How Software Developers Mitigate their Errors when Developing Code

Bhaveet Nagaria and Tracy Hall

Abstract—Code remains largely hand-made by humans and, as such, writing code is prone to error. Many previous studies have focused on the technical reasons for these errors and provided developers with increasingly sophisticated tools. Few studies have looked in detail at why code errors have been made from a human perspective. We use Human Error Theory to frame our exploratory study and use semi-structured interviews to uncover a preliminary understanding of the errors developers make while coding. We look particularly at the Skill-based (SB) errors reported by 27 professional software developers. We found that the complexity of the development environment is one of the most frequently reported reasons for errors. Maintaining concentration and focus on a particular task also underpins many developer errors. We found that developers struggle with effective mitigation strategies for their errors, reporting strategies largely based on improving their own willpower to concentrate better on coding tasks. We discuss how using Reason's Swiss Cheese model may help reduce errors during software development. This model ensures that layers of tool, process and management mitigation are in place to prevent developer errors from causing system failures.

Index Terms—Human error, human factors, software development.

1 Introduction

The impact software defects have on transport[1], banking[2] and business systems[3] continue to make regular newspaper headlines around the world. At the heart of many software defects is human error [1]. Many of those human errors are made by developers during the coding process [2]. Such coding errors are psychological events [3] (for example, loosing track of where you are in a programming task) which have often been caused by a physical event (for example, being interrupted by a colleague) and can result in a physical fault (for example, omitting to initialise a variable); in software engineering this physical fault may be subsequently executed causing a system failure[4]. Human error is probably endemic in all human-centred activities. Human Error Theory (HET) [3] has been used extensively to analyse errors and develop mitigation strategies in a range of other disciplines, most notably in medicine and related health disciplines (e.g., nursing and pharmacy).

The impact of human issues on coding has been previously investigated [4]. Most studies investigate developer errors indirectly via the analysis of the physical defective code that developers submit to repositories. Such previous studies have typically analysed the fixes that developers have made to repair code from a human perspective (e.g., Kini and Tosun studied the impact of developer experience on defects [5]). Such repository mining has also been used in conjunction with Sentiment Analysis and Natural Language Processing to, for example, understand how the emotional state of developers relates to fault fixes (e.g., Ortu et al. [6] report that sad developers take longer to fix faults than happy developers). Most of these previous studies use quantitative approaches to analysing repository data, but some qualitative studies exist. Previous qualitative studies have investigated the relationship between developers and code faults by, for example, investigating code reviews [7] or studying code comprehension [8].

Few studies have considered errors in terms of the developer's psychological events during a programming task that resulted in physical code faults. The few studies that do focus on the psychological world of the developer include tracking cognitive load via electrical brain waves [9]. We take a simpler and potentially more usable approach by considering the errors that developers make are likely to be similar to those errors that all humans make in 'manual' tasks. We investigate the coding errors and deployed mitigation strategies reported in interviews of 27 professional software developers. We analyse this qualitative data using HET [3].

According to Reason [3], errors are introduced during two phases of human cognition: planning and execution. Mistakes are defined as planning errors. A mistake results from lack of knowledge during the planning stage of an activity. An example of a mistake is misdiagnosing a patient due to lack of experience. There are two types of execution error: slips and lapses. A slip is the result of careless or inattentive actions, for example, fat fingering. A lapse is the result of a failure of memory, for example, intending to do a task but forgetting about it. Slips and lapses are further described by Rasmussen [10] as Skill-based (SB) errors with mistakes described as Rule-based (RB) and Knowledge-based (KB) errors. Figure 1 shows the relationship between error types as described by Reason [3] and Rasmussen [10]. We focus on developer SB errors because such errors are common and are potentially simpler to recognise and

- *B. Nagaria is with Department of Computer Science, Brunel University London.*
- *T. Hall is with School of Computing and Communications, Lancaster University.*

Manuscript received Month xx, 2020; revised Month xx, 20xx.

1. British Airways: https://www.bbc.co.uk/news/uk-40069865
2. TSB bank: https://www.bbc.co.uk/news/technology-52121990
3. Facebook: https://www.bbc.co.uk/news/technology-47562281
4. https://standards.ieee.org/standard/24765-2017.html

mitigate. Prioritising developer SB errors potentially offers an opportunity for us to effectively reduce more coding errors, with future work focusing on RB and KB errors.

Although Human Error was the theme of a promising 2015 ICSE Workshop[5] there remain few studies of how HET can be used in software development. The few studies that do use HET tend to focus on errors in user requirements [11], [12], [13], [14], [15], [16], [17], [18] (on which topic another promising one-off workshop occurred in 2017[6]). One prominent piece of work looking at the use of HET in requirements engineering presents a survey study to understand high level human errors, the related faults and methods of mitigation used by requirements engineers on real projects [16]. In addition, Hu et al. [15] performed an empirical study to determine if knowledge of human errors can serve as a fault prevention mechanism during requirements engineering. Hu et al. [15] also reported that the better a developer was able to use error information to find faults in a requirements document, the less likely that developer was to insert faults into their own requirements.

Huang is the only researcher that we know of who has previously looked at coding errors in relation to HET. Huang [19] looked at requirements-based human errors in relation to coding, concluding that there is a high likelihood that developers will introduce post completion errors while implementing software requirements which include a post completion sub-task. Post completion errors happen when a sub-task is omitted at the end of a task which is not necessary for the completion of the task, for example, omitting to collect your bank card from an Automated Teller Machine (ATM) machine after collecting your cash. Li et al. [20] reported that post completion errors are infrequent but persistent. Huang and Liu [21] presented a defect prevention approach which is human centered using their Defect Prevention Based on Human Error Theories (DPeHE) framework. The results of Huang and Liu's study showed that there are promising avenues for defect prevention beyond conventional software process improvement approaches.

HET seems to have much potential to help understand and devise well-founded mitigation strategies for developer errors. This potential seems to have been under-exploited so far in software engineering. Few studies that investigate the use of HET in software engineering use professional software engineers (the only studies we could find were Huang's studies: [22] involved 14 professionals and [21] involved 20 professionals). HET studies predominately use student participants to investigate human error in requirements engineering.

We build on and extend the small amount of previous work to present insights into the SB human errors that professional developers make, why these errors are made and how developers currently mitigate their own errors. Our study is based on semi-structured interviews with 27 professional software engineers. Although a few studies in software engineering conduct more interviews with software professionals (for example, Hoda et al's [23] study of self-organising agile teams included 58 interviews), most software engineering studies are based on fewer interviews than 27, yet are able to report important and rich human insights into developer behaviour (for example, Smith et al's [24] study on developer behaviour in relation to security vulnerabilities is based on input from five students and five professional developers). In this study, we answer the following research questions:

- *RQ1:* What Skill-based (SB) human errors do professional software developers make while performing software development tasks?
- *RQ2:* How do professional software developers mitigate their Skill-based (SB) human errors?

In answering these research questions we contribute new understanding of the type of SB human errors that developers are making. In particular, we report a range of errors that developers say are due to the complexity of their development environment. We also contribute improved understanding of the developer, management, tooling and process error mitigation strategies developers report reduces their errors. In particular, we report that developers feel that to reduce their errors they must largely resort to improved willpower to concentrate and focus on tasks more effectively. Implementing ways in which developers can be supported to improve their concentration skills seems an important priority.

The rest of this paper is organised as follows. Section 2 presents the background to Human Error Theory and relevant literature. Section 3 describes the research methodology. Section 4 presents the results. Section 5 discusses our findings. Section 6 discusses related work. Section 7 explains the threats to validity. Section 8 presents our concluding remarks.

2 Human Error

In this section, we describe human error classification using the Skill-Rule-Knoweldge Framework (SRK) and we discuss the application of HET within other disciplines.

2.1 Human Error Theory

Rasmussen [10] developed an error framework of human cognitive mechanisms. Rasmussen's SRK framework of human performance in terms of error is formed of three levels. These are Skill-based (SB) level, Rule-based (RB) level and Knowledge-based (KB) level. The SB level focuses on patterns of error associated with well understood tasks. The RB level focuses on addressing recognisable patterns of errors which are controlled by stored rules. The KB level focuses on unknown situations where forthcoming actions must be planned by employing analytical processes and stored knowledge [3]. The SB level focuses on slips and lapses e.g., fat fingering, while the RB and KB levels focus on mistakes which originate from rules or knowledge.

As already mentioned, this study focused on SB errors made by developers. Reason [3] describes eight types of SB errors: omission, repetition, reversal, omission following interruption, double-capture slip, reduced intentionality, perceptual confusion and interference error. Table 1 provides more details of each of the eight types of SB errors that we investigate in this study.

5. http://humanerrorinse.org/workshops/WAHESE15/schedule.htm
6. http://humanerrorinse.org/workshops/WHERE2017/

SB Error Type	Definition	Real World Example
Omission	Omissions are when you conclude the process is further along than it actually is, and, as a consequence, omit a necessary step.	Forgetting to turn the kettle on in the tea making process.
Repetition	Repetitions are when you conclude the process has not yet reached the point where it is further along that it actually is and then repeat an action already done.	Setting the kettle to boil for a second time.
Reversal	Checking something outside a sequence, causing you to double back on the sequence.	I intended to take off my shoes and put on my slippers. I took my shoes off and then noticed that a coat had fallen off a hanger. I hung the coat up and then instead of putting on my slippers, I put my shoes back on again.
Omission following interruption	Forgetting something due to an external event.	I picked up my coat to go out when the phone rang. I answered it and then went out of the front door without my coat.
Double-capture Slips	Unintentionally activating a strongly related action pattern.	I intended only to take my shoes off, but took my socks off as well.
Reduced Intentionality	Some delay intervenes between the formulation of an intention to do something and the time for this activity to be executed.	I opened the fridge and stood there looking at its contents, unable to remember what is was I wanted.
Perceptual Confusion	Repeated tasks become automised. When conducting these automised tasks we accept rough rather than precise approximations for expected inputs. This degradation of criteria leads to perceptual slips.	I intended to pick up the milk bottle, but actually reached out for the squash bottle.
Interference Error	Two active plans, or two parts of a single plan can become entangled.	I had just finished talking on the phone when some visitors were ushered in. I got up from behind the desk and walked to greet them with my hand outstretched saying 'Smith speaking'.

Table 1
Definition and Examples of Skill Based Errors [3]

Leape [25] suggested that human errors are affected by physiological, psychological and environmental factors. Physiological factors include fatigue, sleep loss, alcohol, drugs and illness. Psychological factors include other activity and emotional states e.g., boredom, fear, frustration, anger or anxiety. Environmental factors include noise, heat, visual stimuli and motion. Previous studies have investigated the impact some of these individual psychological and physiological factors have on the coding habits of developers, for example, Graziotin et al. studied the impact of mood and, particularly, happiness on the performance of developers [26]. However we could find no study that investigated these factors holistically within a theoretical framework. Our study looks indirectly across all of these factors within the established Human Error Theory. Such an approach allows us to understand better whether these factors generally affect developers and to prioritise the development of mitigation mechanisms.

2.2 Human Error in Other Disciplines

Fitts and Jones' 1947 study [27] is a classic and influential demonstration of how important it is to analyse human errors. Fitts and Jones conducted a series of individual and group interviews with pilots to collect accounts of and analyse pilot-errors. Fitts and Jones [27] found that the redesign of equipment in accordance with human requirements can eliminate a large number of pilot-error accidents. An example error that Fitts and Jones [27] report is pilots confusing wing flap and landing gear controls. Dekker [28] reported the immediate wartime fix was to attach a rubber wheel to the landing gear control and a small wedge to the flap control. On the basis of their findings, Fitts and Jones [27] recommended that aircraft should provide uniform shape-coding of all control knobs which must be grasped quickly or without looking. This wartime design solution went on to become a certification requirement [28].

Much analysis of human error has occurred in medicine. Leape [25] reported that there are five areas in which errors can be prevented in medicine, these are; reduced reliance on memory, improved information access, error proofing, standardisation and training. Ribeiro et al. [29] identified a variety of slips, lapses and mistakes which are made when nurses use equipment in ICU. Ribeiro et al. [29] found that it is mainly infusion pumps and monitoring systems which involve unfavourable events that harm patient safety. Common mechanisms behind these errors are memory and attention lapses in the handling of infusion pumps; planning failures during programming of the monitors; application of rules and knowledge [29]. To mitigate these errors Ribeiro at al. [29] suggested that daily checks of infusion pumps and monitors should be performed.

Beso et al. [30] investigated drug dispensing errors. They conducted a study in a 450 bed London teaching hospital in which they asked pharmacists to self report any dispensing errors. They also conducted semi-structured interviews with the pharmacists who dispensed the medication to explore why the error occurred. Brumby et al. [31] reported that interruptions are disruptive, take time to recover from and lead to an increased chance of errors being made. Beso et al.'s [30] findings suggested that consideration should be given to how interruptions are handled and that automated dispensing systems may reduce many errors. Hakala et al. [32] also reported the most common reason for drug dosing errors is improper patient identification when a dose is obtained from the pharmacy or when a dose is administered. Hakala et al. [32] reported that introducing a bar code system which links the patient dose with the electronic information reduces the number of crucial points for human error and provides a framework to ensure that the prepared dose reaches the correct patient.

We expect to exploit the approaches used by other disciplines when investigating human error. We also expect to identify mitigation strategies for errors that occur during coding just as other disciplines have done when errors have been framed using Human Error Theory.

3 Methodology

Our aim was to understand professional software developers' experiences of the SB human errors they make during development tasks. To address this aim we used qualitative research in the form of semi-structured interviews with 27 professional software developers. This section describes our approach to collecting and analysing this qualitative data. Section 3.1 describes the participants of this study and how they were recruited. Section 3.2 presents the study design. Section 3.3 details how we performed our data analysis.

3.1 Participants and Recruitment

Our study was conducted over a sixteen week period. During this time we recruited 27 industry software developers. We focused on industry software developers because we wanted to understand SB human errors that occur on the job, as opposed to while learning to develop software.

These industry software developers were a convenience sample who were contacted by word-of-mouth, email and face-to-face. In addition, to this we shared details of the study and issued invitations to participate on social media, with posts on LinkedIn and Twitter.

The participants of our study comprised of four women and 23 men of whom 14 had more than ten years of industry experience. The majority (24 developers) worked only on closed sourced systems. The developers worked in varied organisations ranging from startups to multinationals. They developed software for a variety of industries, examples include finance, healthcare, leisure and telecommunications. They developed software using a variety of languages, examples include C++, Python, PHP and JAVA.

3.2 Interview Method

We used semi-structured interviews to question software developers about their errors during coding. We also asked developers how they mitigated their errors. Although the interviews had predetermined questions, there were also unstructured open ended questions that could be used as follow-ups based on an interviewee's answers and the interviewer's interpretation of those answers.

One interviewer (the first author) conducted all the interviews and so consistency was maintained across all interviews. The interviews were either face-to-face (21), mostly at the developer's workplace, and video calls using Skype (6). The interviews lasted between 30 to 45 minutes. All interviews have been recorded and transcribed.

The structure of the interview was as follows (a visual representation can be seen in our Online Appendix here: https://bit.ly/2ZKgIsj):

1) Introduction and overview of logistics.
2) Iterate through this process for each of the eight SB errors.
    a) Provide an explanation of the SB error including a definition and a non software engineering-related real world example. Details of the eight SB errors can be found in Table 1. Given the diversity of software developmental tasks and a participant's experience we did not want to bias their responses by giving software engineering related examples of each SB error.
    b) Ask about their error occurrences and mitigation strategies for each of the eight SB error types. Sample questions: 'Can you provide examples of an omission in the software development work you perform' or 'For the given example, did you employ any mitigation strategies?'
3) Close, Demographic Data and thanks.

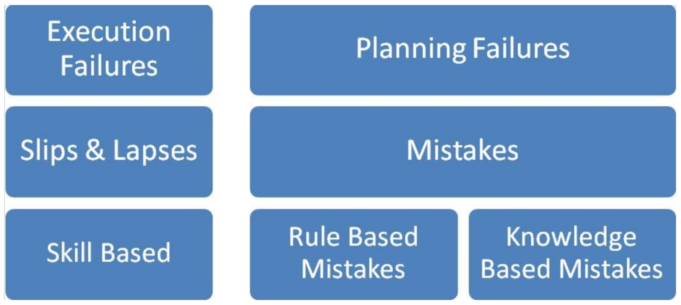

Figure 1. Reason's [3] Slips, Lapses, Mistakes mapped to Rasmussen's [10] Skill-based (SB), Rule-based (RB) and Knowledge-based (KB) error types

3.3 Data Analysis

Below we explain the data analysis process as depicted in Figure 2. Once the interviews were performed, each interview audio recording was then transcribed manually by the first author to allow for coding and analysis.

We identified themes for errors and mitigation strategies by identifying high level groupings of key concepts in the transcripts using Thematic Analysis [33]. Our approach was very similar to that used by Meyer et al. [34] in their analysis of reflective goal setting by developers. We extensively discussed commonalities between the low level themes and noticed that these themes for mitigation strategies related well to the generic high level concepts identified by Graziotin et al. [26]. These high level themes were: management, tools, processes and the developer. Graziotin et al.'s scheme seemed a useful high level clustering of the mitigation strategies themes we extracted and so we adopted this high level structure and organised our low level mitigation strategy themes within it.

Structuring interviews around the eight SB error types meant that all interview data was also already organised into those eight SB error headings.

We open coded each reported error and mitigation strategy based on direct quotes from transcripts against the low level themes established. To ensure that coding was conducted reliably both authors extensively discussed each

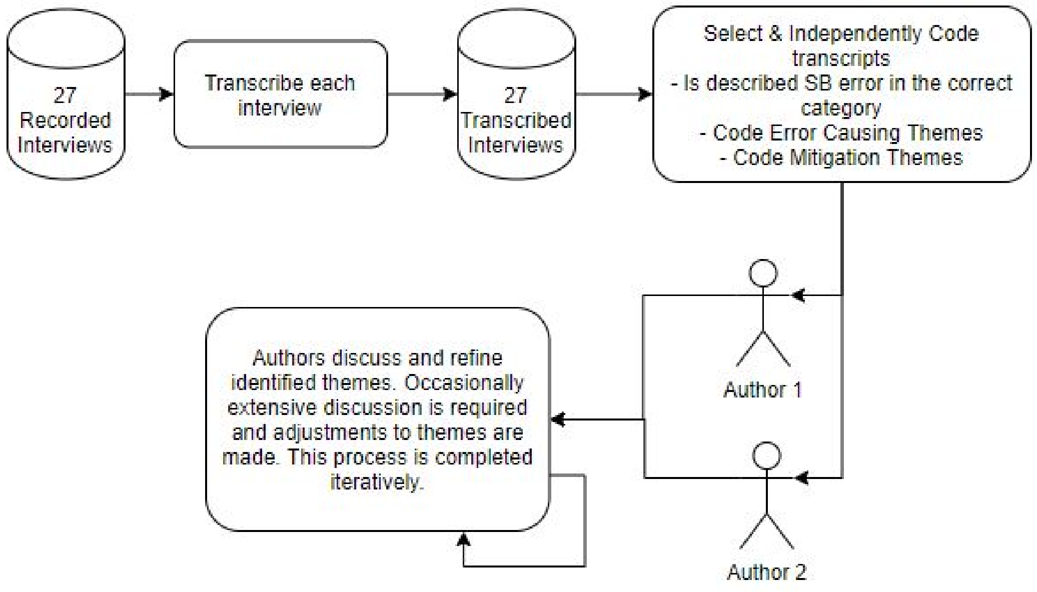
27 Recorded Interviews
Transcribe each interview
27 Transcribed Interviews
Select & Independently Code transcripts
- Is described SB error in the correct category
- Code Error Causing Themes
- Code Mitigation Themes
Author 1
Author 2
Authors discuss and refine identified themes. Occasionally extensive discussion is required and adjustments to themes are made. This process is completed iteratively.

Figure 2. Data Analysis Process

error type prior to performing any coding to ensure their understanding was the same. Then each author independently coded interview transcripts. When coding the transcripts we looked to:

1) Code themes and supporting evidence for a given SB error.
2) Code themes and supporting evidence for a given mitigation strategy.
3) SB Errors and mitigation strategies were coded against one low level theme and not multiple low level themes.

We noticed that the error themes which emerged from the interview data were not all psychological events, i.e., were not errors in the true HET sense. Some of the themes which emerged from the data were the reasons that errors had occurred, while other themes were the consequences of errors having occurred. Although this spread of data across the three aspects of error (reason, error and consequence) was not what we had expected, it seemed to us that what developers reported was nevertheless important. Consequently, we also labelled each developer reported 'error' as either: a reason for error, an error or the consequence of an error.

As recommended by Kitchenham et al. [35] where any disagreement on classification occurred between the authors extensive discussion of the issues ensued and a decision made on a categorisation for that data or an update to the set of themes was made. Errors and mitigation strategies that did not fit well with the current set of themes were discussed extensively and in some cases the set of themes updated. This was an iterative process.

Theme categorisation was documented using a custom kanban style board using Trello[7]. Cards were created with themes to support a SB error type or a mitigation strategy. The cards were assigned labels to show which participant transcript supports the theme. Evidence to support a theme was highlighted on the transcripts and given a reference number. This reference number was added to the appropriate Trello card.

7. https://trello.com/en

4 Results

In this section, we present the findings that arose in the interviews. We present the data for both error and error mitigation in separate sub-sections related to the two research questions posed.

4.1 RQ1: What Skill-based (SB) human errors do professional software developers make while performing software development tasks?

The 27 interviews elicited 57 themes of errors across eight SB error types. Table 3[8] shows the number of error and mitigation strategies broken down by SB error type. Table 3 shows that Perceptual Confusion errors were mentioned most frequently with developers giving examples of errors stemming from being on auto pilot. These autopilot errors relate to a range of error causes (e.g., complexity of developer environment) which we discuss next. Some SB error types (e.g., double capture and interference errors) were not mentioned often (4 and 5 mentions respectively), which may mean that these errors do not happen as often but more research is needed to establish this.

We also considered the errors mentioned by developers in terms of low level error themes as shown in Table 2 with

8. All participants were asked about each SB error type, however, not every participant was able to provide an example, for every SB error type.

Low Level Theme	Explanation	Reason, Error, Consequence	Mentions
Complexity of Development Environment	Too many things going on in development space	R	16
Concentration	Lacking concentration	E	13
Duplications	Repeating tasks	C	10
Requirements Problems	Poor requirements engineering	R	8
Context Switching	Switching tasks	R	6
Rabbit Hole	Going down the rabbit hole while performing a task	C	4
Testing	Not testing as well as they could have / lacking test automation coverage	C	4
Distractions	Variety of distractions e.g., noise	R	4
Work Pressure	Increased work pressure to deliver tasks	R	4
Understanding	Lack of understanding of the task	R	2

Table 2
Developer Error Themes (in ranked order)

Skill Based Error Type	Error Themes	Mitigation Strategy Themes
Omission	9	18
Repetition	8	18
Reversal	6	8
Omission following Interruption	6	23
Double capture slips	4	12
Reduced Intentionality	6	15
Perceptual Confusion	13	13
Interference Error	5	6

Table 3
Number of Theme Occurrences of each Skill Based Error Type

more details of each error theme in Table 2 (which also includes a classification of each theme in terms of whether the theme is an error reason, an error or the consequence of an error). Our results suggest that developers focus more on the reasons for errors and their consequences, than the errors themselves (see Table 2). The psychological event that is the error (e.g. lost concentration) seems no more important to developers than the reasons and consequences of errors. Table 2 suggests developers believe that many errors are as a result of the complexity of the development environment that they work with. In particular, knowledge of multiple languages, multiple tools, multiple views and dependencies between these needs to be maintained and context switched during programming tasks. Complex programming tasks themselves have a significant cognitive load [36], but performing programming tasks within a complex development environment is likely to increase this cognitive load still further. For example, Participant 17 told us that errors occurred when

> '...running some queries or SQL on the database, when you don't realise you are on QA or Live.'

Participant 12 said

> 'typing commands into the wrong window'

caused errors and Participant 10 said that

> '...using a JavaScript based templating language we wrote an if statement with an end condition in and I used the python way of writing the end which meant at the next step the whole thing didn't render.'

Table 2 shows that developers also recognise that a frequent human error is their own concentration being impaired. Issues surrounding concentration, context switching, work pressure, rabbit hole[9], understanding and distractions were cited regularly by developers which suggests that it is easy for developers to lose their awareness of a given task:

- Participant 26
  > 'You go and reply to the email and by the time you come back you have forgotten what you were going to do.'
- Participant 13
  > 'Very often what I found is you can just go down blind alleys, but also you can go into this reversal thing where you can't see why you made a certain change in a certain area of the code...'
- Participant 16
  > 'It is to do with the nature of the speed of when you have to deliver...' '...someone comes in at 5pm and says I need that for tomorrow. So you are under a lot of pressure and that is when a lot of errors can occur.'
- Participant 22
  > '...its like I am jedi, its all flowing out of the fingers and it will be like that for several hours and suddenly I will think oh I am really hungry or oh I need a wee. But it could be anything, it could be like the phone ringing and because you are kind of there in the moment, quite often you have fingers in many different pies all at the same time. And you have got a model you are holding in your head, pulling you out of that flow means that occasionally you drop some of that model on the floor. I mean you usually find it again at bit later, but yeah it has caused some disruption to me in the past.'
- Participant 16
  > '...I do SSRS reports and uploading them is a repetitive task. So you could easily upload the wrong report, if you don't concentrate on what you are doing...'

Duplications are mentioned 10 times by developers as the consequence of an error. Duplicate code is a well-known bad smell so it is interesting that developers recognise duplications as an outcome of human error. Requirements problems also feature in the list of reasons for errors. Requirements problems are well-known as a source of failure

9. Going down a rabbit hole is a metaphor commonly used to indicate someone has gone into a situation or started a process which is particularly difficult, complex or chaotic, especially one that becomes increasingly so as it develops or unfolds

Low Level Theme	Developer	Process	Tools	Management
Focus	✓			
Concentration	✓			
Use Headphones	✓			
Awareness	✓			
Discipline	✓			
Lean	✓			
Checklist		✓		
Code Reviews		✓		
Testing		✓		
Note Taking		✓		
Communication		✓		
Documentation		✓		
Pull Requests		✓		
Automation			✓	
Git			✓	
Compiler			✓	
Where are you			✓	
Planning				✓
Best Practices				✓
Prioritisation				✓
Well formed, low level processes				✓
Awareness				✓
Total Instances	58	57	28	10

Table 4
Mapping High to Low Level Mitigation Strategy Themes

throughout the development process (for example, the London Ambulance System failure [37]), so it is not surprising that developers say these problems underpin some of the errors they make.

Overall our results in response to RQ1 suggest that developers blame their own lack of focus and concentration errors for many faults. With many errors reported by developers to be caused by the complex development environment in which software development occurs.

4.2 RQ2: How do professional software developers mitigate their Skill-based (SB) human errors?

We have classified the mitigation strategies mentioned by developers into four high level themes which are the developer, processes, tools and management. Table 4 presents these high level themes and shows the number of times each theme was mentioned by developers during interviews. Table 4 shows that developers see themselves as highly influential in mitigating errors, suggesting that developers seem to take a great deal of personal responsibility for trying to prevent errors.

Table 5 shows the low level themes of the developer mitigation strategies. Table 5 suggests that a large number of themes relate to the developers' cognitive issues such as focus, concentration, discipline, attention, understanding and awareness. Most of these issues were discussed by developers in terms of them using willpower to improve their coding behaviour. For example, developers said:

- Participant 10

  '...being very disciplined if you know you have a context you know you need to restore...'
- Participant 16

  'So you take responsibility by checking your work to ensure you are filtering out the one record...'
- Participant 12

  '... awareness is there and developers should be aware all the time...'
- Participant 16

  '...focusing on what you are delivering as opposed to meeting the deadline...'
- Participant 12

  'Just by remembering.'

These quotes suggest that developers believe that by being more self controlled they could reduce coding errors. Increased willpower and self discipline is notoriously hard to achieve without structured support. In addition, external factors such as those mentioned previously (e.g., fatigue, illness, boredom, frustration, noise, heat, etc.) can hamper willpower and self discipline.

Although Table 5 shows a variety of developer-based mitigation strategies, it is surprising that reducing tiredness and taking breaks was not explicitly mentioned more frequently by developers. The impact of tiredness on errors seems conventional wisdom. It is unclear why tiredness did not feature more directly in our results. It is clear that tiredness etc. can be a reason for developers to lose their focus on a task [38] so it is surprising that developers did not mention this more often. Similarly it is surprising that interruptions were not mentioned more explicitly by developers. Interruptions are widely thought to underpin errors [39] but were not mentioned much in our study. Bailey and Konstan [40] report that interruptions have a disruptive impact on completion time and error rate. Where interruptions were mentioned it was indirectly, with developers talking about mitigation for errors like using headphones and turning emails off.

Mitigation strategies related to processes were frequently mentioned by developers. Most of these strategies are based on detecting the consequences of errors in terms of spotting faults in code. Table 6 suggests that many process themes are related to getting faster feedback on whether work is likely to contain faults. For example, 'testing', 'code reviews' and 'pull requests' all enable early checking of work products in relation to faults. For example, some developers mentioned:

- Participant 13

  'And the pull request is a deliberate, these are my commits and this is my change set, I'm going to review it myself and someone will also peer review and only then will it become consequential. And that's really important because you will always pick up issues.'
- Participant 19

  '...first of all from a developer point of view any change must be documented. No one is allowed to make any changes before documenting the change and running it by a senior developer first.'

Table 7 suggests that automation is an important element of mitigating developer errors. Developers seem to rely on automation tools or tools that have elements of automation to aid in reducing their errors. For example, some developers mentioned:

- Participant 20

  '...generally the pattern is we have automated something that would have been a human error before and you can't always do that but its nice when you can.'
- Participant 7

  'Usually I stash all of them for instance if I am on some branch, something urgent, imminent comes up and they say you need to fix this one, this is very urgent one.'

Although management did not feature hugely in mitigation strategies, Table 8 suggests that management need to better plan workloads, promote and provide training for developers to use best practises. For example, some developers mentioned:

- Participant 11

  '...to be aware of how you prioritise things and to switch to these new tasks only if it is really urgent. So it takes some effort to do that. Prioritisation'
- Participant 10

  '...we have always had someone from a more product-y perspective or the user or client or whoever, looking at a staging server or a version of the code running, making sure that the functional requirements were being fulfilled.'

The mitigation strategies cited by developers are not very surprising. Developers seem to know what errors they make and what they need to do to mitigate errors and to detect the consequential faults. Despite this knowledge developers seem to struggle to implement these mitigation strategies themselves. Taking personal responsibility for error mitigation may not be the most effective route to reduce errors. Supporting developers in effective error mitigation is likely to depend on better tools, management and processes.

Overall in response to RQ2 our results suggest that developers predominately rely on trying to improve their own willpower to mitigate errors. Developers use a variety of strategies to retain their focus and concentration in order to reduce the number of errors they make. Developers also use mitigation strategies within the development process, often to detect faults (e.g., checklists), tools (e.g., automation) and management (e.g., planning).

Theme	**Explanation**	**Mentions**
Focus	To give special attention to a specific task.	10
Concentration	To think intensely about a specific task.	8
Use headphones	Use headphones to reduce background distractions.	7
Planning	Appropriately plan the task.	6
Awareness	Develop an awareness of the project and task, so you can address potential issues.	5
Discipline	To follow a series of rules or a code of practice.	4
Learn	Learn about the project, software tools, language etc.	3
Other	*Example* Ignore emails when conducting tasks	15

Table 5
Themes of Developer Mitigation Strategies

Theme	**Explanation**	**Mentions**
Checklist	Use checklists to verify process flow has been conducted.	12
Code Reviews	Use code reviews to peer review work prior to committing.	9
Testing	Have better test coverage to ensure more cases are covered.	8
Note taking	Don't rely on memory, keep physical/typed notes to serve as a prompt.	6
Communication	Promote communication within the team.	6
Documentation	Create and use code documentation.	6
Pull Requests	Use pull requests to peer review work prior to committing.	3
Other	*Example* Complete refactoring tasks separately	7

Table 6
Themes of Processes Mitigation Strategies

5 Discussion

Our results suggest that developer-based mitigation strategies are the most frequently reported ways to reduce human errors. We find this interesting as we had believed that developers may have leaned more towards using tool automation or reliance on process based mitigation strategies. Developers seem to think that they individually should stop making errors that get through to production. Other disciplines, for example, health [41], have shown that a human-based approach is likely to have limited success without the embedded support of tools and processes.

The majority of the mitigation strategies cited could be described as psychological, and 'internal' to a developer (e.g., developers saying that they need to concentrate more). Many of the process, tool and management strategies could be described as 'external' to a developer, and mostly focused on detecting consequential physical code faults. These external strategies are likely to aid developers in implementing internal strategies to prevent the error. Figure 4 shows how

Theme	**Explanation**	**Mentions**
Automation	Where appropriate automate routine tasks.	9
Git	Utlise the full power of Git so slip of the finger errors cannot be commited.	5
Compiler	Utilise the power of the compiler to pick up on syntax errors.	4
Navigation Helper	Provide developers a clear indication of where they are within an application(s) and or code base.	2
Other	*Example* Utilise helpers that are ingrained in the IDE	8

Table 7
Themes of Tools Mitigation Strategies

Theme	Explanation	Mentions
Planning	Appropriately planning the workflow of developers.	3
Training	Provide formal and informal training opportunities to developers.	2
Best Practices	Actively encourage and promote the use best practices in all developmental activities.	2
Prioritisation	Appropriately prioritise developmental tasks, so as to minimise interruptions to developers.	1
Well formed, low level processes	Ensure work units are small, measurable and achievable.	1
Awareness	Having a global awareness of the project to ensure tasks like planning and prioritisation can be done correctly.	1

Table 8
Themes of Management Mitigation Strategies

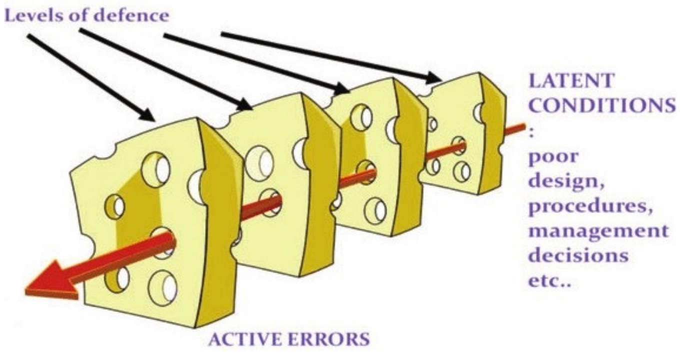
Levels of defence
LATENT CONDITIONS: poor design, procedures, management decisions etc..
ACTIVE ERRORS

Figure 3. James Reasons's Swiss Cheese Model [42] adapted from [43]

the developer is likely to be central to all mitigation strategies. The developer is closely supported by process and tool mitigation strategies and more widely by management mitigation strategies. Indeed we suggest that the commonly mentioned mitigation strategies of using tools (e.g., automation) and processes (e.g., checklists) can support developers to implement internal mitigation strategies (e.g., staying focused).

Our results resonate with Reason's [44] Swiss Cheese model of accident causation. The Swiss Cheese model (shown in Fig. 3) is an approach to building effective organisational defences against failure. The aim of the model is to enable a system-centric rather than a human-centric approach to reducing failures. The model shows that layers of barriers are needed to block errors from slipping through to cause major failures. Reason's Swiss Cheese model has been used extensively to manage the prevention of medical errors and reduce accidents in engineering settings (e.g., in the oil field industry[10]). Taking such an approach to error by building defensive tooling, process and management layers around the developer is likely to be effective in preventing developer errors from causing major system problems. More work is needed to adapt and evaluate the Swiss Cheese approach in software development contexts.

10. https://www.oilfieldtechnology.com/special-reports/23042015/rallying-against-risk/

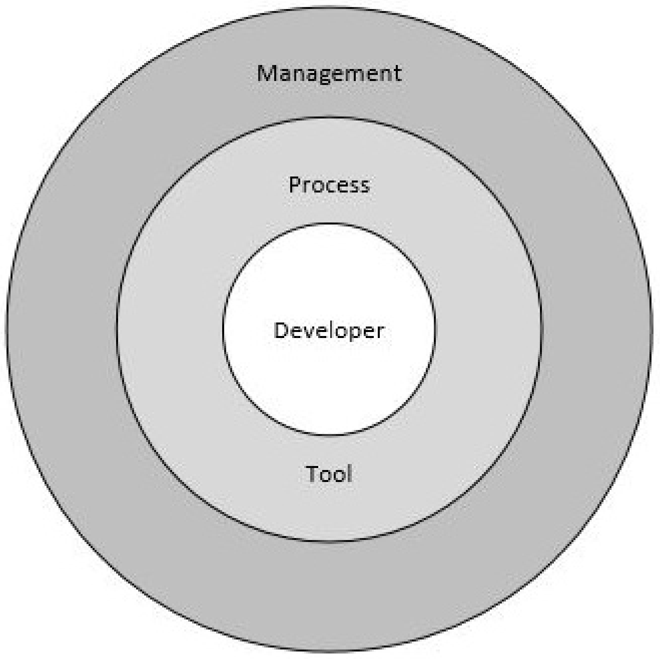

Figure 4. Mitigation Strategies

Few of the errors reported by developers are previously unknown. For example, it is not a surprise that complexity, requirements or concentration underpin developer errors. Similarly, many of the mitigation strategies identified by developers are also previously known approaches. For example, using testing, pull-requests and well formed processes to mitigate errors are all known to be helpful. It is more surprising that these errors continue to be problematic, suggesting that the existing mitigation strategies do not seem to be working effectively. More work is needed to understand why established approaches do not seem to, either be effectively embedded in software development practice, or, if they are embedded, not effectively reducing developer errors.

5.1 Improving Situation Awareness

Our results suggest that situation awareness is a problem for developers. Endsley [45] describes situation awareness as maintaining an understanding of what is going on around you while you perform a task so that you can predict what is likely to happen next. Many of the error causes given by developers suggest lost situation awareness. For example, Participant 12 says

> '...start trying to do something and you find some kind of annoyance gets in your way and it makes it impossible or more difficult than it ought to be so you kind of fix that and maybe end up going down a bit of a rabbit hole that is more complicated than you expected. I can recognise the feeling of just kind of having fought my way through that and not remember why I was doing that in the first place.'

Situation awareness is also a problem in other disciplines. Procidas [46] discusses how the Air France 447 crash resulted from the pilot's lack of situation awareness. The pilot was unable to see that the actions performed would lead to the aircraft stalling (aerodynamic loss of lift that occurs when an airfoil exceeds its critical angle). The pilot lost situational awareness and was not aware of everything going on around him. Other domains use maintaining situation awareness as an approach to reducing human errors, for example; autonomous driving [47], medicine [48], transportation [49] and cyber security [50].

In software development coding on 'autopilot' can lead to the loss of situation awareness which can lead to developers going down rabbit holes. The unplanned refactoring of code is usually such a rabbit hole. Developers should probably resist temptation to refactor while working on other development tasks and/or in an ad-hoc manner, as it is difficult to predict what else they will encounter while they make these additional refactoring changes. We recommend that developers conduct refactoring tasks as part of planned work rather than embedded in other tasks.

Participant 9 explains a mitigation strategy to prevent going down rabbit holes

> 'Discipline in one sense. Experience teaches you that not all code is perfect. You have to accept other people might write code in other ways so you have to have a pragmatic style of programming to look at code. Understanding is most important, thing for me is understanding the intention of the code rather than how it was written.'

Our results suggest that developers understand the dangers of losing situation awareness and also know some mitigation strategies for this. These mitigation strategies seem to be based on their own willpower. Using only willpower is likely to have limited success. Approaches to support developers in recognising when situation awareness is being lost need to be developed, perhaps in the form of training for developers as such training in situational awareness is common in health and medical practice. Brennan et al. [51] report that improving surgeons' ability to manage their awareness levels is an essential requirement to reducing medical error.

5.2 Improving Cognitive Skills

Our results suggest that cognitive skills such as remaining focused, remembering, maintaining self discipline and attention all affect developers. These cognitive skills also impact a developer's ability to maintain situational awareness. Developers indicated that these were skills they wanted to improve. Related to this, our results also show that many developers said they need to concentrate better, pay more attention, focus more. Participant 24 says

> '...you have to be 100% concentrated on the job otherwise you wouldn't succeed.'

Participant 16 says

> '...if I am not giving 100% attention then I take a break'.

The working environment and time pressure is likely to impact on how developers are affected by cognitive issues. For example, Participant 16 says

> '...it's actually detaching yourself from the urgency and focusing on what you're actually doing...'

Developers also seem to take personal responsibility for their ability to deploy effective cognitive skills. The underlining assumption being that developers just need more self-discipline and willpower. More work is needed to investigate the impact of training in existing approaches to improved cognitive skills. For example, training in using the Orient-Observe-Decide-Act Loop (OODA) [52] is often a part of military training and may prove worthwhile in helping developers improve their cognitive skills.

5.3 Using Checklists

Our results suggest that developers find checklists to be a useful mitigation strategy. Reducing the reliance to remember every step can aid in reducing human errors. Ely et al. [53] report that checklists provide an alternative to reliance on intuition and memory in clinical problem solving. For example, Participant 21 says

> 'Checklist. I basically tried to checklist for instance when you are implementing code and you have a use case instance, you check if you have done everything.'

This suggests that developers may use materials they already have as a checklist to aid the prevention of human errors. Checklists are the most frequently occurring process related mitigation strategy reported by developers we interviewed. This is a theme consistent with other industries where checklists have been used in many different situations. For example, in aviation pilots must complete a series of checklists during each flight stage [54]. Checklists have also been used in software development, for example, in risk management and extensively in software inspection [55]. Given the value of checklists that developers we interviewed reported in mitigating errors, more work is required to understand whether embedding the comprehensive use of checklists throughout the development process could reduce human error.

5.4 Tool use

Our results suggest that using software tools can provide developers with greater support. Tool related mitigation strategies were ranked second out of the four high level themes by the developers we interviewed. Table 7 shows the specific tools and types of tools which developers mentioned they used to mitigate errors. For example:

Participant 9 says

> '...the application helper is there to say you know. Start, finish in effect and when you finish you kind of end your transaction scope.'

This suggests that a developer uses a feature of their IDE to aid them in their code generation process. Such use can help with reducing omissions and repetitions as the application helper will identify that specific structures are missing or repeated.

Participant 18 says

> '...it [a specific tool] might be better if it was quiet maybe, that it says building. Doesn't give out all that information so at least if I come back to it, it will just have nothing there. And if it went wrong there would be something there.'

This quote suggests that the developer feels bombarded with information which is not clearly useful. By making small adjustments to tools it might be possible to make them more useful to developers.

Some developers also indicated that they commit code as they worry about machine failure and losing code. If this is not managed properly part completed code could be pushed to the master branch and lead to merge conflicts. For example, Participant 7 says 'So if I tried to push a WIP, Git would hook it and say that you are not allowed to do it...'. This simple solution suggests that thoughtful tool configurations might benefit developers. Following the lead of the aviation industry, standardising the interaction of tools with developers could help across multiple platforms and languages.

5.5 Faster feedback loops

Developers mentioned process related mitigation strategies (see Table 6) many of which seem to relate to getting fast feedback on code faults (e.g., reviews, testing and pull requests). Our results suggest that the quicker developers can get feedback on the consequences of their error, the quicker they can make improvements.

Pair programming is a well established approach to developers getting instant feedback on their code [56], [57]. Participant 20 says of pair programming:

> '...comparing it to for example, code reviews, it's just a very simple benefit. It's just faster feedback loops is all it is.'

Pair programming allows for developers to detect and remove faults in code caused by SB errors very quickly. The second developer's cognition is independent of the primary developer, therefore they are likely to be able to see the consequences of SB human errors in the code that have being written by the primary developer.

Pull requests seem to be an increasingly important way to get feedback on code. Participant 9 says

> 'so the majority of time we would pick it up in pull requests and you're basically reviewing someone else's code...'

The value of code review [58] is reiterated by Participant 1:

> '...sometimes you do not check for some errors and it just ends up sitting in the code base. It comes up in code review...'

Pull requests and code reviews both provide external feedback on code. Ideally, both should be used to identify developer errors in the form of code defects. Pull requests are not available in all tools used by developers, but where they are, we encourage the setting up of repositories so that pull request use is mandatory on top of code reviews. This intervention adds another layer of defence against errors translating into production code defects.

Overall our results suggest that developers value mechanisms where code is checked for defects, which have occurred as a consequence of their errors, and feedback quickly provided to them.

5.6 Tiredness

In some other industries, tiredness can be a serious cause of error for which mitigation is embedded in the processes used. For example, in the transportation industry drivers must not exceed driving a given number of hours in a day and must have a break at set intervals. This is tracked through the systematic use of a tacograph. Sugden et al. [59] report that tiredness will affect decision-making, complex mental tasks and awareness. Our results suggest that tiredness does not seem to be a high cause of errors in software development. This is surprising to us as we expected tiredness to be more problematic to software developers. Participant 11 says

> '...it is a sign that I'm tired...' and '...it's a matter of relaxing a little bit or resting for a few minutes or grabbing a coffee.'

This suggests that developers may recognise that they are fatigued but can deploy their own mitigation strategies to combat tiredness.

6 Related Work

Although there is limited work looking at HET in software development, some previous studies could be reconsidered in terms of HET. For example, LaToza et al.'s [60] early interview study found that the three most frequent problems reported by developers were 'understanding the rationale behind a piece of code', 'having to switch tasks often because of requests from my teammates or manager' and 'being aware of changes to code elsewhere that impact my code'. These problems could be categorised as slip, lapse or mistake, as for example, dealing with frequently switching tasks could lead to lapses and being unaware of significant changes in other parts of the code could lead to mistakes. Each of LaToza et al.'s [60] three problems have been subsequently studied with [61], [62] and [7] providing more understanding of the rationale behind a piece of code, [63] and [64] further investigating the impact of switching tasks and [65], [66] and [67] reporting on code change impact. Our results are consistent with both LaToza et al.'s [60] original and subsequent studies, and we confirm that, for example, context switching remains a source of human error for developers. Our results suggest that since LaToza et al.'s 2006 [60] study was published, despite attracting 600 google scholar citations, these causes of human error continue to be problematic for developers. Indeed it is likely that the proliferation of tools and languages together with the increasing sophistication of software means that the problems reported in 2006 may be worse now, though further work is needed to confirm this.

Huang et al., [22] and Hu et al., [15] are among the few previous researchers to have looked directly at HET in software development. Huang et al. developed taxonomies to help classify various types of error made by developers in terms of HET. On the basis of which, Huang and Liu [21] subsequently proposed a human centered defect prevention approach using their DPeHE framework. The DPeHE method focuses on the cognitive ability of the developer for error prevention. This framework is made up of three key stages namely knowledge training (stage 1), regulation training (stage 2) and continuous improvement (stage 3) [21]. Huang and Liu [21] concluded that approaches which go beyond conventional software process improvement are needed to prevent human errors. Our findings corroborate

Huang and Liu's [21] conclusions by highlighting the need for methods to improve the cognitive skills of developers.

A variety of previous studies have looked specifically at the impact and mitigation of specific causes of human error during coding. Many of these studies have focused on interruptions. For example, Züger et al. [68] reported on FlowLight, a tool developed to help developers reduce their interruptions. Züger et al. [68] reported that interruptions were reduced by 46% when using FlowLight. Such tools could be a useful mitigation strategy if transferred into professional practice. Our results suggested that the mitigation strategies developers report include many aimed at reducing interruptions, so it is possible that interruptions are now more understood, recognised and controlled by developers.

Previous work has looked at HET in relation to requirements errors. Anu et al. [12] used two taxonomies to understand errors in requirements, the first is Requirements Error Taxonomy (RETa) and the second is Human Error Taxonomy (HETa). The RETa categorises errors at a high level in terms of people (communication), process (elicitation) and documentation (specification) errors. The HETa categorises errors at a high level using Reason's slips (lack of consistency in the requirement specification), lapses (accidentally overlooking requirements), and mistakes (not having clear distinction between client and users) categories [12]. Our approach builds on and complements this previous work by introducing a finer grained classification of skill-based errors for coding tasks. Walia and Carver [18] developed a taxonomy by identifying and classifying software requirement faults. Hu et al. [17] reported prevention and mitigation methods within software requirements.

Huang [19] also reported on human errors in software requirements, looking specifically at post-completion errors.[11] Huang [19] concluded that there is a high likelihood of post-completion errors in software requirements. Although we did not look explicitly at requirements errors, our results show that developers continue to identify requirements problems as the reason for some errors.

7 Threats to Validity

As with any empirical research our study has several threats to validity. Below we explore these as construct validity, internal validity, external validity and repeatability.

7.1 Construct Validity

Construct validity assesses our ability to measure an 'object' we intend measuring. The responses given to us by a developer have been processed by the developer, therefore, we can not be certain the data is accurate. Interviewees may distort their responses to influence the impression of themselves they are presenting. Accuracy is a common threat to validity in all interview and questionnaire studies. We also need to consider participants' reactions to the researcher presence. We have tried to address this by allowing participants to volunteer their time to the study and schedule interviews on days/times that suit them best. In order to reduce the observer bias we recorded and transcribed the interviews. In addition we investigate only a subset of all human errors i.e. SB errors. Future work will investigate RB and KB errors.

11. As mentioned previously, post-completion errors are when a subtask is omitted at the end of a task but is not necessary for the completion of the task, an example, would be removing your bank card from an ATM machine prior to your cash being issued.

7.2 Internal Validity

Internal validity assesses whether all elements of the study have been designed and executed correctly. Interviewer bias is a common internal validity threat to interview studies. It is difficult to mitigate interviewer bias, however, we used the same interviewer throughout the study. This allowed us to ensure all interviews had been conducted in the same way and the approach to asking questions to probe and clarify were uniform throughout. Two other aspects of the interview design may have introduced unintentional bias:

1. The focus of the interviews was developer errors. This focus may have biased interviewee answers such that they over-reported developer-based mitigation strategies and under-reported tool, process and manager mitigation. Further research is needed to explore mitigation strategies which most effectively prevent developer errors becoming system failures.

2. Our interviews required developers to discuss the historic errors that they had made. The accurate recall of these errors relied on memory. Reliance on retrospective reporting has been shown [69] to not necessarily be reliable and to be systematically biased towards particularly memorable events.

These two potential biases are intrinsic to most interview studies and hopefully do not detract from the value of the exploratory results that we report.

7.3 External Validity

External validity assesses the ability to generalise our results. Our sample size of industry software developers is relatively small and it is not a random sample. So it is difficult to claim generalisabity to the wider population. In addition our sample of developers may have been biased, as participants may have already had an interest in the research or they were hoping to learn something about the topic. On the other hand, over half of our respondents have been practicing in industry for over 10 years and the qualitative data provided by these developers during the semi-structured interviews was very detailed. So the insights reported are hopefully authentic and potentially useful. In addition, lack of generalisability is almost always the case in empirical research in software engineering so our study is not an exception to the norm.

7.4 Repeatability

Repeatability assesses whether if this study were to be repeated by another they would get the same results. We provide a replication package (see our online appendix: https://bit.ly/2ZKgIsj) which contains a description of how participants have been recruited, interviews have been performed and analysis has been conducted. We encourage the replication of this study with a wider group of participants. Recruiting a similar group of participants could prove difficult as each researcher's social media, email, word of mouth outreach to potential participants is different. The demographics of the group recruited is important, as the type of SB human errors may vary based on a developer's experience.

8 Conclusion

In this paper we presented our preliminary study on the SB errors 27 industry software developers report they make and how developers mitigate these errors. We answer the following Research Questions:

RQ1: What SB human errors do industry software developers make while performing software development tasks? Our results suggest that developers blame their lack of focus and concentration errors for many faults. With many errors reported by developers caused by the complex development environment in which software development occurs.

RQ2: How do industry software developers mitigate these SB human errors? Our results suggest that developers predominately rely on trying to improve their own willpower to mitigate errors. Developers use a variety of strategies to retain their focus and concentration in order to reduce the number of errors they make. Developers also use mitigation strategies within the development process (e.g., checklists), tools (e.g., automation) and management (e.g., planning) to detect the code faults that are the consequence of their errors.

Our study raises interesting tensions around developers taking personal responsibility for errors as opposed to development processes effectively blocking the inevitable code faults that results from the human errors made by developers. We identify the need for a more systematic approach to developer errors being prevented from causing serious failures. Software development is likely to benefit from adopting a Swiss Cheese approach which has successfully been used in other disciplines to prevent errors causing major accidents and failures.

Acknowledgments

The authors would like to thank all the study participants. This study has has been approved by Brunel University Ethics Committee reference 12218-LR-Jul/2018-13605-1. The study was partly funded by the UK's Engineering and Physical Sciences Research Council (EP/S005730/1).

References

[1] F. Huang and B. Liu, "Systematically improving software reliability: considering human errors of software practitioners," in *23rd Psychology of Programming Interest Group Annual Conference (PPIG 2011)*, 2011.

[2] F. Huang, B. Liu, Y. Song, and S. Keyal, "The links between human error diversity and software diversity: Implications for fault diversity seeking," *Science of Computer Programming*, vol. 89, no. PART C, pp. 350–373, 2014.

[3] J. Reason, *Human Error*. New York; Cambridge [England]: Cambridge University Press, 1990.

[4] L. Pirzadeh, "Human Factors in Software Development: A Systematic Literature Review," Master's thesis, 2010.

[5] S. Ozcan Kini and A. Tosun, "Periodic developer metrics in software defect prediction," in *2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 2018, pp. 72–81.

[6] M. Ortu, B. Adams, G. Destefanis, P. Tourani, M. Marchesi, and R. Tonelli, "Are bullies more productive? empirical study of affectiveness vs. issue fixing time," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 2015, pp. 303–313.

[7] F. Ebert, F. Castor, N. Novielli, and A. Serebrenik, "Confusion in code reviews: Reasons, impacts, and coping strategies," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2019, pp. 49–60.

[8] J. Siegmund, "Program comprehension: Past, present, and future," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 5. IEEE, 2016, pp. 13–20.

[9] L. Goncales, K. Farias, B. da Silva, and J. Fessler, "Measuring the cognitive load of software developers: A systematic mapping study," in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, 2019, pp. 42–52.

[10] J. Rasmussen, "Skills, rules, and knowledge; signals, signs, and symbols, and other distinctions in human performance models," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 3, pp. 257–266, May 1983.

[11] V. Anu, G. Walia, W. Hu, J. Carver, and G. Bradshaw, "Effectiveness of human error taxonomy during requirements inspection: An empirical investigation," in *Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE*, vol. 2016-Janua, 2016, pp. 531–536.

[12] V. Anu, G. Walia, W. Hu, J. C. Carver, and G. Bradshaw, "Using a Cognitive Psychology Perspective on Errors to Improve Requirements Quality: An Empirical Investigation," in *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*. IEEE, 2016, pp. 65–76.

[13] V. Anu, G. Walia, and G. Bradshaw, "Incorporating Human Error Education into Software Engineering Courses via Error-based Inspections," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '17. New York, NY, USA: ACM, 2017, pp. 39–44.

[14] W. Hu, J. Carver, V. Anu, G. Walia, and G. Bradshaw, "Detection of Requirement Errors and Faults via a Human Error Taxonomy: A Feasibility Study," in *International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '16, vol. 08-09-Sept. New York, NY, USA: ACM, 2016, p. 30:10.

[15] W. Hu, J. C. Carver, V. Anu, G. Walia, and G. Bradshaw, "Defect Prevention in Requirements Using Human Error Information: An Empirical Study," pp. 61–76, 2017.

[16] W. Hu, J. C. Carver, G. Walia, and V. Anu, "Understanding Human Errors In Software Requirements : An Online Survey," *REFSQ Workshops*, 2017a.

[17] W. Hu, J. C. Carver, V. Anu, G. S. Walia, and G. L. Bradshaw, "Using human error information for error prevention," *Empirical Software Engineering*, vol. 23, no. 6, pp. 3768–3800, 2018.

[18] G. S. Walia and J. C. Carver, "A systematic literature review to identify and classify software requirement errors," *Information and Software Technology*, vol. 51, no. 7, pp. 1087–1109, 2009.

[19] F. Huang, "Post-completion Error in Software Development," *Proceedings of the 9th International Workshop on Cooperative and Human Aspects of Software Engineering*, pp. 108–113, 2016.

[20] S. Y. W. Li, A. Blandford, P. Cairns, and R. M. Young, "The effect of interruptions on postcompletion and other procedural errors: An account based on the activation-based goal memory model." *Journal of Experimental Psychology: Applied*, vol. 14, no. 4, pp. 314 – 328, 2008.

[21] F. Huang and B. Liu, "Software defect prevention based on human error theories," *Chinese Journal of Aeronautics*, 2017.

[22] F. Huang, B. Liu, and B. Huang, "A Taxonomy System to Identify Human Error Causes for Software Defects," *Proceedings of the 18th International Conference on Reliability and Quality in Design, ISSAT 2012*, pp. 44–49, 2012.

[23] R. Hoda, J. Noble, and S. Marshall, "Self-organizing roles on agile software development teams," *IEEE Transactions on Software Engineering*, vol. 39, no. 3, pp. 422–444, 2013.

[24] J. Smith, B. Johnson, E. R. Murphy-Hill, B. Chu, and H. Lipford, "How developers diagnose potential security vulnerabilities with a static analysis tool," *IEEE Transactions on Software Engineering*, vol. 45, pp. 877–897, 2019.

[25] L. L. Leape, "Error in medicine," *Jama*, vol. 272, no. 23, pp. 1851–1857, 1994.

[26] D. Graziotin, F. Fagerholm, X. Wang, and P. Abrahamsson, "On the

unhappiness of software developers," *CoRR*, vol. abs/1703.04993, 2017. [Online]. Available: http://arxiv.org/abs/1703.04993

[27] P. M. Fitts and R. E. Jones, *Analysis of factors contributing to 460" pilot-error" experiences in operating aircraft controls*. Aero Medical Laboratory Wright-Patterson Air Force Base, OH, 1947.

[28] S. Dekker, *Ten questions about human error: a new view of human factors and system safety*. Mahwah, N.J: Lawrence Erlbaum Associates, 2005.

[29] G. d. S. R. Ribeiro, R. C. d. Silva, M. A. d. A. A. A. Ferreira, and G. R. d. Silva, "Slips, lapses and mistakes inthe use of equipment by nurses in an intensive care unit," *Revista da Escola de Enfermagem da USP*, vol. 50, pp. 419 – 426, 06 2016.

[30] A. Beso, B. D. Franklin, and N. Barber, "The frequency and potential causes of dispensing errors in a hospital pharmacy," *Pharmacy World and Science*, vol. 27, no. 3, pp. 182–190, 2005.

[31] D. P. Brumby, A. L. Cox, J. Back, and S. J. J. Gould, "Recovering from an interruption: Investigating speedaccuracy trade-offs in task resumption behavior." *Journal of Experimental Psychology: Applied*, vol. 19, no. 2, pp. 95 – 107, 2013.

[32] J. L. Hakala, J. C. Hung, and E. A. Mosman, "Minimizing human error in radiopharmaceutical preparation and administration via a bar code–enhanced nuclear pharmacy management system," *Journal of nuclear medicine technology*, vol. 40, no. 3, pp. 183–186, 2012.

[33] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative research in psychology*, vol. 3, no. 2, pp. 77–101, 2006.

[34] A. N. Meyer, G. C. Murphy, T. Fritz, and T. Zimmermann, *Developers' Diverging Perceptions of Productivity*. Berkeley, CA: Apress, 2019, pp. 137–146.

[35] B. Kitchenham, D. I. K. Sjøberg, O. P. Brereton, D. Budgen, T. Dybå, M. Höst, D. Pfahl, and P. Runeson, "Can we evaluate the quality of software engineering experiments?" in *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '10. New York, NY, USA: Association for Computing Machinery, 2010. [Online]. Available: https://doi.org/10.1145/1852786.1852789

[36] N. Peitek, J. Siegmund, S. Apel, C. Kästner, C. Parnin, A. Bethmann, T. Leich, G. Saake, and A. Brechmann, "A look into programmers' heads," *IEEE Transactions on Software Engineering*, vol. 46, no. 4, pp. 442–462, 2020.

[37] P. Beynon-Davies, "Human error and information systems failure: the case of the london ambulance service computer-aided despatch system project," *Interacting with Computers*, vol. 11, no. 6, pp. 699–720, 1999.

[38] S. Sarkar and C. Parnin, "Characterizing and predicting mental fatigue during programming tasks," in *2017 IEEE/ACM 2nd International Workshop on Emotion Awareness in Software Engineering (SEmotion)*, 2017, pp. 32–37.

[39] E. R. Sykes, "Interruptions in the workplace: A case study to reduce their effects," *International Journal of Information Management*, vol. 31, no. 4, pp. 385–394, 2011.

[40] B. P. Bailey and J. A. Konstan, "On the need for attention-aware systems: Measuring effects of interruption on task performance, error rate, and affective state," *Computers in Human Behavior*, vol. 22, no. 4, pp. 685 – 708, 2006, attention aware systems. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S074756320500107X

[41] J. Abimanyi-Ochom, S. Bohingamu Mudiyanselage, and M. Catchpool, "Strategies to reduce diagnostic errors: a systematic review." *BMC Med Inform Decis Mak*, vol. 19, no. 1, p. 174, 2019.

[42] J. T. Reason, *The human contribution: unsafe acts, accidents and heroic recoveries*. Burlington, VT;Farnham, England;: Ashgate, 2008.

[43] J. Carthey, "Understanding safety in healthcare: The system evolution, erosion and enhancement model," *Journal of public health research*, vol. 2, p. e25, 12 2013.

[44] J. Reason, "Human error: models and management," *BMJ: British Medical Journal*, vol. 320, no. 7237, p. 768, 2000.

[45] M. R. Endsley, "Situation awareness global assessment technique (sagat)," in *Proceedings of the IEEE 1988 national aerospace and electronics conference*. IEEE, 1988, pp. 789–795.

[46] D. Procida, "Fighting the controls: Madness and tragedy in programming," 2017, djangoCon Europe 2017. [Online]. Available: https://www.youtube.com/watch?v=qI7NZV-rak0

[47] L. Petersen, L. Robert, J. Yang, and D. Tilbury, "Situational Awareness, Driver's Trust in Automated Driving Systems and Secondary Task Performance," *SAE International Journal of Connected and Autonomous Vehicles, Forthcoming*, 2019.

[48] M. C. Wright, J. M. Taekman, and M. R. Endsley, "Objective measures of situation awareness in a simulated medical environment," *BMJ Quality & Safety*, vol. 13, no. suppl 1, pp. i65–i71, 2004.

[49] C. D. Wickens, "Situation awareness and workload in aviation," *Current directions in psychological science*, vol. 11, no. 4, pp. 128–133, 2002.

[50] G. Ioannou, P. Louvieris, and N. Clewley, "A Markov Multi-phase Transferable Belief Model for Cyber Situational Awareness," *IEEE Access*, 2019.

[51] P. A. Brennan, D. A. Mitchell, S. Holmes, S. Plint, and D. Parry, "Good people who try their best can have problems: recognition of human factors and how to minimise error," *British Journal of Oral and Maxillofacial Surgery*, vol. 54, no. 1, pp. 3–7, 2016.

[52] J. Boyd, "A discourse on winning and losing [briefing slides]," *Maxwell Air Force Base, AL: Air University Library.(Document No. MU 43947)*, 1987.

[53] J. W. Ely, M. L. Graber, and P. Croskerry, "Checklists to reduce diagnostic errors," *Academic Medicine*, vol. 86, no. 3, pp. 307–313, 2011.

[54] R. Clay-Williams and L. Colligan, "Back to basics: checklists in aviation and healthcare," *BMJ Quality & Safety*, vol. 24, no. 7, pp. 428–431, 2015. [Online]. Available: https://qualitysafety.bmj.com/content/24/7/428

[55] B. Brykczynski, "A survey of software inspection checklists," *ACM SIGSOFT Software Engineering Notes*, vol. 24, no. 1, p. 82, 1999.

[56] L. Williams, E. M. Maximilien, and M. Vouk, "Test-driven development as a defect-reduction practice," in *14th International Symposium on Software Reliability Engineering, 2003. ISSRE 2003.* IEEE, 2003, pp. 34–45.

[57] J. E. Tomayko, "A comparison of pair programming to inspections for software defect reduction," *Computer Science Education*, vol. 12, no. 3, pp. 213–222, 2002.

[58] C. Sadowski, E. Söderberg, L. Church, M. Sipko, and A. Bacchelli, "Modern code review: A case study at google," in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, ser. ICSE-SEIP '18. New York, NY, USA: ACM, 2018, pp. 181–190. [Online]. Available: http://doi.acm.org/10.1145/3183519.3183525

[59] C. Sugden, T. Athanasiou, and A. Darzi, "What are the effects of sleep deprivation and fatigue in surgical practice?" in *Seminars in thoracic and cardiovascular surgery*, vol. 24, no. 3. Elsevier, 2012, pp. 166–175.

[60] T. D. LaToza, G. Venolia, and R. DeLine, "Maintaining Mental Models: A Study Of Developer Work Habits," in *Proceeding of the 28th international conference on Software engineering - ICSE '06*, ser. ICSE '06. New York, NY, USA: ACM, 2006, p. 492.

[61] Amit Seal Ami and M. S. Islam, "An efficient approach for providing rationale of method change for object oriented programming," in *2014 International Conference on Informatics, Electronics Vision (ICIEV)*, 2014, pp. 1–6.

[62] A. Sutherland and G. Venolia, "Can peer code reviews be exploited for later information needs?" in *2009 31st International Conference on Software Engineering - Companion Volume*, 2009, pp. 259–262.

[63] T. Fritz, D. C. Shepherd, K. Kevic, W. Snipes, and C. Bräunlich, "Developers' code context models for change tasks," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 7–18. [Online]. Available: https://doi.org/10.1145/2635868.2635905

[64] K. Kevic, B. Walters, T. Shaffer, B. Sharif, D. Shepherd, and T. Fritz, "Eye gaze and interaction contexts for change tasks – observations and potential," *Journal of Systems and Software*, vol. 128, pp. 252 – 266, 2017.

[65] H. Cai and R. Santelices, "A comprehensive study of the predictive accuracy of dynamic change-impact analysis," *Journal of Systems and Software*, vol. 103, pp. 248–265, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121215000424

[66] S. Jiang, C. McMillan, and R. Santelices, "Do programmers do change impact analysis in debugging?" *Empirical Software Engineering*, vol. 22, no. 2, pp. 631–669, 2017.

[67] H. Cai, R. Santelices, and S. Jiang, "Prioritizing Change-Impact Analysis via Semantic Program-Dependence Quantification," *IEEE Transactions on Reliability*, vol. 65, no. 3, pp. 1114–1132, 2016.

[68] M. Züger, C. Corley, A. N. Meyer, B. Li, T. Fritz, D. Shepherd, V. Augustine, P. Francis, N. Kraft, and W. Snipes, "Reducing interruptions at work: A large-scale field study of flowlight," in *Proceedings of the 2017 CHI Conference on Human Factors*

in Computing Systems, ser. CHI '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 61–72. [Online]. Available: https://doi.org/10.1145/3025453.3025662

[69] K. A. Ericsson and H. A. Simon, *Protocol analysis: Verbal reports as data.* the MIT Press, 1984.

PLACE PHOTO HERE

Bhaveet Nagaria is working toward the PhD degree with Brunel University London. His research interests include human factors and software engineering. Contact him at bhaveet.nagaria@brunel.ac.uk; http://www.brunel.ac.uk/bhaveet-nagaria

PLACE PHOTO HERE

Tracy Hall is a professor with Lancaster University. Her research interests include software engineering, ode analysis and defect prediction. Contact her at tracy.hall@lancaster.ac.uk; https://www.lancaster.ac.uk/scc/about-us/people/tracy-hall