technique Use

Chapter 5 provided evidence of minimal use of security assurance techniques by developers in the particular domain of Android App development. Specifically, less than half of the developers use any assurance techniques regularly; less than two thirds of those working in teams do so.

The survey also provide evidence of the need for interventions that did not require security professionals; less than a quarter of those developers had access to such professionals.

9.3.4 Proof of Concept of an Intervention Package

Chapter 7 provided an ‘existence proof’ that a simple ‘intervention package’ structured as a facilitated series of workshops can improve the security of software developed by a team.

Specifically, the first phase of trials of the Developer Security Essentials package led, for two of the three organisations involved, to improvements in both project security and understanding among the more experienced team members of the need for assurance techniques (Section 7.7.1)

9.3.5 Pioneering Use of Design-Based Research

As Section 3.6 explains, Design-Based Research has been used mostly in the field of education research. While the creation of an intervention in the field of Developer-centred Security is arguably a form of education, we are not aware of any other researchers using Design-Based Research in this field.

As Chapter 8 shows, Design-Based Research proved an effective basis for trialling, evaluating, and deducing theory from the creation of a Developer-centred Security intervention.

9.3.6 Business Benefit of Security

Perhaps the most exciting outcome for future research was, first, the identification of the importance of representing security enhancements in terms of their business benefit (Section 7.7.6); and second, the discovery that teams of software developers were both capable and happy to identify these business benefits when asked to do so (Section 8.8.4)

While there is considerable current interest in establishing the economics of software security and privacy enhancements at organisations’ board level[38], there has been little

[38] The UK NCSC recently awarded a number of small grants for research around this topic.

research into economic behaviour at the developer level, and this offers an exciting scope for further research.

9.4 Uniqueness of this Research

There has been little academic research into interventions to support software developers in addressing security and privacy issues. Secure Development Lifecycles have had little attention since 2010; possibly since many developers rejected them (Section 2.2.2). While there has been some work on how to encourage the adoption of tools, this has not translated into practical interventions (Section 2.2.3).

Several researchers explored consultancy-based, training-based, and literature-based interventions, but without finding any formulae for success (Section 2.2.4). Though there have been some encouraging experiments with ways to improve the interactions between Security Experts and others (Section 2.2.5), and to improve security behaviour in company employees (Section 2.2.7), none of these have been presented in a way that would support anyone else in using the techniques.

The 'Motivating Jenny' project has explored motivational means to encourage developer teams in doing security. Encouragingly, their conclusions based on ethnographic techniques were similar to the conclusions in this thesis based on industry survey (Section 4.14). Their interventions do promote security (Section 2.2.6); however only the research described in this thesis puts any objective measurement on the effectiveness of an intervention.

9.5 Future Research Agenda

The findings in this thesis point to two further areas of future research: the microeconomics of security and privacy improvements; and the mass deployment of the Developer Security Essentials package. The next two sections explore these topics.

9.5.1 Microeconomics of Security and Privacy Improvements

An important theme throughout the thesis has been 'Product Negotiation'. It is the only 'new' intervention technique, in that it derived from the analysis of experts' and participants' comments rather than from other research (Sections 7.7.2.1, 8 Introduction, 1, 8.7.1). Section 8.7.4 shows that developers had less trouble than we had expected in identifying microeconomic benefits to security. However, we have no evidence as to how effective their use was of the technique: how the subsequent discussions with product management, customers and other stakeholders went. Nor do we know what might improve the process of identifying benefits, analysing them and discussing them.

So, a future research project will address these questions, asking research questions such as:

> *What makes effective discussions between developers and other stakeholders about security and privacy issues?*

The research project will, naturally, need to define 'effective' in appropriate microeconomic and commercial terms, and will generate suitable interventions to improve the effectiveness of such discussions.

9.5.2 Mass Deployment of Developer Security Essentials

The current Developer Security Essentials package, though effective at improving development teams' software security in the short and long term, has a practical limitation: it requires time from a researcher to train the facilitators. Yet to have an appreciable impact on the security of UK software, the package needs to be saleable to a much wider audience of development teams; the findings from the last set of trials show that this is possible (Section 8.8.7).

A further research project, therefore, will convert Developer Security Essentials to a form suitable for mass deployment, and measures its success. By adding detailed and approachable instructions, full materials, and a supporting website, it will be made suitable for any development team to use at any time, potentially allowing viral and mass-market spread throughout the UK developer community. By collaborating with marketing experts, the package will be promoted to a wide range of development teams. And to measure the intervention's effectiveness in this form, the project will create online survey mechanisms and support for surveying participants beforehand and afterwards, to collect data from as many users of the intervention package as possible.

9.6 Conclusion

The research described in this thesis has established software developers themselves as the best drivers for software security and privacy improvements; established the need for improvements in at least one developer population; identified requirements for a lightweight intervention to improve software security; created such an intervention; and iteratively trialled and improved it with many software development teams.

Mass deployment of the Developer Security Essentials intervention will improve the cyber security of software developed throughout the UK and worldwide, reducing harms to both individuals and organisations. As the trials have proved, the effect of a large number of software development teams using the Developer Security Essentials intervention will be improvement in security-related activities in development projects, with a resulting improvement in the security and privacy of the software on which we all rely.

10 References

[1] Abadi, M. 11 Dramatic Ways the World Has Changed in the Last 20 Years Alone. *Business Insider*, 2018. https://www.businessinsider.com/progress-innovation-since-1998-2018-3.

[2] Acar, Y., Backes, M., Fahl, S., et al. Comparing the Usability of Cryptographic Apis. *Symposium on Security and Privacy - S&P*, IEEE (2017), 154–171.

[3] Acar, Y., Backes, M., Fahl, S., Kim, D., Mazurek, M.L., and Stransky, C. You Get Where You're Looking For: The Impact of Information Sources on Code Security. *Symposium on Security and Privacy - S&P*, IEEE (2016), 289–305.

[4] Acar, Y., Fahl, S., and Mazurek, M.L. You Are Not Your Developer, Either: A Research Agenda for Usable Security and Privacy Research Beyond End Users. *Conference on Cybersecurity Development - SecDev*, IEEE (2016), 3–8.

[5] Acar, Y., Stransky, C., Wermke, D., Mazurek, M.L., and Fahl, S. Security Developer Studies with GitHub Users: Exploring a Convenience Sample. *Symposium on Usable Privacy and Security - SOUPS*, USENIX Association (2017).

[6] Acar, Y., Stransky, C., Wermke, D., Weir, C., Mazurek, M.L., and Fahl, S. Developers Need Support, Too: A Survey of Security Advice for Software Developers. *Secure Development Conference - SecDev*, IEEE (2017), 22–26.

[7] Adolph, S., Hall, W., and Kruchten, P. Using Grounded Theory to Study the Experience of Software Development. *Empirical Software Engineering 16*, 4 (2011), 487–513.

[8] Allan, G. A Critique of Using Grounded Theory as a Research Method. *The Electronic Journal of Business Research Methods 2*, 1 (2003), 1–10.

[9] Anderson, R. *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, Indianapolis, IN, USA, 2008.

[10] Anscombe, F.J. The Transformation of Poisson, Binomial and Negative-Binomial Data. *Biometrika 35*, 3/4 (1948), 246.

[11] Arzt, S., Rasthofer, S., Fritz, C., et al. FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps. *Conference on Programming Language Design and Implementation - PLDI*, ACM (2014).

[12] Ashenden, D. and Lawrence, D. Security Dialogues: Building Better Relationships between Security and Business. *IEEE Security & Privacy 14*, 3 (2016), 82–87.

[13] Assal, H. and Chiasson, S. Security in the Software Development Lifecycle. *Symposium on Usable Privacy and Security - SOUPS*, USENIX Association (2018), 281–296.

[14] Assal, H. and Chiasson, S. Think Secure From the Beginning: A Survey With Software Developers. *Conference on Human Factors in Computing Systems - CHI*, ACM (2019).

[15] Ayewah, N., Pugh, W., Hovemeyer, D., Morgenthaler, J.D., and Penix, J. Using Static Analysis to Find Bugs. *IEEE Software 25*, 5 (2008), 22–29.

[16] Backes, M., Bugiel, S., and Derr, E. Reliable Third-Party Library Detection in Android and Its Security Applications. *Conference on Computer and Communications Security - CCS*, ACM (2016), 356–367.

[17] Bai, X., Zhang, F., and Lee, I. Predicting the Citations of Scholarly Paper. *Journal of Informetrics 13*, 1 (2019), 407–418.

[18] Bakker, A. *Design Research in Education: A Practical Guide for Early Career Researchers*. Routledge, Abingdon, 2018.

[19] Balebako, R. and Cranor, L. Improving App Privacy: Nudging App Developers to Protect User Privacy. *IEEE Security and Privacy 12*, 4 (2014), 55–58.

[20] Balebako, R., Marsh, A., Lin, J., Hong, J., and Cranor, L. The Privacy and Security Behaviors of Smartphone App Developers. *Internet Society*, October (2014).

[21] Bank of America. Bank of America Revolutionizes Banking Industry. https://about.bankofamerica.com/en-us/our-story/bank-of-america-revolutionizes-industry.html.

[22] Barab, S. and Squire, K. Design-Based Research: Putting a Stake in the Ground. *Journal of the Learning Sciences 13*, (2004).

[23] Baskerville, R. and Wood-Harper, A.T. Diversity in Information Systems Action Research Methods. *European Journal of Information Systems 7*, 2 (1998), 90–107.

[24] Baskerville, R.L. Investigating Information Systems with Action Research. *Communications of the Association for Information Systems 2*, (1999), 4.

[25] Baum, T., Liskin, O., Niklas, K., and Schneider, K. Factors Influencing Code Review Processes in Industry. *Symposium on the Foundations of Software Engineering - FSE*, (2016), 85–96.

[26] Becker, I., Parkin, S., and Sasse, M.A. Finding Security Champions in Blends of Organisational Culture. *European Workshop on Usable Security - EuroUSEC*, (2017).

[27] Beecham, S., Baddoo, N., and Hall, T. Motivation in Software Engineering: A Systematic Literature Review. *Information and Software Technology 50*, 9 (2008), 860–878.

[28] Bell, L., Brunton-Spall, M., Smith, R., and Bird, J. *Agile Application Security: Enabling Security in a Continuous Delivery Pipeline*. O'Reilly, Sebastopol, CA, 2017.

[29] Bessey, A., Engler, D., Block, K., et al. A Few Billion Lines of Code Later. *Communications of the ACM 53*, 2 (2010), 66–75.

[30] Box, G.E.P. and Cox, D.R. An Analysis of Transformations. *Journal of the Royal Statistical Society 26*, 2 (1964), 211–252.

[31] Brown, A.L. Design Experiments : Theoretical and Methodological Challenges in Creating Complex Interventions in Classroom Settings. *Journal of the Learning Sciences 2*, 2 (1992), 141–178.

[32] Caputo, D.D., Pfleeger, S.L., Sasse, M.A., Ammann, P., Offutt, J., and Deng, L. Barriers to Usable Security? Three Organizational Case Studies. *IEEE Security and Privacy 14*, 5 (2016), 22–32.

[33] Charmaz, K. *Constructing Grounded Theory*. Sage, London, 2014.

[34] Christakis, M. and Bird, C. What Developers Want and Need From Program Analysis: An Empirical Study. *International Conference on Automated Software Engineering - ASE*, ACM Press (2016), 332–343.

[35] Clarke, V., Braun, V., and Hayfield, N. Thematic Analysis. In J.A. Smith, ed., *Qualitative Psychology: A Practical Guide to Research Methods*. SAGE Publications, 2015, 222–248.

[36] Collins, A. Toward a Design Science of Education. In *New Directions in Educational Technology*. Springer, 1992, 15–22.

[37] Conradi, R. and Dybå, T. An Empirical Study on the Utility of Formal Routines to Transfer Knowledge and Experience. *ACM SIGSOFT Software Engineering Notes 26*, 5 (2001), 268–276.

[38] CONSORT. Checklist of Information to Include When Reporting a Randomized Trial. 2010, 11–12. http://www.consort-statement.org/consort-2010.

[39] Coopamootoo, K.P.L. and Gross, T. *A Codebook for Evidence-Based Research: The Nifty Nine Completeness Indicators*. Newcastle, 2017.

[40] Cooperrider, D.L. and Whitney, D. *Appreciative Inquiry: A Positive Revolution in Change*. Berrett-Koehler, San Francisco, CA, USA, 2005.

[41] Cooperrider, D.L., Whitney, D.K., and Stavros, J.M. *Appreciative Inquiry Handbook*. Berrett-Koehler Publishers, 2003.

[42] Craft, R.C. and Leake, C. The Pareto Principle in Organizational Decision Making. *Management Decision 40*, 8 (2002), 729–733.

[43] Davison, R.M., Martinsons, M.G., and Kock, N. Principles of Canonical Action Research. *Information Systems Journal 14*, 1 (2004), 65–86.

[44] Deborah J. Rumsey. *Statistics Essentials For Dummies*. Wiley, For Dummies, 2019.

[45] Derr, E., Bugiel, S., Fahl, S., Acar, Y., and Backes, M. Keep Me Updated: An Empirical Study of Third-Party Library Updatability on Android. *Conference on Computer and Communications Security - CCS*, ACM Press (2017), 2187–2200.

[46] Devanbu, P. and Stubblebine, S. Software Engineering for Security: A Roadmap. *International Conference on the Future of Software Engineering - ICSE*, (2000), 227–239.

[47] Dittrich, Y., Rönkkö, K., Eriksson, J., Hansson, C., and Lindeberg, O. Cooperative Method Development: Combining Qualitative Empirical Research With Method, Technique and Process Improvement. *Empirical Software Engineering 13*, 3 (2008), 231–260.

[48] Dybå, T. An Empirical Investigation of the Key Factors for Success in Software Process Improvement. *IEEE Transactions on Software Engineering 31*, 5 (2005), 410–424.

[49] Easterbrook, S., Singer, J., Storey, M.-A., and Damian, D. Selecting Empirical Methods for Software Engineering Research. In *Guide to Advanced Empirical Software Engineering*. Springer, London, 2008, 285–311.

[50] Edelson, D.C. Design Research: What We Learn When We Engage in Design. *Journal of the Learning Sciences 11*, 1 (2002), 105–121.

[51] Egelman, S. and Peer, E. Scaling the Security Wall : Developing a Security Behavior Intentions Scale (SeBIS). *Conference on Human Factors in Computing Systems - CHI*, ACM (2015).

[52] Eichberg, M. and Hermann, B. A Software Product Line for Static Analyses: The OPAL Framework. *Conference on Programming Language Design and Implementation - PLDI*, ACM (2014).

[53] Ejersbo, L.R., Engelhardt, R., Frølunde, L., Hanghøj, T., Magnussen, R., and Misfeldt, M. Balancing Product Design and Theoretical Insights. In *The Handbook of Design Research Methods in Education*. Routledge, 2008, 149–164.

[54] Enck, W., Gilbert, P., Chun, B.G., et al. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. *USENIX Symposium on Operating Systems Design and Implementation - OSDI*, USENIX Association (2010).

[55] European Commission. General Data Protection Regulation (GDPR). 2019. https://ec.europa.eu/info/law/law-topic/data-protection_en.

[56] Fahl, S., Harbach, M., Muders, T., Smith, M., Baumgärtner, L., and Freisleben, B. Why Eve and Mallory Love Android: An Analysis of Android SSL Security Categories and Subject Descriptors. *Conference on Computer and Communications Security - CCS*, ACM Press (2012).

[57] Fahl, S., Harbach, M., Perl, H., Koetter, M., and Smith, M. Rethinking SSL Development in an Appified World. *Conference on Computer & Communications Security - CCS*, ACM Press (2013), 49–60.

[58] Faily, S. and Flechais, I. Persona Cases: A Technique for Grounding Personas. *Conference on Human Factors in Computing Systems - CHI*, ACM (2011), 2267–2270.

[59] Felderer, M., Büchler, M., Johns, M., Brucker, A.D., Breu, R., and Pretschner, A. Security Testing: A Survey. *Advances in Computers 101*, (2016), 1–51.

[60] Fischer, F., Bottinger, K., Xiao, H., et al. Stack Overflow Considered Harmful? the Impact of Copy&Paste on Android Application Security. *Symposium on Security and Privacy - S&P*, IEEE (2017), 121–136.

[61] Fisher, R., Ury, W.L., and Patton, B. *Getting to Yes: Negotiating Agreement Without Giving In*. Penguin, 2011.

[62] Fogg, B.J. *Persuasive Technology: Using Computers to Change What We Think and Do*. Morgan Kaufmann, 2003.

[63] Fogg, B.J. A Behavior Model for Persuasive Design. *International Conference on Persuasive Technology - PERSUASIVE*, ACM (2009), 40:1--40:7.

[64] Fowler, F.J. *Survey Research Methods*. SAGE, Thousand Oaks, CA, 2009.

[65] Frey, S., Rashid, A., Anthonysamy, P., Pinto-Albuquerque, M., and Naqvi, S.A. The Good, the Bad and the Ugly: A Study of Security Decisions in a Cyber-Physical Systems Game. *IEEE Transactions on Software Engineering*, (2017), 1–16.

[66] Furniss, D., Blandford, A.A., and Curzon, P. Confessions From a Grounded Theory PhD: Experiences and Lesson Learnt. *Conference on Human Factors in Computing Systems - CHI*, ACM (2011), 113.

[67] Gagné, M. and Deci, E.L. Self-Determination Theory and Work Motivation. *Journal of Organizational Behavior 26*, 4 (2005), 331–362.

[68] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-oriented Software*. Pearson Education, 1994.

[69] Ge, X., Paige, R., Polack, F., Chivers, H., and Brooke, P. Agile Development of Secure Web-Based Applications. *International Conference on Web Engineering - ICWE*, ACM (2006), 1–24.

[70] Geer, D. Are Companies Actually Using Secure Development Life Cycles? *IEEE Computer June*, 2010, 12–16.

[71] Gibbs, G. *Qualitative Data Analysis: Explorations with NVivo*. Cromwell Press, Trowbridge, Wiltshire, UK, 2002.

[72] Glanz, L., Amann, S., Eichberg, M., et al. CodeMatch: Obfuscation Won't Conceal Your Repackaged App. *European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE*, ACM (2017), 638–648.

[73] Glaser, B.G. *Theoretical Sensitivity*. Sociology Press, 1978.

[74] Glaser, B.G. and Strauss, A.L. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine Transaction, Chicago, 1973.

[75] Goldkuhl, G. Pragmatism vs Interpretivism in Qualitative Information Systems Research. *European Journal of Information Systems 21*, 2 (2012), 135–146.

[76] Green, M. and Smith, M. Developers are Not the Enemy!: The Need for Usable Security APIs. *IEEE Security and Privacy 14*, 5 (2016), 40–46.

[77] Guest, G., Bunce, A., and Johnson, L. How Many Interviews are Enough? An Experiment with Data Saturation and Variability. *Field Methods 18*, 1 (2006), 59–82.

[78] Gwet, K.L. *Handbook of Inter-Rater Reliability: The Definitive Guide to Measuring the Extent of Agreement Among Raters*. Advanced Analytics LLC, 2014.

[79] Hall, T., Sharp, H., Beecham, S., Baddoo, N., and Robinson, H. What Do We Know about Developer Motivation? *IEEE Software 25*, 4 (2008), 92–94.

[80] Handy, C.B. *Understanding Organizations*. Penguin Books, 1993.

[81] Haney, J.M. and Lutters, W.G. Skills and Characteristics of Successful Cybersecurity Advocates. *Workshop on Security Information Workers - SIW*, USENIX Association (2017).

[82] Haney, J.M. and Lutters, W.G. It's Scary… It's Confusing… It's Dull: How Cybersecurity Advocates Overcome Negative Perceptions of Security. *Symposium on Usable Privacy and Security - SOUPS*, USENIX Association (2018), 411–425.

[83] Hardgrave, B., Davis, F., and Riemenschneider, C. Investigating Determinants of Software Developers' Intentions to Follow Methodologies. *Management Information Systems 20*, 1 (2003), 123–151.

[84] Herzberg, F. One More Time: How Do You Motivate Employees? *Harvard Business Review 46*, January/February (1968), 53–62.

[85] Hilton, M., Nelson, N., Tunnell, T., Marinov, D., and Dig, D. Trade-Offs in Continuous Integration: Assurance, Security, and Flexibility. *European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE*, ACM (2017), 197–207.

[86] Hoadley, C., Baumgartner, E., Bell, P., et al. Design-Based Research: An Emerging Paradigm for Educational Inquiry. *Educational Researcher 32*, 1 (2002), 5–8.

[87] Hoda, R., Noble, J., and Marshall, S. Grounded Theory for Geeks. *Conference on Pattern Languages of Programs - PLoP*, ACM (2011), 1–17.

[88] Howard, M., LeBlanc, D., and Viega, J. *24 Deadly Sins of Software Security: Programming Flaws and How to Fix Them*. McGraw-Hill, Inc., New York, NY, 2009.

[89] Ioannidis, J.P.A. Why Most Published Research Findings Are False. *PLOS Medicine 2*, 8 (2005), 0696–0701.

[90] ISO/IEC. 21827:2008 - Systems Security Engineering - Capability Maturity Model. 2008, 144.

[91] James Carifio and Rocco J. Perla. Ten Common Misunderstandings, Misconceptions, Persistent Myths and Urban Legends about Likert Scales and Likert Response Formats and their Antidotes. *Journal of Social Sciences 3*, 3 (2007), 106–116.

[92] Johnson, B., Song, Y., Murphy-Hill, E., and Bowdidge, R. Why Don't Software Developers Use Static Analysis Tools to Find Bugs? *nternational Conference on Software Engineering - ICSE*, IEEE (2013), 672–681.

[93] Jones, D.W. *Deep Secret*. Harper Collins.

[94] Kirlappos, I., Beautement, A., and Sasse, M.A. "Comply or Die" Is Dead: Long Live Security-Aware Principal Agents. In *Financial Cryptography and Data Security*. Springer Berlin, Heidelberg, 2013, 70–82.

[95] Kirlappos, I., Parkin, S., and Sasse, M.A. Shadow Security as a Tool for the Learning Organization. *Computers and Society 45*, 1 (2015), 29–37.

[96] Kline, T. Classical Test Theory: Assumptions, Equations, Limitations, and Item Analyses. In *Psychological Testing: A Practical Approach to Design and Evaluation*. SAGE Publications, Inc., Thousand Oaks, California, 2005.

[97] Kluyver, T., Ragan-kelley, B., Pérez, F., et al. Jupyter Notebooks: A Publishing Format for Reproducible Computational Workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. IOS Press, 2016, 87–90.

[98] Kohn, A. Why Incentive Plans Cannot Work. *Harvard Business Review*, (1993).

[99] Kohn, M. and Senyak, J. Sample Size Calculators. http://www.sample-size.net/.

[100] Kruger, S., Nadi, S., Reif, M., et al. CogniCrypt: Supporting Developers in Using Cryptography. *Conference on Automated Software Engineering - ASE*, IEEE (2017), 931–936.

[101] Lancaster University. Information Security Policy. 2019, 1–15. https://www.lancaster.ac.uk/iss/security/policy/.

[102] Lau, F. A Review on the Use of Action Research in Information Systems Studies. *Information Systems and Qualitative Research*, (1997), 31–68.

[103] Lavallee, M. and Robillard, P.N. The Impacts of Software Process Improvement on Developers: A Systematic Review. *International Conference on Software Engineering - ICSE*, IEEE (2012), 113–122.

[104] Lewin, K. Action Research and Minority Problems. *Journal of Social Issues 2*, 4 (1946), 34–46.

[105] Lewis, C.F. Motivational Design Patterns. 2013.

[106] Likert, R. A Technique for the Measurement of Attitudes. *Archives of Psychology 140*, (1932).

[107] van der Linden, D., Anthonysamy, P., Nuseibeh, B., et al. Schrödinger's Security: Opening the Box on App Developers' Security Rationale. *International Conference on Software Engineering - ICSE*, IEEE (2020).

[108] Lopez, T., Nuseibeh, B., Bandara, A.K., et al. Motivating Jenny to Write Secure Software. 2020. https://motivatingjenny.org.

[109] Lopez, T., Sharp, H., Tun, T., Bandara, A., Levine, M., and Nuseibeh, B. Hopefully We Are Mostly Secure: Views on Secure Code in Professional Practice. *Workshop on Cooperative and Human Aspects of Software Engineering - CHASE*, IEEE (2019), 61–68.

[110] Lopez, T., Sharp, H., Tun, T., Bandara, A., Levine, M., and Nuseibeh, B. Talking about Security with Professional Developers. *Workshop on Conducting Empirical Studies in Industry - CESSER-IP*, IEEE Computer Society (2019).

[111] Lopez, T., Weir, C., Cooper, H., et al. Motivating Jenny Developer Security Toolkit. 2020. https://doi.org/10.21954/ou.rd.c.4957223.v1.

[112] McGraw, G. *Software Security: Building Security In*. Addison-Wesley Professional, 2006.

[113] Merrill, N. Security Fictions : Bridging Speculative Design and Computer Security. *Designing Interactive Systems Conference - DIS*, ACM (2020).

[114] Microsoft. Microsoft Secure Development Lifecycle. https://www.microsoft.com/en-us/sdl/.

[115] Microsoft. Microsoft Security Intelligence Report, Volume 23. 2018. https://info.microsoft.com/rs/157-gqe-382/images/en-us_cntnt-ebook-sir-volume-23_march2018.pdf.

[116] Moolenburgh, W. and Provenmodels. Gods of Management. 2019. https://www.provenmodels.com/8/gods-of-management/charles-b.-handy.

[117] Nadi, S., Krüger, S., Mezini, M., and Bodden, E. Jumping Through Hoops: Why do Java Developers Struggle With Cryptography APIs? *International Conference on Software Engineering - ICSE*, IEEE (2015).

[118] Naiakshina, A., Danilova, A., Tiefenau, C., Herzog, M., Dechand, S., and Smith, M. Why Do Developers Get Password Storage Wrong? A Qualitative Usability Study. *Conference on Computer and Communications Security - CCS*, ACM Press (2017), 311–328.

[119] Naiakshina, A., Danilova, A., Tiefenau, C., Herzog, M., Dechand, S., and Smith, M. Why Do Developers Get Password Storage Wrong? *Conference on Computer and Communications Security - CCS*, ACM Press (2017), 311–328.

[120] Naqvi, S.A.A. The Grounded Incident Fault Theories (GIFTs) Method. 2014.

[121] Natural Language Processing Group from University of the Republic Uraguay. Fast Computation of the Krippendorff's Alpha Agreement Measure in Python. 2020. https://github.com/pln-fing-udelar/fast-krippendorff.

[122] Nayak, K., Marino, D., Efstathopoulos, P., and Dumitraş, T. Some Vulnerabilities Are Different Than Others: Studying Vulnerabilities and Attack Surfaces in the Wild. *International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, (2014).

[123] Nguyen, D.C., Wermke, D., Backes, M., Weir, C., and Fahl, S. A Stitch in Time: Supporting Android Developers in Writing Secure Code. *Conference on Computer and Communications Security - CCS*, ACM (2017).

[124] O'Brien, R.M. The Use of Pearson's with Ordinal Data. *American Sociological Review 44*, 5 (1979), 851–857.

[125] Oates, B.J. *Researching Information Systems and Computing*. Sage, 2005.

[126] Oltrogge, M., Derr, E., Stransky, C., et al. The Rise of the Citizen Developer: Assessing the Security Impact of Online App Generators. *Symposium on Security and Privacy - S&P*, IEEE (2018), 634–647.

[127] Ben Othmane, L., Ranchal, R., Fernando, R., Bhargava, B., and Bodden, E. Incorporating Attacker Capabilities in Risk Estimation and Mitigation. *Computers & Security 51*, (2015), 41–61.

[128] OWASP. Mobile Security Project - Top Ten Mobile Risks. https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Top_Ten_Mobile_Risks.

[129] OWASP Foundation. *OWASP Code Review Guide Book*. OWASP Foundation, 2008.

[130] Pal, S. The Assumptions in Linear Correlations. *Helpful Stats*, 2017. https://helpfulstats.com/assumptions-correlation/.

[131] Perera, C., McCormick, C., Bandara, A.K., Price, B.A., and Nuseibeh, B. Privacy-by-Design Framework for Assessing Internet of Things Applications and Platforms. *Conference on the Internet of Things - IoT*, (2016), 83–92.

[132] Pfleeger, S.L., Sasse, M.A., and Furnham, A. From Weakest Link to Security Hero: Transforming Staff Security Behavior. *Journal of Homeland Security and Emergency Management 11*, 4 (2014), 489–510.

[133] Pirsig, R.M. *Zen and the Art of Motorcycle Maintenance: An Inquiry into Values*. Random House, 1999.

[134] Poller, A., Kocksch, L., Türpe, S., Epp, F.A., and Kinder-Kurlanda, K. Can Security Become a Routine? A Study of Organizational Change in an Agile Software Development Group. *Conference on Computer Supported Cooperative Work - CSCW*, ACM (2017), 2489–2503.

[135] Ponemon Institute. The State of Mobile Application Insecurity. 2015. https://securityintelligence.com/mobile-insecurity/.

[136] Presser, S., Couper, M.P., Lessler, J.T., et al. Methods for Testing and Evaluating Survey Questions. *Public Opinion 68*, 1 (2004), 109–130.

[137] QSR International. NVivo. https://www.qsrinternational.com/nvivo/nvivo-products.

[138] Qualtrics. Qualtrics Survey Service. https://www.qualtrics.com/.

[139] Rashid, A., Levine, M., Nuseibeh, B., Petre, M., Towse, J., and Tun, T. The Johnny Project. 2017. https://www.writingsecuresoftware.org/.

[140] Reed, J. *Appreciative Inquiry: Research for Change*. Sage, 2006.

[141] Riemenschneider, C.K., Hardgrave, B.C., and Davis, F.D. Explaining Software Developer Acceptance of Methodologies: A Comparison of Five Theoretical Models. *IEEE Transactions on Software Engineering 28*, 12 (2002), 1135–1145.

[142] RiskBased Security. Mid Year Data Breach Report. 2019, 1–14. https://pages.riskbasedsecurity.com/hubfs/Reports/2019/2019 MidYear Data Breach QuickView Report.pdf.

[143] Rogers, E.M. *Diffusion of Innovations*. Simon and Schuster, 2010.

[144] Rumsey, D. *Statistics II for Dummies*. Wiley, Indianapolis, 2009.

[145] Sadeghi, A., Bagheri, H., Garcia, J., and Malek, S. A Taxonomy and Qualitative Comparison of Program Analysis Techniques for Security Assessment of Android Software. *IEEE Transactions on Software Engineering 43*, 6 (2016).

[146] SANS Institute. SANS Institute Security Resources. https://www.sans.org/security-resources/.

[147] Santos, P. and Travassos, G. Action Research Use in Software Engineering: An Initial Survey. *Symposium on Empirical Software Engineering and Measurement - ESEM*, IEEE (2009), 414–417.

[148] Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., and Sommerlad, P. *Security Patterns: Integrating Security and Systems Engineering*. John Wiley & Sons, 2005.

[149] Security and Privacy Research Lab University of Washington. The Security Cards. 2013. https://securitycards.cs.washington.edu/.

[150] Senarath, A. and Arachchilage, N.A.G. Why Developers Cannot Embed Privacy into Software Systems? *Conference on Evaluation and Assessment in Software Engineering - EASE*, ACM (2018), 211–216.

[151] Senarath, A.R. and Arachchilage, N.A.G. Understanding User Privacy Expectations: A Software Developer's Perspective. *Telematics and Informatics 35*, 7 (2018), 1845–1862.

[152] Shah, A.K., Mullainathan, S., and Shafir, E. Some Consequences of Having Too Little. *Science 338*, 6107 (2013), 682–685.

[153] Sharp, H., Dittrich, Y., and De Souza, C.R.B. The Role of Ethnographic Studies in Empirical Software Engineering. *IEEE Transactions on Software Engineering 42*, 8 (2016), 786–804.

[154] Shostack, A. *Threat Modeling: Designing for Security*. John Wiley & Sons, 2014.

[155] Sindre, G. and Opdahl, A.L. Eliciting Security Requirements with Misuse Cases. *Requirements Engineering 10*, 1 (2005), 34–44.

[156] Singer, L.-G. *Improving the Adoption of Software Engineering Practices Through Persuasive Interventions*. Amazon, 2013.

[157] Stack Overflow. Annual Developer Survey 2016. https://insights.stackoverflow.com/survey/2016.

[158] Stack Overflow. Developer Survey Results 2011-2019. 2019. https://insights.stackoverflow.com/survey.

[159] Statistica. Total Numbers of Programmers and Software Development Professionals in the United Kingdom from 2011 to 2018. 2018. https://www.statista.com/statistics/318818/numbers-of-programmers-and-software-development-professionals-in-the-uk/.

[160] Steel, C., Nagappan, R., and Lai, R. *Core Security Patterns*. Prentice Hall, Upper Saddle River, NJ, 2006.

[161] Stevens, S.S. On the Theory of Scales of Measurement. *Science 103*, 2684 (1946), 677–680.

[162] Stol, K., Ralph, P., and Fitzgerald, B. Grounded Theory in Software Engineering Research: A Critical Review and Guidelines. *International Conference on Software Engineering - ICSE*, ACM (2015), 120–131.

[163] Strauss, A.L. and Corbin, J.M. *Basics of Qualitative Research*. Sage Newbury Park, CA, 1990.

[164] Such, J.M., Gouglidis, A., Knowles, W., Misra, G., and Rashid, A. Information Assurance Techniques: Perceived Cost Effectiveness. *Computers and Security 60*, (2016), 117–133.

[165] Tahaei, M. and Vaniea, K. A Survey on Developer-Centred Security. *European Workshop on User-Centered Security - EuroUSec*, (2019), 14.

[166] Tavakol, M. and Dennick, R. Making Sense of Cronbach's Alpha. *International Journal of Medical Education 2*, (2011), 53–55.

[167] Tech Partnership. Factsheet: Cyber Security Specialists in the UK. 2017. https://www.tpdegrees.com/globalassets/pdfs/research-2017/factsheet_cybersecurityspecialists_feb17.pdf.

[168] Thaler, R.H. and Sunstein, C.R. *Nudge: Improving Decisions About Health, Wealth and Happiness*. Penguin Books, 2009.

[169] Tietjen, M.A. and Myers, R.M. Motivation and Job Satisfaction. *Management Decision 36*, 4 (1998), 226–231.

[170] Tuma, K., Calikli, G., and Scandariato, R. Threat Analysis of Software Systems: A Systematic Literature Review. *Journal of Systems and Software 144*, February (2018), 275–294.

[171] Türpe, S. The Trouble with Security Requirements. *International Requirements Engineering Conference - RE*, IEEE (2017), 122–133.

[172] Türpe, S., Kocksch, L., and Poller, A. Penetration Tests a Turning Point in Security Practices? Organizational Challenges and Implications in a Software Development Team. *Workshop on Security Information Workers - SIW*, USENIX Association (2016).

[173] Umarji, M. and Seaman, C. Predicting Acceptance of Software Process Improvement. *Workshop on Human and Social Factors of Software Engineering - HSSE*, ACM (2005).

[174] University of Georgia. A PEER Tutorial for Design-Based Research. 2006. http://dbr.coe.uga.edu/enact01.htm.

[175] Ur Rahman, A.A. and Williams, L. Software Security in DevOps: Synthesizing Practitioners' Perceptions and Practices. *Workshop on Continuous Software Evolution and Delivery - CSED*, ACM Press (2016), 70–76.

[176] Vaniea, K. and Rashidi, Y. Tales of Software Updates: The Process of Updating Software. *Conference on Human Factors in Computing Systems - CHI*, ACM (2016), 3215–3226.

[177] Veracode. State of Software Security Report Volume 9. 2018. https://info.veracode.com/report-state-of-software-security-volume-9.html.

[178] Versta Research. How to Estimate the Length of a Survey. https://verstaresearch.com/newsletters/how-to-estimate-the-length-of-a-survey/.

[179] Viera, A.J. and Garrett, J.M. Understanding Interobserver Agreement: The Kappa Statistic. *Family Medicine 37*, 5 (2005), 360–363.

[180] Votipka, D., Stevens, R., Redmiles, E., Hu, J., and Mazurek, M. Hackers vs. Testers: A Comparison of Software Vulnerability Discovery Processes. *Symposium on Security and Privacy - S&P*, IEEE (2018), 374–391.

[181] Wang, F. and Hannafin, M.J. Design-Based Research and Technology-Enhanced Learning Environments. *Educational Technology Research and Development 53*, 4 (2005), 5–23.

[182] Weir, Charles; Knight, Jack; Ford, N. Developer Security Essentials. https://www.securedevelopment.org.

[183] Weir, C. How to Improve the Security Skills of Mobile App Developers: An Analysis of Expert Knowledge. 2017. http://eprints.lancs.ac.uk/id/eprint/84664.

[184] Weir, C. The Agile App Security Game: Leader's Instructions. 2018. https://www.securedevelopment.org/resources/agile-security-game/.

[185] Weir, C., Becker, I., Noble, J., et al. Interventions for Long-Term Software Security Creating a Lightweight Program of Assurance Techniques for Developers. *Software - Practice and Experience*, October (2019), 275–298.

[186] Weir, C., Hermann, B., Stransky, C., Wermke, D., and Fahl, S. Public Dataset from Online Android App Developer Survey. 2019. https://dx.doi.org/10.17635/lancaster/researchdata/319.

[187] Weir, C., Kayla Friedman, and Malcolm Morgan. GitHub: Word Template for a Lancaster University Thesis. https://github.com/charlesweir/LUThesisTemplate.

[188] Weir, C., Noble, J., and Rashid, A. Challenging Software Developers: Dialectic as a Foundation for Security Assurance Techniques. *Journal of Cybersecurity*, (2020), 30.

[189] Weir, C. and Penrillian. *Penrillian's Secure Development Process*. 2014.

[190] Weir, C., Rashid, A., and Noble, J. Reaching the Masses: A New Subdiscipline of App Programmer Education. *Symposium on the Foundations of Software Engineering Proceedings: Visions and Reflections - FSE*, ACM (2016).

[191] Weir, C., Rashid, A., and Noble, J. Early Report: How to Improve Programmers' Expertise at App Security? *Workshop on Innovations in Mobile Privacy and Security - IMPS*, CEUR-WS.org (2016), 49–50.

[192] Weir, C., Rashid, A., and Noble, J. How to Improve the Security Skills of Mobile App Developers: Comparing and Contrasting Expert Views. *Workshop on Security Information Workers - SIW*, USENIX Association (2016).

[193] Weir, C., Rashid, A., and Noble, J. I'd Like to Have an Argument, Please: Using Dialectic for Effective App Security. *European Workshop on Usable Security - EuroUSEC*, Internet Society (2017).

[194] Werlinger, R., Hawkey, K., Botta, D., and Beznosov, K. Security Practitioners in Context: Their Activities and Interactions with Other Stakeholders within Organizations. *International Journal of Human Computer Studies 67*, 7 (2009), 584–606.

[195] Wermke, D., Reaves, B., Huaman, N., Traynor, P., Acar, Y., and Fahl, S. A Large Scale Investigation of Obfuscation Use in Google Play. *Annual Computer Security Applications Conference - ACSAC*, ACM (2018), 222–235.

[196] Whyte, W.F. *Participatory Action Research*. Sage Publications, Inc, 1991.

[197] De Win, B., Scandariato, R., Buyens, K., Grégoire, J., and Joosen, W. On the Secure Software Development Process: CLASP, SDL and Touchpoints Compared. *Information and Software Technology 51*, 7 (2009), 1152–1171.

[198] Witschey, J., Xiao, S., and Murphy-Hill, E. Technical and Personal Factors Influencing Developers' Adoption of Security Tools. *Workshop on Security Information Workers - SIW*, (2014), 23–26.

[199] Xiao, S., Witschey, J., and Murphy-Hill, E. Social Influences on Secure Development Tool Adoption: Why Security Tools Spread. *Conference on Computer Supported Cooperative Work - CSCW*, ACM (2014), 1095–1106.

[200] Xie, J., Lipford, H.R., and Chu, B. Why Do Programmers Make Security Errors? *IEEE Symposium on Visual Languages and Human Centric Computing*, (2011), 161–164.

[201] Xiong, W. and Lagerström, R. Threat Modeling – a Systematic Literature Review. *Computers and Security 84*, (2019).

[202] Yang, X.L., Lo, D., Xia, X., Wan, Z.Y., and Sun, J.L. What Security Questions Do Developers Ask? A Large-Scale Study of Stack Overflow Posts. *Computer Science and Technology 31*, 5 (2016).

[203] Yoder, J. and Barcalow, J. Architectural Patterns for Enabling Application Security. *Conference on Pattern Languages of Programs - PLoP*, (1997), 31.

[204] Yoshioka, N., Washizaki, H., and Maruyama, K. A Survey on Security Patterns. *Progress in Informatics*, 5 (2008), 35–48.

[205] Yskout, K., Scandariato, R., and Joosen, W. Do Security Patterns Really Help Designers? *International Conference on Software Engineering - ICSE*, IEEE (2015), 292–302.

[206] OWASP Foundation. https://www.owasp.org/index.php/Main_Page.

[207] Building Security In Maturity Model | BSIMM. https://www.bsimm.com/.

[208] Wikipedia: Alec Muffet. https://en.wikipedia.org/wiki/Alec_Muffett.

Appendix A. **Most Cited DCS Publications**

The following table shows the most frequently cited Developer Centred Security papers. It gives the identifier used in Figure 3, Section 2.1.2; the paper type (C for conference paper, J for journal paper, and B for professionally-edited book); the reference, the Google Scholar citation count at 12 March 2020, and a calculation of the corresponding annual citation rate.

ID	Title	Type	Ref	Cites	Cites p.a.
Acar+16	You Get Where You're Looking For: The Impact of Information Sources on Code Security	C	[3]	127	32
Acar+16a	You Are Not Your Developer, Either: A Research Agenda for Usable Security and Privacy Research Beyond End Users.	C	[4]	43	11
Acar+17	Security Developer Studies with GitHub Users: Exploring a Convenience Sample.	C	[5]	38	13
Acar+17a	Developers Need Support, Too: A Survey of Security Advice for Software Developers	C	[6]	32	11
Anderson08	Security Engineering: A Guide to Building Dependable Distributed Systems	B	[9]	3574	298
Assal&Chiasson18	Security in the Software Development Lifecycle.	C	[13]	25	13
Ayewah+08	Using Static Analysis to Find Bugs	J	[15]	422	35
Balebako&Cranor14	Improving App Privacy: Nudging App Developers to Protect User Privacy.	J	[19]	63	11
Balebako+14	The Privacy and Security Behaviors of Smartphone App Developers.	J	[20]	82	14
Christakis&Bird16	What Developers Want and Need from Program Analysis: An Empirical Study	C	[34]	75	19
Derr+17	Keep Me Updated: An Empirical Study of Third-Party Library Updatability on Android.	C	[45]	49	16
Devanbu&Stubble-bine00	Software Engineering for Security: A Roadmap.	C	[46]	579	29
Fahl+13	Rethinking SSL Development in an Appified World	C	[57]	144	21
Faily&Flechais11	Persona Cases: A Technique for Grounding Personas.	C	[58]	97	11
Felderer+16	Security Testing: A Survey.	J	[59]	69	17
Fischer+17	Stack Overflow Considered Harmful? the Impact of Copy&Paste on Android Application Security	C	[60]	95	32
Green&Smith16	Developers are Not the Enemy!: The Need for Usable Security APIs	J	[76]	82	21
Hilton+17	Trade-Offs in Continuous Integration: Assurance, Security, and Flexibility	C	[85]	70	23
Howard+09	24 Deadly Sins of Software Security: Programming Flaws and How to Fix Them	B	[88]	200	18
Johnson+13	Why Don't Software Developers Use Static Analysis Tools to Find Bugs?	C	[92]	348	50
Nadi+15	Jumping Through Hoops: Why do Java Developers Struggle with Cryptography APIs	C	[117]	99	20
Naiakshina+17	Why Do Developers Get Password Storage Wrong?	C	[119]	39	13

ID	Title	Type	Ref	Cites	Cites p.a.
Nguyen+17	A Stitch in Time: Supporting Android Developers in Writing Secure Code.	C	[123]	39	13
Perera+16	Privacy-by-Design Framework for Assessing Internet of Things Applications and Platforms.	C	[131]	47	12
Schumacher+05	Security Patterns: Integrating Security and Systems Engineering	B	[148]	868	58
Shostack14	Threat Modeling: Designing for Security	B	[154]	462	77
Sindre&Opdahl05	Eliciting Security Requirements with Misuse Cases	J	[155]	1235	82
Steel+06	Core Security Patterns	B	[160]	328	23
Tuma+18	Threat Analysis of Software Systems: A Systematic Literature Review	J	[170]	22	11
Votipka+18	Hackers vs. Testers: A Comparison of Software Vulnerability Discovery Processes.	C	[180]	19	10
Xiao+14	Social Influences on Secure Development Tool Adoption: Why Security Tools Spread.	C	[199]	62	10
Xiong&Lagerström19	Threat Modeling – a Systematic Literature Review	J	[201]	11	11
Yang+16	What Security Questions Do Developers Ask? A Large-Scale Study of Stack Overflow Posts.	J	[202]	38	10
Yoder&Barcalow98	Architectural Patterns for Enabling Application Security	C	[203]	453	21
Yoshioka+08	A Survey on Security Patterns	J	[204]	229	19

Appendix B. Expert Survey Interview Questions

Introduction – establish context

What is your current role, and what do you find yourself doing day-to-day? Tell me about a typical day at work?

Briefly, how did you first get involved with secure software development?

Exploration

What's your interest in security? What do you do about it, and how do you deal with it day-to-day?

What do you want to achieve when you're helping a team improve software security? How do you define and measure success?

What is the most successful intervention technique you've found? Where do you concentrate your efforts?

Can you think of a particular triumph in your work – where you've worked with a team that has improved their security? How did you achieve that?

Have any of your teams used code checking tools? How happy were you with their effectiveness at finding problems; and their ease of use?

What do you find effective as motivation for secure development?

How do you frighten developers into security, or emphasise the positive aspects?

To what extent are laws and standards helpful in getting teams to be effective at software security? How do you find out about them and keep up to date?

When new people join an existing team, how do you motivate them and how do they learn what's required? Do you encourage double checking of contributions from new people or treat them 'as usual'?

What are the best ways you've found to get teams to tackle specific things:

- Security coordination with other teams?
- Reviews and penetration testing?
- Designing to get feedback from the users?
- What else?

Have you had a nightmare scenario? Or consider this nightmare scenario. You're working with a team that's just learned they have a security flaw in a website that's very heavily used. Have you even had a situation like that (no details required)? What did or would you do to help the team tackle it?

Vision

Let's imaging we're a few years in the future, and the problem of getting teams up to speed with app security has been licked; it's now a part of everyday software development life. How was it done? What were the first small steps?

Clarification (as appropriate)

And how did you achieve that?
Oh, I see. Could you give an example?

Appendix C. **Assurance Technique Names Used**

Such et al.	**Expert Survey Analysis**	**Expert Survey Chapter**	**Developer Survey Questions**	**Magid 1 Analysis**	**Magid 1 Chapter**	**Magid 2 Analysis**	**Magid 2 Chapter**
Configuration Review	Plugin reviews	Configuration Review	tool to scan for lib… vulnerabilities	Component choice	Configuration Review	Configuration Review	Configuration Review
Source Code Review	Code review	Source Code Review	Code review by someone other …		(explicitly omitted)	Code Review	Code Review
Penetration Tests	Penetration testing	Penetration Testing	Penetration testing		(explicitly omitted)	Penetration Testing	Penetration Testing
Vulnerability Scan	Code review tools	Automated Static Analysis	Scanning code with an automatic .. tool	Automated code review	Automated Static Analysis	Automated Static Analysis	Automated Static Analysis
Threat Assessment	Threat modelling	Threat Assessment	Producing a threat assessment…	Threat modelling	Threat Assessment	Threat Assessment	Threat Assessment
Red Teaming	Red teaming	(with Pen. Testing)					
	Stakeholder relations	Stakeholder Negotiation		Business interaction	Stakeholder Negotiation	Product Negotiation	Product Negotiation
	Incentivisation talk					Incentivisation Session	(with Further Workshops)
	Checklists			Checklists		Standardisation	Standardisation
				Continuous reminder		Follow up Sessions	
	Developer training	On-the-job Training		On-the-job Training	On-the-job Training	On-the-job Training	On-the-job Training
						Automated Pen Testing	Automated Pen. Testing
	Security champion	(with On-the-job Training)		Incentivisation Session		Security Champion	Security Champion
						Contingency Plan	Contingency Plan
						Further Workshops	Further Workshops

Appendix D. Agile App Security Game Facilitator Instructions

These are an abbreviated version of the full instructions. References to documents and PDFs refer to the full set of materials [184].

Preparation in Advance

1. Calculate how many players you'll have, and so how many teams of 3-6 players you'll need.
2. Print the two side PlayerInstructions.pdf sheet, one per player, preferably 2-sided.
3. Print from CardsFourToPage.pdf a set of task cards for each team, on A4 paper, one sided and ideally on light card (black and white is fine, colour better). Then cut them out, preferably with a guillotine (or see below).

Setup on the Day

4. Arrange the room with separate tables with 2-6 chairs around each table. Make there's a piece of blank paper and pen for each team.
5. Set up a projector/display visible to all the players, showing the presentation.
6. Sort out cards in advance into individual mini-packs of a set of each (A, B, C, or D), with cover. A packs have 8 cards; B, C, and D have 4 cards.

The workshop has an introduction, up to four rounds, or 'sprints', and a wrap-up session.

Table 27 suggests a timetable for a typical 90 minute session. Adjust timings according to how the teams are getting on in each step.

Figure 56: Playing the Agile App Security Game

Introduction (5 min)

Organise the players into teams of 2-6 people, each team sitting at chairs around a table, as shown in Figure 51. Give each player a set of Player Instructions.

Tell the participants that they are taking the role of agile product managers for the MoneyZoom product; their role is to decide on the stories for the development team to tackle each sprint. In the first couple of sprints each team will be able to complete 11 story points. Stories chosen from previous sprints will remain part of the product and available to mitigate attacks. Stories not chosen in a given sprint will remain on the backlog as candidates to be chosen in later sprints.

The workshop then proceeds in 'sprints'. Each sprint is as follows.

Table 26: When to Give Out the Cards

A	Start of sprint 1
B	Start of sprint 2
C	After the team selected PT (Penetration Testing)
D	After the team selected RA (Review of App Code)

Sprints 1 – 4 (10 – 25 minutes each)

1. Hand out the cards to each team as shown in Table 26. (there may be more one set of cards to hand out). The players then select their tasks to carry out. Note that you don't take cards away at any point – cards that haven't been used remain in the 'backlog' for future sprints.
2. Allow the players time to discuss and decide. Typical timings are as in Table 27, but depend on the teams. Remember the point of the game is the discussion, not to win; if there's a good deal of discussion taking place and time permits, allow the teams longer.
3. When they've selected the cards, get each team to write down the two letter IDs for each card they've selected on the sheet of paper.
4. Then show the attacks for the corresponding sprint on the presentation (or read them out from the Attacks and Mitigations document). The teams see which attacks succeeded on them. Note that card sets C and D do NOT correspond to sprints 3 and 4, but instead are given out after the team choses particular activities.

Wrap-up (10 min)

Ask the teams how they did and what they feel they learned from it.

Table 27: Suggested Timetable

Time taken	Activity
5 m	Introduction
5 m	Familiarisation with context and rules
15 min	Sprint 1
25 min	Sprint 2
15 min	Sprint 3
10 min	Sprint 4
10 min	Learning and sharing

Appendix E. **Threat Assessment and Sales Instructions**

This appendix gives abbreviated facilitator instructions for the second two workshops, using some of the techniques identified from the giving for Group K. Section 8.6.9 provides illustrations.

Workshop Setup in Advance

1. Ensure there is a project for each participant to discuss. Each project should have at least 3 participants who know it well enough to discuss possible security problems. For participants with no such shared projects (or who have reasons not to discuss them), use the 'MoneyZoom' project from the previous Agile App Security Game workshop, and ensure each such participant has a printed description available from the previous workshop.
2. Arrange tables for participants so each table has 3-6 participants, all familiar with one project.
3. Place a large number of various coloured post-it notes and appropriate (sharpie-style) pens on each table.
4. Set up whiteboards and flip charts (one for each project to be discussed the participants and one extra).
5. Have a few (whiteboard/flipchart) marker pens in two contrasting (allowing for colour blindness) colours. E.g. Black and red, or blue and red.
6. Locate a timing device (e.g. mobile phone, physical timer, or Google Search "countdown timer").

Running the Threat Assessment Workshop

Adjust the timings according to how the teams are getting on in each step.

Step 1: Ideation (30 min)

Arrange the participants in groups around tables so that each group can discuss a single project.

Write up on a board/flipchart/screen:

> **Who might do what bad thing to whom?**
> **Person – Thing – Reason**

Ask the groups to think about their project, and 'ideate' answers, writing down the person-thing-reason combinations on the post-its. Encourage everyone to write (duplicates will be combined in the next step).

Remind them that not every bad thing is malicious; sometimes security or privacy issues happen due to accident or misunderstandings. Encourage them to discuss the issues, and to gather as many completed post-its ('threats') as they reasonably can.

Step 2: Organisation (15 min)

Assign a whiteboard/flipchart to each project. Have the table participants all bring the post-its to their corresponding board; and, working together, cluster them into 'topics', posting related ones close together and duplicates on top of each other.

Step 3: Evaluation (15 min)

This step identifies the most important threats.

Ask each participant to pick the threats (post-its) that are (a) most likely, and (b) most damaging. And to use the marker pen to make a dot of one colour (black, say) next to the each of the three they consider most likely; and a dot of the other colour (red, say) next to the three they think most damaging.

When they're done, take a photo of each of the resulting boards, for reference.

Step 4: Summary (10 min)

This requires only a few people; it might be best done in the interval before the next workshop.

For each project, create a three by three 'Risk-Impact' grid on a board/flipchart page, labelled with 'Low/medium/high impact' along the top and 'Likely/fairly unlikely/unlikely' down the left. If you need to use the same board, move the post-its out of the way and use the photo as the record of the numbers of dots.

Get the Technical Leads or a responsible couple of people from each project to use the dots next to each post-it to position a selection of the threats on the grid. A dozen or so is a good number to aim for.

The threats towards the top right are the most important ones to consider in the corresponding projects.

Threat Sales Setup in Advance

From the matrix of threats created in the previous workshop, select the most important 3-5 to discuss. There should be at least three participants to discuss each one; if participant numbers are limited, use a smaller number and have each participant discuss more than one (which will take longer).

Put the corresponding post-its widely separated on the walls of the room (perhaps one on each wall).

Running the Threat Sales Workshop

The timings are approximate. Adjust them according to how the teams are getting on in each step.

Step 1: Selection (5 min)

Have the participants form groups next to the post-its according to their preferences for the threats they each would like to discuss, arranging themselves so that appropriate numbers are next to each one.

Step 2 Finding Mitigations (30 minutes):

Ask the participants to think ways to address the threat (a 'mitigation', in security jargon), ideally with an idea of the effort required, or a means to discover what effort is required. This only needs to be sketchy.

Display the checklist from https://www.securedevelopment.org/handbook/checklist/, and invite the participants to consider them as possibilities.

Step 3: Promoting the Solutions (30 minutes)

After a suitable break…

Tell the participants that the mission for each group, then, is to make a case for product management to address the threat.

Specifically, we want to know the ***positive*** benefit to the organisation of addressing the threat. This often requires some ingenuity.

Ask the participants to appoint a recorder to produce a poster from the conclusions.

Step 4: Presentation (5 minutes per group)

Ask each group to select a presenter, and have each present the selected poster to all the groups together. Record the poster contents and notes of the presentations for reference later.

Appendix F. **Magid Trials Entry Interview Questions**

Introduction – establish context

- What is your current role, and what do you find yourself doing day-to-day? What's your involvement with this project?

Exploration

- Have you considered security for this project yourself? What's been done so far?
- In what ways do you consider security important for this product?

Experience

- What's the last time you came across a security issue in a project? Can you describe the issue?
- How did you deal with that issue?
- How confident are you about that solution?

Vision

- Let's imagine the project's finished, and it's been an excellent piece of work. What do you feel you'll have done related to security and privacy to get it that way?

Clarification (as appropriate)

- Oh, I see. Could you give an example?

Appendix G. **Magid Trials Exit Interview Questions**

Introduction – establish context

- Now that we've been working together for a while, this is a discussion to see how things have progressed in the project.

Exploration

- What do you think has changed?
- What are your feelings about the change in the project?
- What did you make of the three activities we did: game, workshop, follow-ups?
- In what way might you have a better story on security now?

Experience

- What changes did you make as a result of the workshops and discussion?
- What exactly did you do?
- How did you go about implementing the changes?
- Why you chose to do those things?
- What is it that's better now as a result?
- Would you do something similar again?
- What would you do differently?
- How does this relate to these specific threats you've identified (from the threat modelling workshop)?

Vision

- Let's imagine there's a team starting a similar project now, and you're advising the team coming in to help them improve their security. What would you recommend that's the same as we did, and how would you recommend improving it?

Appendix H. **Online Survey Invitation Email**

The tags '`${something}`' are Qualtrics macros, and are self-explanatory.

Participate in a Scientific Survey - Android App Development

Greetings,

As the publisher of the Android App, ${e://Field/AppName}, your experience is valuable to help others.

Please would you consider completing a ${l://SurveyLink?d=survey} about how you do your app development, for a research project by the Universities of Lancaster, Hannover and Paderborn? Your responses will help developers in future to produce better apps – and we'd love you to complete the survey whatever your role. If you are not on the app development team, please would you forward this email to a lead developer?

Your survey responses will be held in the strictest confidence, and nothing that can identify you or your app will be shared outside the research team. There's no payment, and if you choose not to take part, we shall not email you again; but the survey should take no more than ten minutes to complete, and if you want we shall be happy to share the results with you. To find out more about the project, please take a look at the survey information sheet, or email us at developersurvey@lancaster.ac.uk.

Please click here to start the survey: ${l://SurveyLink?d=Take the Survey}

Thank you,

Charles Weir
Secure Development Researcher
Lancaster University
http://www.lancaster.ac.uk/security-lancaster/enterprise/projects/secure-development/
${l://OptOutLink?d= }

Appendix I. **Online Survey Questions**

The following are the survey questions. Some questions were skipped if appropriate (marked with *). The answer formats are abbreviated as follows:

YN	Yes or No
SS	Single Selection.
MS	Multiple Selection
LSS	Likert-Style Scale: Extremely, though to Not at all.
0-100	Slider selecting an integer
N	Integer

In addition, '?' indicates an 'I don't know' option, and 'O' an 'Other' option, where the participant could enter open text. In Q10 and Q21, the option descriptions give the encodings used in Appendix J.

Q1-Q3 were text-only statements.
Q4 Are you working in a team with others, such as developers, testers, project managers? [YN]
Q5* What is your role? [SSO?]
    Programmer, Tester, Project Manager, Non-Specific
Q6* What other roles apart from yourself are there in your team? [MS?]
    Programmer, Tester, Project Manager, Non-Specific
Q7* About how many people (including developers, project managers, testers) are there in your team? [N]
Q8 Please select all the ways you use to develop Android apps [MSO]
    Native Java, JavaScript, C#, Dart, Python, Kotlin, Lua, Native C++
Q10 How often did you release a new version of your app over the past two years? Please give your best estimate; if you have more than one app, please answer for that app that was most frequently updated. [SS]
    Never (0), Annually (1), Quarterly (4), Monthly (12), More frequently (24)
Q11* Over the last one to two years, what content has been in your app updates (%)?
    New features [0-100]
    Non-security bug fixes [0-100]
    Security bug fixes [0-100]
    Third party library updates [0-100]
    Regular maintenance and refactoring [0-100]
Q12 How important is each of the following for your app(s)?
    Runs on many different devices [LSS]
    Secure against malicious attackers [LSS]
    Protects users' privacy [LSS]
    Easy to use [LSS]
    Supports many features [LSS]
    Runs smoothly [LSS]
Q13 How important is security for sales? [LSS]
Q14 How knowledgeable do you consider yourself about information security? [LSS]

Q15 Does your app development ever get support from professional security experts? [YN?]
Q16* Who are these professional security experts (on team/external)? [SS]
Q17* What support do you get from them? Please select all that apply [MSO]

Penetration testing	Security training
Audits	Design reviews
Working on team	I don't know

Q18* About how often do you get support from them? [SS?]
Continuously, Weekly, Monthly, Quarterly, Yearly
Q19 Which of the following have led to changes in the security of your app(s) in the past one to two years? [MSO]
Decision from management
Security crisis within your organisation
Media coverage about app security
Something bad happening to a competitor
Pressure from a partner company
Drive from product or sales team
Pressure from customers
Developer initiative
GDPR requirements
Something bad almost happening to your organisation
Q20* What changes have you made as a result of GDPR requirements? [MSO]
Addition of popup dialog(s)
Removal of analytics or advertising based on it
Adding or changing privacy policy
Q21 How much do you use each of the following techniques to find security problems? [SS for each:
Every build (4), Every release (3), Once or occasionally (2), Decided not to use (1), Haven't considered it (0).]
Producing a threat assessment for the app
Scanning code with an automatic code review tool
Using a tool to scan for libraries with known vulnerabilities
Code review by someone other than the developer
Penetration testing
Q22 What other techniques do you use (if any)? [O]
23 Do you have a security champion within your team? A security champion -- or security hobbyist -- is a non-expert, who takes a particular interest in security. [YN?]
Q24 For how many years have you been developing Android apps? [N]
Q25 For how many years have you been programming in general (not just for Android)? [N]
Q26 About how many Android apps have you helped develop in total? [N]
Q27 Is developing Android apps your primary job? [YN]
Q28 Have you contributed to an open source project in the past year? [YN]
Q29 To which gender identity do you most identify? [SS]:
Female, Non-binary, Male, Prefer not to say
Q30 What is the main spoken language you use at work? [SS]
English, Chinese, Spanish, Arabic, German, French, Other
Q31 In which country do you currently reside? [SS]

Appendix J. Survey Score Calculations

This section describes how scores were calculated from the survey answers.

Likert-Style Scales were encoded as:

Extremely … (4), Very … (3), Moderately … (2), Slightly …(1), Not … at all (0)

Assurance Technique Score: sum of all five sub-questions of Q21, each encoded as shown.

Developer Knowledge Score: LSS encoding of Q14

Expertise Support Score: as the following table:

Q15: / **Q23:**	**No**	**Yes**
No	0	2
Yes	1	3

Requirements Score: sum of LSS encodings for Q12 (Secure against malicious attackers), Q12 (Protects users' privacy) and Q13

Security Update Frequency Score: This required an Update Frequency Estimate of Q10 encoded as shown multiplied by Q11 (Security bug fixes) and divided by 100. The score was Log (this value plus 1).

Appendix K. **Workshop Styles and Assessments**

	Threat Assessment	**Threat Sales**	**Approach**
D	Energy: Low, then Moderate Failed to find any interest in first project; second had almost too much.	Energy Moderate-High. Saw relevance for customer reports	Flipchart, listening facilitator
E	Energy Low, except from 'moderator', who did most of the talking.	Energy moderate. Saw that order of activities might change, even if every security issue must ultimately be resolved.	Flipchart, facilitator dominates
F	Energy moderate; Constructed a good asset for the future.	Energy low-moderate. Future sale an incentive, but not an immediate issue for the team.	Flipchart, listening facilitator
G	Energy moderate. The issues discussed were topical for several of the participants. Testers not very engaged.	Energy moderate-high: the approach was a good way forward.	Flipchart, listening facilitator, group discussions.
H	Energy moderate. Extensive discussions, but little new findings.	Energy low-moderate. Dominated by H1 who wanted white-paper representations of the USPs.	Flipchart, listening facilitator
I	Energy high. Enthusiasm for this new way of looking at their issues.	Energy moderate. Product management interested in addressing client security demands.	Peer discussion
J	Energy very low. Overlong session, dominated by conversation between moderator and one developer.	Unmemorable. No particular new selling points detected.	Whiteboard, dominant facilitators.
K	Energy high. Very well facilitated, used post-it technique to involve everyone.	Energy high. Used 'chose corner of room' and breakout sessions, presented to rest of team. Some interesting new ideas.	Minimal intervention, framed by facilitators. Specialists did 3x3 for likelihood and impact.