# An Investigation of Human Error in Software Development

A thesis submitted as partial fulfillment of the requirement of Doctor of Philosophy (Ph.D.)

by

Bhaveet Nagaria

Computer Science Department

Brunel University London

January 2021

## Abstract

**Context:** Software defects occurring in code bases lead to an increased cost for software production and maintenance. To err is human nature and the process of software development is human centric. My analysis of the literature shows that the use of human error theory is emerging as an important tool for software development. **Aim:** The aim of my thesis is to present a training tool aimed at reducing the number of human errors developers make while working within the development phase of the Software Development Life cycle (SDLC) by improving developer situation awareness. **Methods:** My first study uses semi structured interviews to gain insight into what Skill-based (SB) errors developers make and how they mitigate these errors. My second study employs an experimental setup where developers log all human errors they make during developmental tasks across two weeks. At the beginning of week two the developers are asked to complete an online training package which I have developed on situation awareness. **Results:** The first study shows that the complexity of the development environment is one of the most frequently reported reasons for errors. I found that software developers struggle with effective mitigation strategies for their errors, reporting strategies largely based on improving their own willpower to concentrate better on development tasks. The results from the second study show that training software developers in situation awareness does lead to a decrease in the number of human errors made by those software developers. **Conclusion:** My doctoral research shows that human errors are a problem for software developers and loss of situation awareness is key for many of these developers. My preliminary results show that training tools which address situation awareness can aid developers in reducing the number of human errors that they make. Further work is required to investigate other means of improving developer situation awareness and determine whether my findings are generalisable.

## Acknowledgements

I would like to thank my principal supervisor, Professor Tracy Hall for her invaluable guidance, mentorship, support and encouragement throughout my studies both academically and personally. Our fruitful discussions always gave me the motivation to keep on going and gave me the confidence I was progressing in the right way. Special thanks to Professor Steve Counsell for his support throughout by doctoral journey. I would like to thank my research development advisors and members of my academic review panel Dr Nour Ali, Dr Andrea Capillupi, Dr Giuseppe Destefanis, Dr Allan Tucker and Professor Burak Turhan for their invaluable comments and suggestions.

I have been inspired, motivated, supported, guided and much more by many of the academics and support staff within the department and university. I would like to express my thanks to you all, in addition to my supervisors and panel members, key individuals are; Peter, Amy, Martyn, Sandi, Ela, Dr Theodora Koulouri, Dr Mahir Arzoky, Dr Stephen Swift, Profesor Simon Taylor and Professor Kate Hone.

Thanks to Gabriel, Mohammed, my DigiBru family (Toyah, Ben, Ashley, Isabel and Fawzia) and others, you have always been on hand to motivate me and more importantly been a great bunch of friends during this incredible journey.

## Declaration

The following papers have been accepted for publication as a direct result of the research discussed in this thesis:

- **Nagaria, B.** and Hall, T., 2020. How Software Developers Mitigate their Errors when Developing Code. IEEE Transactions on Software Engineering. (Accepted to appear) [Presented in Chapter 4 & 6]

- **Nagaria, B.** and Hall, T., 2020. Reducing Software Developer Human Errors by Improving Situation Awareness. IEEE Software, 37(6), pp.32-37. [Presented in Chapter 5 & 6]

The following paper has been accepted for publication as a part of collaborative research conducted during my doctoral research:

- Capiluppi, A., Ajienka, N., Ali, N., Arzoky, M., Counsell, S., Destefanis, G., Miron, A., **Nagaria, B.**, Neykova, R., Shepperd, M. and Swift, S., 2020. Using the Lexicon from Source Code to Determine Application Domain. In Proceedings of the Evaluation and Assessment in Software Engineering (pp. 110-119).

# Contents

# List of Figures

# List of Tables

# Acronyms

**CSS** Closed Source System.

**DPeHE** Defect Prevention Based on Human Error Theories.

**GEMS** Generic Error-Modelling System.

**KB** Knowledge-based.

**OSS** Open Source System.

**RB** Rule-based.

**SA** Situation Awareness.

**SB** Skill-based.

**SDLC** Software Development Life cycle.

**SE** Software Engineering.

**SLR** Systematic Literature Review.

# Chapter 1

# Introduction

The use of software is heavily embedded within our daily lives. The media commonly reports on defective software, recent examples include; Facebook[1] mass outage, British Airways[2] IT crash, Microchips[3] affected by Spectre & Meltdown bugs. These defects are typically introduced by industry software developers i.e. humans. Humans are susceptible to making errors. This chapter forms the basis of my exploration using Human Error Theory within software development.

Section 1.1 defines the problem. Section 1.2 details the problem solution. Section 1.3 defines the aims and objectives of my doctoral research. Section 1.4 provides insight into the expected contributions. Section 1.5 highlights the ethical issues that need to be addressed. Section 1.6 describes the structure of this thesis.

## 1.1 Problem Definition

Software defects are a problem as they lead to an increased cost for software production and maintenance. The development of software is very human

[1] https://www.bbc.co.uk/news/technology-47562281
[2] https://www.bbc.co.uk/news/uk-40069865
[3] https://www.bbc.co.uk/news/technology-42575033

centered, therefore I look at which human factors have been investigated within software development. Pirzadeh (2010) identifies that many human issues in software engineering have been heavily researched. Pirzadeh's Systematic Literature Review (SLR) reports investigations of the main aspects of estimation skills, management skills, pair programming, group performance, knowledge sharing, cost estimation and scheduling, staffing and management. Pirzadeh (2010) does not report of any studies which investigate software development using Human Error Theory as its focal point. Since Pirzadeh's SLR a decade ago, researchers have started to tackle the problem of human error within software requirements e.g. Anu et al. (2016*a*), Hu et al. (2016), Walia & Carver (2009) and software development e.g. Huang (2016), Huang & Liu (2017), Li et al. (2008).

To err is human nature and software development is a process which is centered around humans. Reason (1990) reports that errors are introduced during two phases of human cognition: execution and planning. 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. 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. Execution errors can be viewed as day to day things you know how to do, but for some reason you get wrong. Planning errors can be viewed as things you do not how to do and get wrong due to insufficient planning. Slips and lapses are further described by Rasmussen (1983) as Skill-based (SB) errors with mistakes described as Rule-based (RB) and Knowledge-based (KB) errors.

Huang et al. (2014) report that the number one cause of defects is the developer. Huang & Liu (2011) report that human error is a major contributor to software defects. Reason (1990) paved the way within human error

research and explains how Human Error Theory has been successfully used within the nuclear and transportation industries. By using Human Error Theory I investigate whether the number of human errors can be reduced during software development. Human error has been studied extensively and implemented successfully in other disciplines but there are very few studies of how Human Error Theory has been used in software development. One example such study by Huang (2016) reports that post completion errors[4] do occur within software development.

## 1.2 Problem Solution

I have started to address the problem of human errors within software development in two stages. The first uncovers what types of human errors are made and the second provides a training package which helps to safeguard against the most frequent form of human error identified.

Firstly I use Rasmussen (1983)'s error based framework of human errors to establish which Skill-based (SB) human errors are the most likely to occur within software development. Rasmussen (1983)'s error based framework describes a number of human errors, SB human errors best fit the day to day work that software developers undertake. I use SB errors to survey 27 industry software developers to gain an understanding of what type of errors they make and how they currently mitigate them. Reason (1990) describes the Swiss Cheese Model, as a model showing that layers of barriers are needed to block errors from slipping through to cause major failures. The errors identified in the survey study form a virtual Swiss Cheese Model. As a result I am able to propose four layers of 'cheese' i.e. developer, tool, process and management. The identified errors are represented as holes of varying

[4]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.

sizes Within these layers, where an error is more frequently occurring the larger the size of the hole with a layer.

I use my results and identify that Situation Awareness (SA) is a key problem to industry software developers. Endsley (1988) reports Situation Awareness (SA) is maintaining an understanding of what is going on around you while performing a task. Endsley (1988) describes three levels of situation awareness: (1) perception of the environment, (2) comprehension of the situation, (3) predicting the future situation. I develop, test and implement a four part training package centered around SA. The training package employs work on the OODA Loop by Boyd (1987) which is a cognitive training method designed to improve decision-making.

## 1.3 Research Aim and Objectives

The aim of this doctoral research is to:

> Deliver a training package for industry software developers aimed at reducing the number of human errors industry software developers make while working on development tasks.

In order to allow for successful completion of the aim, I have set out the following objectives:

- **Obj1** To understand what SB human errors occur during software development.
- **Obj2** To understand how industry software developers currently try to mitigate against their human errors.
- **Obj3** To establish whether industry software developers make more slips/lapses vs mistakes.

- **Obj4** To deliver and validate a training tool which aids industry software developers to mitigate against the most frequent forms of human error.

To gain a preliminary understanding of human error and its mitigation **Obj1** and **Obj2** are conducted together. I develop a survey which is delivered through semi structured interviews in which industry software developers participate. This survey will provide me with a preliminary understanding about what type of human errors manifest to industry software developers. The participants will provide an insight into how they currently mitigate against human errors. This knowledge will aid in appropriately designing the training tool mentioned in objective four.

To successfully complete **Obj4**, I design, implement and test a training tool which aims to improve software developer SA by training them on the OODA loop. Industry software developers will use the tool for a period of time in an experimental setup. During this tool use I will ask industry software developers to record all the human errors they make. Recording what human errors occur during the two week window will allow me to complete **Obj3** and further expand our understanding for **Obj1**.

I use an experimental setup to evaluate the impact of the tool by analysing the human errors made before using the training package vs the human errors made after using the training package. Huang & Liu (2017) successfully uses experiments to achieve a similar goal, where Huang & Liu attempts to reduce the number of defects by training industry software developers on a human error framework called Defect Prevention Based on Human Error Theories (DPeHE). My research differs from Huang & Liu (2017) as it focuses specifically on improving industry software developer SA instead of targeting all human errors. It employs a shorter period of training and is delivered via an online training package vs a trainer.

## 1.4 Expected Contributions

I expect four main contributions as a result of this doctoral research, these are;

- **Cont1** New understanding about typical human errors made by industry software developers is identified. While all forms of human error occur, slips and lapses (SB errors) seem to occur most commonly for industry software developers. Causes of human errors during development activities include the complex development environment, lack of developer concentration and going down rabbit holes.

- **Cont2** New understanding is presented about how industry software developers mitigate human errors during development activities e.g. focusing more, using headphones, increased automation and planning. Maintaining focus and concentration were reported frequently as a means of mitigating against human error.

- **Cont3** Designed, implemented and evaluated an online training package which allows industry software developers to improve their situation awareness. The package was trialled using industry software developers and enabled a reduction in the number of human errors being made.

- **Cont4** Gain an initial understanding of the layers (developer, process, tool and management) in the Swiss Cheese model as described by Reason (1990). Identification of SA being a large route through for errors in the developer slice of cheese. This led to the development of a training package which aids industry software developers in reducing the size of the hole in the slice.

## 1.5 Ethical Issues

This research project will require ethical approval as data will be collected and analysed from humans. In order to complete objectives one, two, three and four ethical approval will need to be obtained from the College using BREO.

1. To complete the first and second objective I will need to develop a survey delivered through a semi structured interview for industry software developers to participate in. [Ethics approval was granted on 20/07/2018 by University Research Ethics Committee under reference 12218-LR-Jul/2018-13605-1. The duration was extended for this study. This extension was granted ethics approval on 28/09/2018 by University Research Ethics Committee under reference 12218-A-Sept/2018-14228-1.]
2. To complete the third and fourth objective I will need to run an experiment for industry software developers to participant in. [Ethics approval was granted on 08/10/2019 by University Research Ethics Committee under reference 18067-LR-Oct/2019-20590-1.]

## 1.6 Thesis Structure

The thesis is delivered in seven chapters; there is a brief description of each chapter below.

**Chapter One: Introduction**

This chapter provides a high-level overview into the nature of the problem. It starts by discussing the problem and proposed solution. Furthermore, it details the aims and objectives, highlights the approach to the project, expected contribution to knowledge and ends with an outline of the projects ethical issues.

**Chapter Two: Literature Review**

This chapter provides an insight in the existing literature surrounding Human Error Theory, successful application of Human Error Theory in industries including; medicine, power and transportation. It explores research conducted within Human Error Theory in software requirements and Human Error Theory in software development. Finally explores how different aspects of Human Error Theory could be used to mitigate SB errors. This chapter presents the conceptulisation for **Cont4**.

**Chapter Three: Research Methodology**

This chapter considers various research methods that were considered for this empirical software engineering study. I explore why mixed method research is the chosen research methodology for my doctoral research. I explore what type of research methods have been used to address similar questions, provide details of data collection techniques, analysis techniques, empirical validity and how these change based on the type of research being conducted.

**Chapter Four: Which SB errors occur in the development phase of the SDLC and how can we mitigate these? - An interview approach.**

This chapter considers the type of Skill-based (SB) errrors which are made by industry software developers and methods used to mitigate these errors during the development phase of the Software Development Life cycle (SDLC). Using a semi structured interview method I interviewed 27 industry software developers. The interviews were transcribed and coded. The analysis showed that industry software developers' cognition tended to cause the most SB errors. This chapter details the approach and results for **Cont1**

**and 2**.

**Chapter Five: Can training industry software developers Situation Awareness (SA) reduce the number of human errors they make? - An experimental approach.**

This chapter describes an experimental study to gain insight into whether training software developer SA can lead to a reduction in the human errors being made. Industry software developers were asked to log human errors for 10 working days. On day 5 of the study they were asked to complete the training package which I developed to train in SA. I conduct this experiment with 10 industry software developers. The human error logs were coded. The analysis showed that there is a reduction in human errors post training. This chapter details the approach and results for **Cont1 and 3**.

**Chapter Six: Discussion**

This chapter discusses our findings for the studies detailed in Chapter 4 & 5. I discuss the various types of human errors that are made, what these look like, how they are mitigated and whether training software developer SA is an effective method to reducing software developer human errors. This chapter discusses the findings which contribute to **Cont1, 2, 3** and shows the application for **Cont4**.

**Chapter Seven: Conclusion**

This chapter summarises the key points within the literature, highlights the research methods being used and reports the importance behind the initial findings.

# Chapter 2

# Background

Human factors research within empirical software engineering investigates a plethora of topics, however, one that does not appear to have been extensively investigated is the use of Human Error Theory within software development (Pirzadeh 2010). Reason (1990) describes a number of small Human Error Theories which can be used to help us understand things that may go wrong for example; slips, lapses, mistakes which can manifest as accidents. Human Error comes from field of psychology. Requirements engineering has recently seem some application of Human Error Theory, an example of this can be seen in works by who .... This chapter provides an insight into the literature and details the conceptual model which is the foundation behind **Cont4**.

Section 2.1 discusses human factors within software development. Section 2.2 provides insight into the theoretical strands within Human Error Theory. Section 2.3 describes applications of Human Error Theory within other industries e.g. medicine and transportation. Section 2.5 explains the application of Human Error Theory within my doctoral research. Section 2.6 presents a summary of this chapter.

## 2.1 Human Factors

Pirzadeh (2010) conducted a SLR of human factors research within software development published in journal papers between 2000 and 2010. After the application of search criteria there were 67 papers remaining in the review. Pirzadeh (2010) answers four research questions, three of which are relevant to my work. I explore two of these below and the third in the next chapter.[1]

Pirzadeh's first RQ asks 'Which phases of software development have been addressed in SE human factors research? What are the most and least studied phases and what is missing?'. Pirzadeh (2010) categorised the papers against the following five software development phases; requirements engineering (67%), design (52%), implementation (46%), test (37%) and maintenance (31%). Pirzadeh (2010) report that 16% or 11 papers were common to all five development phases. The mid and later phases of the software development cycle seem most in need of further research addressing human factors i.e. implementation, test and maintenance.

Pirzadeh's second RQ asks 'What is the human role in these papers?'. Pirzadeh (2010) finds that there were three different groups of human in the studies these are; developer, manager and customer/user. Pirzadeh (2010) reports that 94% of papers looked at developers, 58% of papers looked at managers and 34% of papers looked at customers. When we consider that large amounts of software development is centered around developers i.e. the development of software itself these results are not surprising. Pirzadeh (2010)'s findings suggest that customers are least frequently study and therefore would benefit with more research. Given Human Error Theory research within SE is new, focus should be given to developers as this is the human group who is developing the software. Once Human Error Theory research is established within SE, researchers can branch out and focus on managers

[1] Due to overlapping category responses the total percentage exceeds 100 in Pirzadeh's research questions discussed below.

and customers.

Pirzadeh (2010) also investigates the focus of each human factor paper. The focus of each paper was classified into three groups: individual (76%), interpersonal (56%) and organisational (71%). Individual characteristics which impact the development process include, psychological issues, estimation skills and management skills. Group characteristics which influence the development process tend to reside within the interpersonal category, examples of which include, pair programming, group performance and knowledge sharing. Organisational characteristics which affect the development process tend to reside within the organisational category, examples include, cost estimation and scheduling, staffing and management issues. Pirzadeh (2010)'s findings suggest that interpersonal characteristics are least frequently studied and therefore could do with more research. Given Human Error Theory research within SE is new, focus should be given to individual characteristics before branching out to interpersonal and organisational characteristics.

Pirzadeh (2010) finds the requirements and design stages of the software development lifecycle to be more frequently studied in the literature compared to implementation, testing and maintenance. I have decided to target my literature review at the development stage of the lifecycle. In this stage we know that the human role is with the developer and therefore we know that the focus is at the individual level. However where developers work with agile methodologies the focus lies between both individual and interpersonal. Given that the individual focus persists regardless of whether a plan-driven or agile development process I have decided to pursue an individual focus. The employment of Human Error Theory in other industries is common, after reviewing the factors highlighted by Pirzadeh, I notice that human error is not cited. Following the successful use of Human Error Theory in multiple safety critical industries e.g. transportation, power and medicine, I have decided to delve into it further.

## 2.2 Theoretical Concepts in Human Error Theory

Human Error Theory is not widely adopted within software development. Software development is an entirely human driven process, understanding what and why human errors are made and how these can be safeguarded needs more focused attention within software development. Reason (1990) paved the way in human error research concentrating on incidents occurring within the nuclear and aviation industries. In addition to nuclear power and aviation, medicine regularly uses Human Error Theory, we are able to learn from their past work and implement safeguards within software development. These industries are safety critical, consequently employ a number of physical and digital safeguards e.g. checklists, physical barriers, two person controls and more. Firstly I provide a brief explanation of how Human Error Theory has been used within aviation. Secondly I explore the landscape of human error and provide an explanation on many theories.

A simple example of the use of how Human Error Theory has been used is within aviation can be found examining why some control knobs form part of certification requirements. Fitts & Jones (1947) conducted a series of individual and group interviews with pilots to collect accounts and analyse 460 "pilot-errors". Fitts & Jones (1947) found that by redesigning equipment in accordance with human requirements then it should be possible to eliminate a large number of "pilot-error" accidents. An example error that Fitts & Jones report is pilots confusing wing flap and landing gear controls. Dekker (2005) reports the immediate wartime fix was to attach a rubber wheel to the landing gear control and a small wedge to the flap control. Fitts & Jones (1947) make a suggestion that aircraft should provide uniform shape-coding of all control knobs which must be grasped quickly or without looking. Dekker (2005) reports that wartime design solution went on become a certification requirement.

### 2.2.1 Basic Cognition

Reason (1990) reports that human errors are introduced during two phases of human cognition; planning and execution. There are three error types which manifest at different stages of cognition. These are slips, which occur during execution, lapses, which occur during storage and mistakes, which occur during planning. A slip is a result of carelessness or inattentive actions for example day-to-day activities such as fat fingering. A lapse is a result of forgetfulness or a failure of memory, examples could include intending to do task A but not doing so due to an interruption and then resuming with another task. A mistake is the most severe type of error and a result of lack of knowledge during the planning stage of an activity. An example could include misdiagnosing a patient due to lack of experience and or not exploring their signs/symptoms properly. At a high level, slips and lapses are classified as execution errors whereas mistakes are referred to as planning errors.

In addition to these errors are violations, which are errors that look like slips, lapses and mistakes. Reason (1990) describes violations as illegal actions which are performed intentionally to cause harm, an example would be deliberately not following a safety protocol. Violations can be split into three groups; routine violations, optimising violations and necessary or situational violations. Routine violations are when people cut corners where the opportunity presents itself. Optimising violations are when actions are taken to further the goals of an individual. Necessary or situational violations are when it is deemed to be the only method of accomplishing a specific job (Reason 1995).

Reason (1990) reports that when referring to error mechanisms there are two structural features of human cognition which are working memory and the knowledge base. Reason (1990) also refers to brain bottlenecks in these structures, however they are known as attentional control mode (working

memory) and schematic control mode (the knowledge base). Reason (1990) reports that the attentional control mode is slow, effortful and difficult to uphold for more than brief periods. On the other hand the schematic control mode is able to process familiar information rapidly and without conscious effort. Reason (1990) reports that knowledge-based errors stem from limitations and incomplete or incorrect knowledge.

### 2.2.2 Skill - Rule - Knowledge Framework

Rasmussen developed an error-based framework of cognitive control mechanisms. Rasmussen's skill-rule-knowledge classification of human performance is error oriented and formed of three levels. These are Skill-based (SB) level, Rule-based (RB) level and Knowledge-based (KB) level. At the SB level the focus is on patterns of preprogrammed instructions. Reason (1990) defines these preprogrammed instructions as errors related to the intrinsic variability of force, space or time coordination. At the RB level the focus is on addressing recognisable patterns which are controlled by stored rules. These rules tend to take the form of if *state* then *diagnosis* or if *state* then *remedial action*. Therefore the errors in the RB level usually take the form of classification of a situation (state) therefore application of wrong rule (diagnosis) or incorrect recall of procedures (remedial action). At the KB level the focus is on unknown situations where by forth coming actions must be planned by employing analytical processes and stored knowledge. Consequently we can expect errors occurring from a combination of incomplete and incorrect knowledge and resource limitations (Reason 1990).

### 2.2.3 Generic Error-Modelling System (GEMS)

This subsection provides theoretical understanding and mapping in between the fundamental error types within Human Error Theory.

Reason (1990) describes GEMS which is based on the skill-rule-knowledge classification. GEMS is formed of three error types which map to the skill-rule-knowledge classification. The mapping between error type and performance level is as follows; slips and lapses occur at the SB level and are classed as execution failures. mistakes occur either at the RB or KB level and are classed as planning failures. Figure 2.1 helps to visualise how the two classifications map to each other.

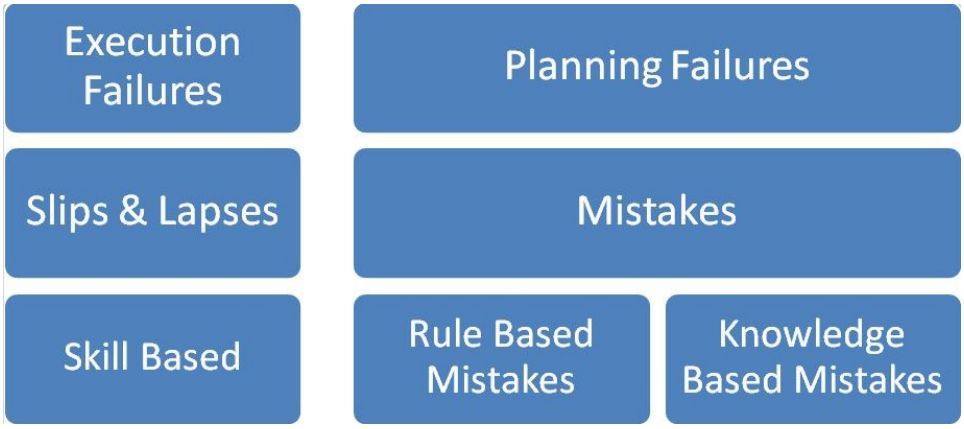



Figure 2.1: Reason's Reason (1990) Slips, Lapses, Mistakes mapped to Rasmussen's Rasmussen (1983) Skill-based (SB), Rule-based (RB) and Knowledge-based (KB) error types

Below I provide a comprehensive list of all the SB, RB and KB errors. By understanding the various error modes within Human Error Theory, I am able to classify the varying human errors software developers encounter during their work. Understanding each error is important as it will enable researchers to address issues at their route cause.

The major failure modes at the SB level as described by Reason (1990) are split in to two sub groups which are inattention or omitted check and overattention or mistimed checks. Within inattention there are five failure modes which are; double-capture slips, omissions following interruptions, reduced intentionally, perceptual confusions and interference errors. Within

overattention there are three failure modes which are; omission, repetitions and reversals (Reason 1990). SB errors take form as slips or lapses and are known as execution failures. For further information of these failure modes see Appendix A - Skill-based (SB) Errors.

The major failure modes at the RB level as described by Reason (1990) are split in to two sub groups which are misapplication of good rules and application of bad rules. Within misapplication of good rules there are seven failure modes which are; first exceptions, countersigns and nonsigns, informational overload, rule strength, general rules, redundancy and rigidity. Within the application of bad rules there are 4 failure modes which are; encoding deficiencies and action deficiencies. Within action deficiencies there is; wrong rules, inelegant rules and inadvisable rules (Reason 1990). RB errors take form as mistakes and are known as planning failures.

The major failure modes at the KB level as described by Reason (1990) are; selectivity, workspace limitations, out of sight out of mind, confirmation bias, overconfidence, biased reviewing, illusory correlation, halo effects, problem with causality and problems with complexity. Within problems with complexity there is; problems with delayed feedback, insufficient consideration of processes in time, difficulties with exponential developments, thinking in causal series not causal nets, thematic vagabonding and encysting (Reason 1990). KB errors take form as mistakes and are known as planning failures.

### 2.2.4 Swiss Cheese Model

In this subsection I explore what the Swiss Cheese Model is and how it has been applied within a SE environment.

The Swiss Cheese Model is a model which shows the route of an error through various layers of defence. Reason (2000) reports that each countermeasure or layer in the Swiss Cheese Model is likely to work effectively

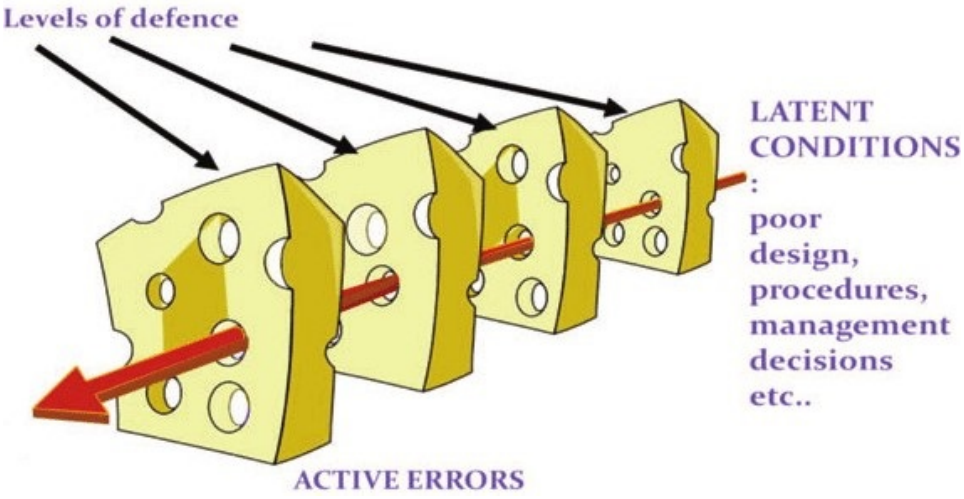



Figure 2.2: James Reasons's Swiss Cheese Model Reason (2008) adapted from Carthey (2013)

but will have some weakness. The Swiss Cheese Model (see Figure 2.2) of accident causation assumes that each defence layer is a slice of cheese. Each of the weaknesses represents a hole within the cheese. When all the slices of cheese are lined up usually there is no direct route through the layers of cheese. This represents a model where errors may occur but are likely to be caught out by another layer. In the event where the cheese layers are in a different angle such that a direct route becomes available this represents an ideal opportunity for an accident to occur (Reason 2000).

Reason (1990) reports that the holes in the cheese are nearly always caused by a combination of active failures and latent conditions. Active failures are acts of an unsafe nature which are committed by people with direct access to a system (Reason 1990). These acts tend to be direct and tend to have a short lived impact on the integrity of the defences. Defences are found a various layers of the Swiss Cheese Model and take form as physical or virtual safeguards e.g. access control or software systems. Slips, lapses, mistakes and violations are some of the forms active failures take. Latent

conditions are resident causes within a system. Latent conditions can have two types of effect, firstly, they have the potential to translate into error provoking conditions within the local workplace. Secondly they can create long lasting holes or weaknesses in defences. Untrustworthy alarms, design deficiencies, inexperience, inadequate equipment are some of the forms latent conditions take (Reason 2000). Latent conditions typically lie undetected within the system for long periods of time before a combination of local triggers and active failures create an accident opportunity. As such latent conditions can be discovered and mitigated against before an adverse event occurs, this creates proactive risk management (Reason 2000). I have explored the theoretical background of the Swiss Cheese Model above. My doctoral research elicits four layers of defence in the Swiss Cheese Model from interviews with industry software developers (see Chapter 4 for more details). These layers are; Developer, Process, Tool and Management. One latent condition that was identified is poor SA. I developed a online training package to mitigate against the latent condition of poor SA (see Chapter 5 for more details).

Within the context of SE Sommerville (2015) reports four strategies that can be employed when thinking about the Swiss Cheese Model within SE. These strategies are;

(i) 'Reduce the probability of the occurrence of an external event that might trigger system failures' - Analyse system triggers and implement mechanisms to reduce the likelihood of them occurring e.g. Prevent overloaded staff by giving staff more control over their workload (Sommerville 2015).

(ii) 'Increase the number of defensive layers' - Reduces the likelihood of holes in the cheese lining up (Sommerville 2015).

(iii) 'Design a system so that diverse types of barriers are included' - Decrease the likelihood of holes lining up by having holes in different

places (Sommerville 2015).

(iv) 'Minimise the number of latent conditions in a system' - Reduce the size and number of holes in each layer of cheese (Sommerville 2015).

Sommerville (2015) reports that all options must be considered and a decision is required to select the most cost effective method when improving a system's defence to human error. Sommerville (2015) reports that if custom software is being created then increased layers in the model may be the best option. Sommerville (2015) reports that sociotechnical defences and or changes to training procedures maybe required if off-the-shelf software is being used. My doctoral research focuses on employing suggestions 3 & 4. Within SE, safeguards are typically embedded within process e.g. daily stand up meetings and tools e.g. compilers. I introduce an short online training package which forces software developers to look at their own decision making ability. By doing so this reduces the size of the SA hole.

The problem of reducing software faults caused by human error has already been addressed using the Swiss Cheese Model unknowingly. The Swiss Cheese Model highlights the path an error takes through various layers. The error could be prevented through use of different types of safeguards, barriers and other mechanisms, within SE these already exist. One such examples is automated deployment pipelines which can significantly reduce deployment errors, another is the compiler which can catch syntax errors. By increasing the number of safeguards at each layer in the Swiss Cheese Model we as a SE community can further reduce the number of errors made. Work is required to understand exactly what errors occur at the human level and how these are currently safeguarded.

Looking at the Swiss Cheese Model, the ideal situation is one in which the holes in the cheese never line up therefore never cause a system failure. I target one type of latent condition in my solution - SA. Through the introduction of a training package, I am introducing a defensive mechanism

which aims to significantly reduce the size of the SA hole in the developer cheese layer relating to the targeted human error.

### 2.2.5 Summary

In this section I have explored the theoretical background of many of the human error theories. I have explained basic cognition, skill-rule-knowledge based framework, how the two map to each other within the GEMS and explaining the Swiss Cheese Model along with how it is applied within SE and my research.

## 2.3 Applied Human Error Theory

In this section I explore how Human Error Theory has been used within the field of medicine, its relevance to software engineering and its emerging application within software requirements and software development.

### 2.3.1 Human Error Theory in Medicine

In this section I explore the work of Leape (1994) who explores human error within the field of medicine. He explores how SB, RB and KB errors are affected by physiological, psychological and environmental factors. I report on work conducted by Ribeiro et al. (2016) who looks at what types of errors nurses make while operating specific machinery in an Intensive Care Unit. The literature I present provides examples of how Human Error Theory has been successfully applied within medicine.

Leape (1994) report that SB, RB and KB errors are affected by physiological, psychological and environmental factors. The physiological factors include fatigue, sleep loss, alcohol, drugs and illness. The psychological factors include other activity and emotional stakes e.g. boredom, fear, frustration, anger or anxiety. The environmental factors include noise, heat,

visual stimuli and motion. Given the abundance of mechanisms and error causes there is no single means of reducing human error. There are varying methods of error reduction at each phase of system design. Short-term memory, planning and problem solving are amongst the weakest aspects of cognition, therefore tasks should be simplified to minimise the load on these aspects (Leape 1994). Given the infancy of Human Error Theory research within SE, researchers should focus on all issues. Future work could uncover the impacts of each of these factors in more depth and also consider implementing safeguards around these factors. An example would be the use of tacographs in commercial driving to ensure that drivers are getting sufficient amounts of rest daily.

Ribeiro et al. (2016) ask what is the nature of the use of equipment by nurses working in the Intensive Care Unit, and what is the relationship between such use and the occurrence of errors? Ribeiro et al. (2016) identify human errors relating to use of equipment by Intensive Care Unit nurses. They use observations and interviews of eight Intensive Care Unit unit nurses between March to December 2014 totally 130 hours of observations. The observations were only conducted when nurses were using specific equipment. This subset of equipment has been selected as the literature supports a higher incidence of error occurring when using these items. The interview process was aimed at establishing the type of errors, related factors, behaviour and damage to patients.

Slips and lapse errors that occur during the programming of infusion pumps relate directly to the SB level. The tasks the nurses undertake while using the machines are typically routine tasks which are performed automatically and do not require much thought (Ribeiro et al. 2016).

Ribeiro et al. (2016) identify a variety of slips, lapses and mistakes which are made when nurses use equipment in Intensive Care Unit. Ribeiro et al. (2016) find that it is mainly infusion pumps and monitoring systems which

involve unfavourable events that harm patient safety. Ribeiro et al. (2016) report the route mechanism behind these errors are memory and attention lapses in the handling of infusion pump ability; planning failures during programming of the monitors; application of rules and knowledge. To mitigate against these errors, Ribeiro et al. suggest that daily checks of infusion pumps and monitors are performed.

My doctoral research follows a similar pattern to that identified in the example above, a list of latent conditions are identified by observing nurses. Following on from the observation Ribeiro et al. (2016) implement a safeguard of daily checks. This will assist with reducing the hole size of the latent condition within the Swiss Cheese Model. My research uses semi structured interviews to identify human errors. I develop an online training package as the safeguard which aims to reduce the hole size of the latent condition.

### 2.3.2 Human Error Theory in SE

Over the years human factors have been researched within SE. This research has not been tied back to Human Error Theory, below I highlight two examples of where existing research could be tied into Human Error Theory. I show how an experimental study reduces the rate of human error and the application of the nested application of the Swiss Cheese Model within another study.

LaToza et al. (2006) interviewed a group of developers about activities, tools and problems. LaToza et al. (2006) found that the top three 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 can be categorised within slip, lapse and mistake groupings. Having many tasks changes could lead to lapses and being un-

aware of significant changes could lead to mistakes. The problems described by LaToza et al. (2006) can be linked to Situation Awareness (SA), which is knowing what is going on around you and within your environment (SA is explained in detail in Section 2.4). Developers who are able to better manage their SA, will likely not struggle as much with these problems.

Developers invest greatly in recovering knowledge by code exploration and disrupting teammates; the acquired knowledge is only saved in their memory (LaToza et al. 2006). There are two control modes within the brain; attentional control mode (working memory) and schematic control mode (the knowledge base). Reason (1990) reports that the attentional control mode is slow, effortful and difficult to uphold for more than brief periods. On the other hand the schematic control mode is able to process familiar information rapidly and without conscious effort. Due to the weakness of the attentional mode we can see how developers could introduce more defects due to increased interruptions.

Typically mistakes otherwise known as RB or KB errors occur when some element of planning is involved. Bird et al. (2011) reports that software testers should give components with low ownership[2] priority. It is easy to see here how low code ownership links with knowledge based errors as there is no given expert or owner of the code therefore increasing the chances of more defects in that part of the code. If software testers develop a better understanding and awareness of who made the code they would be better placed to focus on key areas of code that could be more likely to contain faults.

Some strands of human factors research can be directly linked to aspects of Human Error Theory which can be applied to SE. When Human Error Theory is studied more closely within SE linkages may be made between the two. There are some clear areas of human factors research that should

[2]A developer who has made less than 5% of the changes to a given file Bird et al. (2011)

investigate the impacts it has in terms of Human Error Theory e.g. productivity, do productivity tools reduce the number of distractions in turn reducing the number of human errors made?

Huang et al. (2012) and Hu et al. (2017) have developed taxonomies to help classify various types of error made by developers in terms of Human Error Theory. Anu et al. (2016*b*) evaluate two taxonomies, the first is Requirements Error Taxonomy and the second is Human Error Taxonomy. The Requirements Error Taxonomy categorises errors at a high level which are people (communication), process (elicitation) and documentation (specification) errors. The Human Error Taxonomy 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 (Anu et al. 2016*b*). These studies describe success in being able to extract human errors from requirements engineers. Researchers have been able to create taxonomies with the described errors. Researchers could view the taxonomy levels can be viewed as layers of cheese within the Swiss Cheese Model.

Huang & Liu (2017) propose a defect prevention approach which is human centered using Defect Prevention Based on Human Error Theories (DPeHE) framework. This framework is made up of three key stages namely knowledge training (stage 1), regulation training (stage 2) and continuous improvement (stage 3) (Huang & Liu 2017).

During the first stage developers are educated on human errors specifically the why, when, what and how questions. They are then trained on why and when developers make errors and patterns of errors that maybe committed in specific scenarios. Finally they are at a stage where they can self-regulate in error prone situations, as such they are consciously aware of these situations and are able to employ prevention strategies (Huang & Liu 2017).

In the second stage developers are regulating themselves based on the knowledge they obtained in stage 1. They have to complete checklists pre and during programming tasks in order to monitor problem solving processes assisted by the problem solving regulation list. Developers are required to continue iterating through this process until they become able to consciously reflect and be aware of symptoms under error prone situations (Huang & Liu 2017). In the final stage developers acquire constant cognitive ability of human error prevention and perform continuous improvement with the build up of experience. Huang and Bin, test the DPeHE framework in two companies, the first company is at Capability Maturity Model Level 5 with 8 participating developers. The second company is at Capability Maturity Model Level 1 with 6 participating developers (Huang & Liu 2017). Huang & Liu (2017) report that the case studies suggest that DPeHE is successful in improving software developers' ability to prevent software defects. The Relative Progress of Defect Rate is increased in developers whom participated in the study. The mean Relative Progress of Defect Rate for company A participants it is 56.1% while for non participants it is 28.4%. For company B participants it is 31.3% while for non participants it is 0.7%. Huang and Bin outline that the delivery of DPeHE is not easy as it requires interdisciplinary trainers whom possess knowledge of both software engineering and psychology (Huang & Liu 2017).

Huang & Liu (2017) show that there are other avenues with defect prevention opposed to the conventional routes within organisational software process improvement. The suggested method of using DPeHE focuses effort on the cognitive ability of the developer in relation to human error prevention. This shows that similar approaches can be take with other human factors to further boost defect prevention.

Huang (2016) report that post completion error is a type of human error. This type of error is where a sub task is omitted at the end of 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. Huang (2016) reports that there are three strategies that can be employed to aid in defending against post completion error. These are as follows:

(i) Eliminate the post completion task where it is not necessary.

(ii) Change the procedure of the task if possible.

(iii) Highlight the places of post completion tasks in the requirement documents.

Huang (2016) concludes that there is a high likelihood that developers will introduce post completion errors if the post completion scenario is present in software requirements. In this example we learn how post completion errors move from requirements to development, if they are not caught early. We can see that the error has been identified i.e. post completion errors, removal of tasks that could lead to these errors or highlighting post completion tasks to safeguard against these errors occurring. This employs the Swiss Cheese Model in its basic form, identification of an error and implementing a safeguard to prevent the error from passing through. It would be useful if future work looked to automate these safeguards if and where possible as it is still reliant on a human to identify the post completion tasks.

## 2.4 Situation Awareness (SA)

My research uncovers that maintaining Situation Awareness (SA) is an area of difficulty for software developers. Below we explore what SA is, how it has been applied and explore a means of improving ones SA by using the OODA Loop.

Endsley defines Situation Awareness (SA) as the perception of the elements in the environment within a volume of time and space, the comprehension of their meaning and the projection of their status in the near future

(Endsley 1995). Endsley (1995) reports that there are three levels of SA. These are; Level One - Perception, Level Two - Comprehension and Level Three - Projection. Figure 2.3 shows how SA fits into the dynamic decision making process.

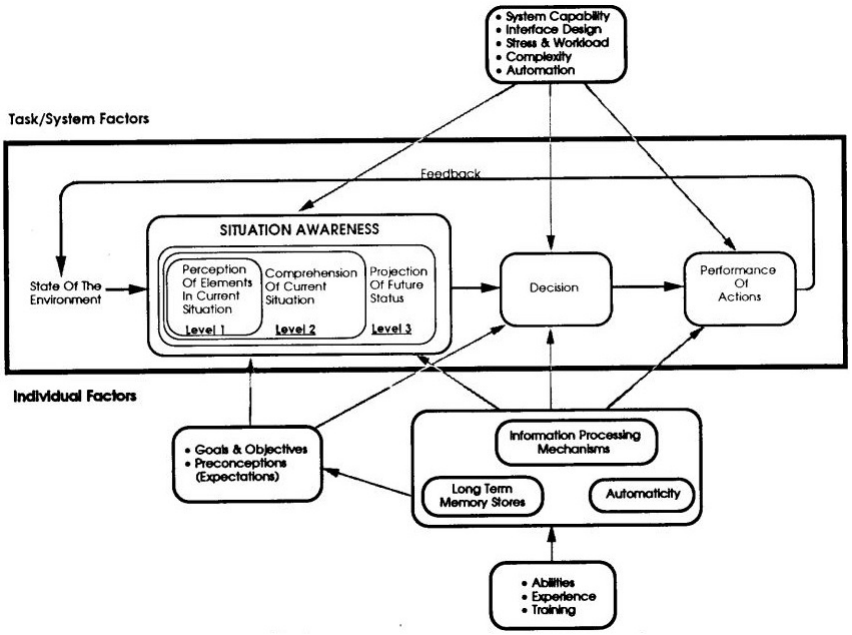



Figure 2.3: Model of SA in Dynamic Decision Making (Endsley 1995)

Level One SA is about the perception of the elements in the environment. Endsley (1995) reports that the first step in achieving SA is to perceive the status, attributes and dynamics of relevant elements in the environment.

Level Two SA is about comprehension of the current situation. Endsley (1995) reports that comprehension of the situation is based on a synthesis of disjointed level one elements. Level two SA goes beyond simply being aware of the elements that are present to include an understanding of the significance of those elements in light of pertinent operator goals.

Level Three SA is about projection of future status. Endsley (1995) reports that the ability to project the future actions of the elements in the environment at least in the very near term forms the third and highest level

of SA. This is achieved through knowledge of the status and dynamics of the elements and comprehension of the situation.

SA is used in any decision making process. First you are aware of a situation, second you understand elements within the situation and you may be able to project future actions. An example of day to day SA is your personal safety. You may enter a room and detect a threat. You consider the threat and possibly exit the room / neutralise the threat / raise the alarm. SA has been used with success in a variety of domains e.g. autonomous driving (Petersen et al. 2019), medicine (Wright et al. 2004), transportation (Wickens 2002) and cyber security (Ioannou et al. 2019). It has also been used in the military for combat situations, an example of this is Colonel John Boyd's OODA Loop (we take a closer look at this in the next section).

Endsley & Garland (2000) report two methods by which to identify methods for improving SA. The first is though an examination of the ways in which SA errors occur. The second is identify the successful methods that pilots use to develop and maintain SA compared to pilots that do not perform well at maintaining SA while performing a task.

Endsley & Garland (2000) report seven SA problems within General Aviation, these are; task management, basic procedures, vigilance, dealing with malfunctions, building mental models and critical skills. Endsley & Garland (2000) identifies four training recommendations; task management, development of comprehension (level 2 SA), projection (level 3 SA) and planning and information seeking and self checking activities.

### 2.4.1 OODA Loop

The OODA (Observe-Orient-Decide-Act) loop is a key element to the online training package I develop to aid software developers to reduce the number of human errors they make (see Chapter 5). We typically go through the OODA loop process hundreds if not thousands of times every day (High-

tower 2007). The OODA loop is a cognitive training method designed to improve decision-making (Boyd 1987). The four stages of the OODA loop (Figure 2.4) are Observe - Orient - Decide - Act, which form the basis of improving critical decision making. The OODA loop will only assist the user in getting to level two SA i.e. Comprehension. The OODA loop encourages the maintenance of situation awareness (SA) by iteratively 'Observing' (level 1 of SA), 'Orienting' (level 2 of SA), and 'Deciding' before 'Acting'.

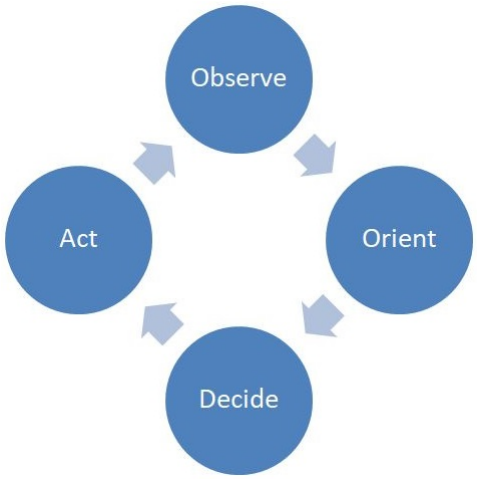



Figure 2.4: OODA Loop (Boyd 1987)

Richards (2020) report that it has sometimes proven advantageous to take extra time selecting a course of action—that is, reaching a decision to act—in order to create a more favorable environment for actions in the future. Given software development is not a combat situation, developers have the luxury of time. When developers use the OODA Loop, it is imperative that they bring the decision making in to the forefront of their minds and actively think during the decision making process before pursuing an action.

An example of day to day use of the OODA loop is the process of purchasing a meal. First you notice you are hungry, this the observation phase. You proceed to the phase of orientation which could be remembering its still breakfast hour in your local bakery and you can get a sausage roll. Next

comes the decision making in which you actively decide to take a break from your current task and get a sausage roll or do nothing about your hunger. Finally comes the action phase in which you actual going to the bakery and purchase a sausage roll.

## 2.5 Application of Human Error Theory Within My Doctoral Research

My doctoral research will use a combination of the two human error frameworks i.e. GEMS and the skill-rule-knowledge framework. My research employs use of the Swiss Cheese Model. Figure 2.5 depicts the process at a high level and explains how I tie the human error classification in to use with the Swiss Cheese Model. This forms the basis of **Cont4**.

I will use part of the skill-rule-knowledge framework to gain an initial understanding of what human errors software developers make and how they mitigate these. In Chapter 4 I identify four high level themes of human errors which were elicited from the participants. These themes can be viewed as layers, which are developer, process, tool and management. I aim to use these layers to create a virtual Swiss Cheese Model with 4 layers of 'cheese' based on insights obtained from these developers. This investigation takes place during Study one as detailed in Chapter 4.

Chapter 4 unpicks all the themes identified during elicitation of human errors from software developers. I learn that the developer layer of 'cheese' has the largest number of frequently occurring holes within it. Looking at the lower level themes within the developer layer, I have identified that SA is the area in significant need of addressing. The identification of SA as the area to address means I am able to develop a training package which acts as a safeguard measure for the SA hole in the developer slice of the cheese. The training package will aim to improve software developer situ-

ation awareness, in turn reducing the size of the SA hole in the developer slice. This investigation takes place during Study two as detailed in Chapter 5.

I will use the GEMS framework for further the understanding I have on what human errors software developers make. By using the GEMS framework I will be able to gain an insight into whether software developers do indeed make SB errors i.e. slips and lapses more often than RB errors and KB errors i.e. mistakes. This investigation takes place during Study one as detailed in Chapter 4.

## 2.6 Summary

Having conducted an initial exploration of human error within the development phase of the SDLC. The literature suggests that there are different error frameworks/models that can be used including the Swiss Cheese Model, GEMS and skill-rule-knowledge framework. My review of existing works shows that there is very little work conducted within the area of Human Error Theory and software development. My review shows numerous successes where sectors such as medicine, transportation and nuclear power have used Human Error Theory. Consequently there was a need for an exploratory study to be conducted so that I could establish exactly which human errors are most frequently occurring within the development phase of the SDLC. The study looked to explore which human errors occur when software developers are developing code on industry projects. In the next chapter we explore how this exploratory study will take place and the appropriate research methodology to be used.

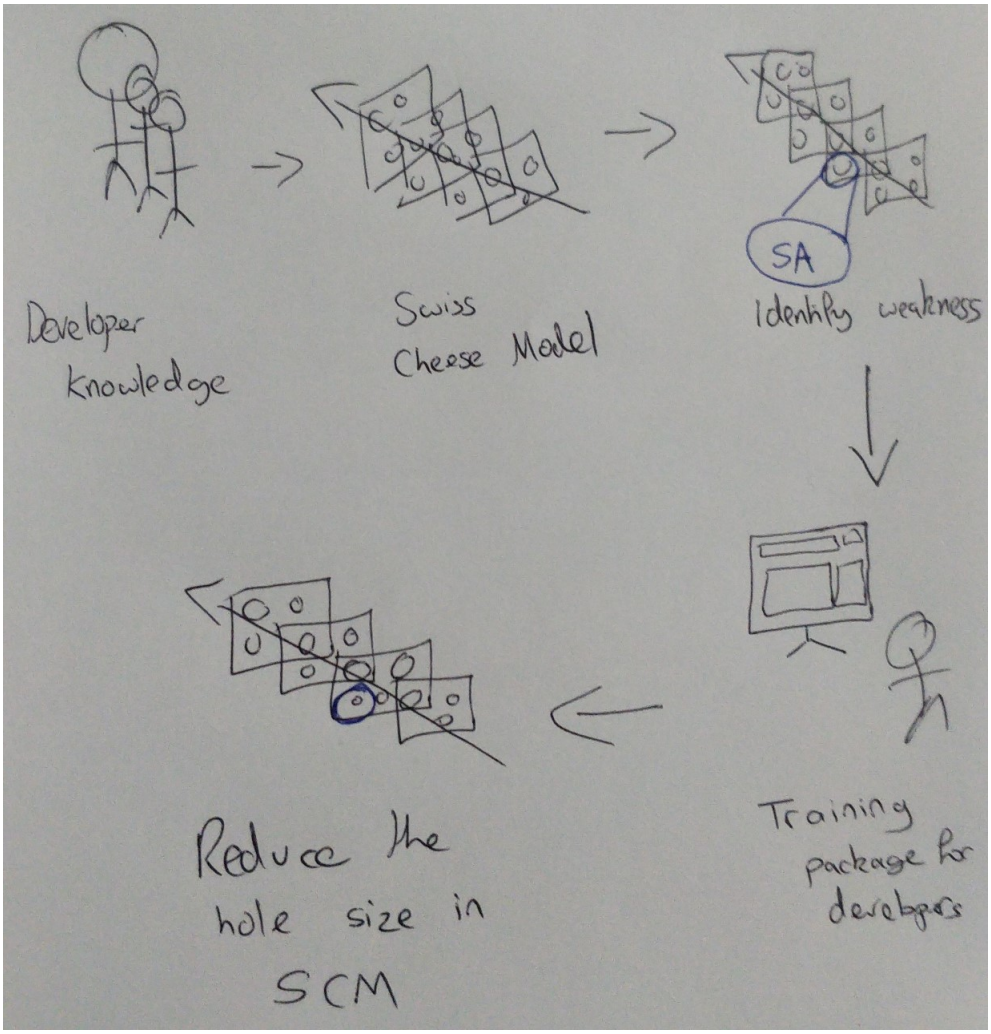



Figure 2.5: Conceptual Model

# Chapter 3

# Research Methodology

Software engineering is a growing and diverse field which pushes many boundaries including social and technical. It is imperative that studies look at surrounding aspects as well as the core to properly understand the implications of the work. This requires researchers to study the users i.e. humans as well as the tools and process. Easterbrook et al. (2008) highlight that researchers select methods with little to no understanding of the purpose of a given method or the available alternatives. Easterbrook et al. (2008) also report that methods from other fields are drawn upon in software engineering when it comes to the study of human behaviour. These fields include psychology when looking at an individual level and sociology when looking at a group level. This chapter focuses mainly on the theory of the methods, I detail the specific methods used for each study in Chapter 4 and 5.

Section 3.1 presents a background to research methodologies used in human factors and Human Error Theory studies. Section 3.2 presents the research questions I ask for my doctoral research. Section 3.3 presents the theory behind survey methods. Section 3.4 presents the theory behind experimental methods. Section 3.5 presents the analysis techniques used. Section 3.6 discusses the need for pilot studies. Section 3.7 argues why industry participants should be used over student participants. Section 3.8 discusses

sampling. Section 3.9 discusses empirical validity. Section 3.10 presents a summary of this chapter.

## 3.1 Background

In this section I explore which research methods have been used in; human factors research within software development and Human Error Theory research conducted in the fields of software requirements and software development. By looking at the research methods used in these areas of research I will be able to better select correct research methodology for my doctoral research.

### 3.1.1 Methods reported by Pirzadeh

Pirzadeh (2010) conducted a SLR of human factors research within software development published in journal papers between 2000 and 2010. I explore the last research question, 'What kind of research methods/tools (empirical, case study, survey, etc) have been used in this research area? What types of papers are published in the area of research and what was their approach to the topic?' asked by (Pirzadeh 2010) below.[1]

Pirzadeh (2010) provides useful context to the methods typically used in research on human factors in software engineering. Pirzadeh (2010) finds that the majority of papers (82%) use empirical research methods. My doctoral research will add to the body of existing empirical research. Four research methods were found to be used across the reviewed papers; case study (78%), design (65%), survey (47%) and controlled experiment (25%). Pirzadeh (2010) reports that industrial reports have not been used in any study. My doctoral research will employ survey and controlled experiment methods.

[1]Due to overlapping category responses the total percentage exceeds 100 in Pirzadeh's research questions discussed below.

When selecting participants for a study more effort should be directed at recruiting industrial partners. Pirzadeh (2010) reports that 74% of papers which use the case study methodology have used industrial case studies. Experiences of students who are learning can be different from experienced professionals, therefore more studies should focus what impacts the practitioners vs students. Survey methods is split between interviews (62%) which use semi structured and structured interviews and questionnaires (35%) which utilise a variety of open ended, self administered, web based and scaling questionnaires. Pirzadeh (2010) reports eight solutions within the design method which are model (25%), guideline (22%), framework (17%), hypothesis (11%), lessons learned (8%), approach/proposal (8%), method (6%) and metaphor (3%). Controlled experiments showed overlap with survey methods (33%) and case study (56%).

Papers that explored human factors using case studies commonly explored factors including collaboration, team work, psychological, cognitive issues, management and communication. Papers that used controlled experiments seemed to target cost estimation and the related cognitive, management and psychological factors. The popular topics of interest were not reported for survey and design methods. Theoretical research review concepts including individual professionalism, systematic studies on pair programming human factors/motivations of software engineers, technical communications, software developers and their personality types and work centered organisations and leadership (Pirzadeh 2010).

I have gained an insight into what research methods have been used in human factors research applied to software engineering. This has allowed me to understand which methods have been used in human factors research and notice that some methods have not been used e.g. ethnographic. This surprises me as observations of people appear an effective way to understand developer experiences in industry. Ethnographic research can be time

and cost heavy to both the researcher and participants/industrial collaborators. This could be why researchers tend to favour survey and controlled experiments when it comes to SE human factors research. In the next two subsections I explore which research methods have been used in Human Error Theory research in software requirements and software development research. To my knowledge these are the only areas with SE that have been researched within terms of Human Error Theory, therefore it is useful to explore what methods have been used.

### 3.1.2 Methods reported in Human Error Theory and software requirements

Anu et al. (2016*a*, 2017), Hu et al. (2017) collect quantitative data by using an experiment and a five point likert scale questionnaire. Anu et al. (2016*b*) collect quantitative and qualitative data by using an experiment, using randomized pre-test post-test control group experiment was planned and executed in controlled settings. Hu et al. (2017a) collect qualitative data using a questionnaire to determine what type of human errors and faults requirements engineers make on real projects and what methods are used/intended to be used to prevent requirement errors. Hu et al. (2017) collect quantitative data using an experiment to evaluate whether an understanding of Human Error helps prevent faults during requirements creation. Participants completed a training session and five experimental steps as part of this evaluation.

Research conducted about Human Error Theory and software requirements has focused on understanding what type of human errors have been made and introduced. The first objective of my doctoral research is to establish which type of human error occurs most within the development phase of the SDLC. By understanding how a similar objective has been achieved within a similar discipline, I am able to better guide my selection of research

methodology. Existing research within software requirements show effective elicitation of human errors by using survey instruments.

### 3.1.3 Methods reported in Human Error Theory and software development

Huang et al. (2014) collect quantitative data from an empirical study using a questionnaire and programming contest. Huang (2016) used a screening questionnaire and collect quantitative data by using an experiment where participants completed a programming contest. Huang & Liu (2017) conduct a case study to investigate DPeHE. Huang & Liu (2017) collect quantitative data from participants using a questionnaire designed using likert-scale questions and data about the defects found in the software system. Huang & Liu (2017) collect qualitative data form participants by using open questions in the questionnaire. Huang & Liu (2017) report that the quantitative data was used to explore the effectiveness of DPeHE while, quantitative data was used to gain an understanding of how DPeHE affected ability to prevent errors. This shows a variety of methods that can be employed within SE Human Error Theory research. Case studies while being rich and insightful are time costly and require commitment from industrial collaborators. Survery instruments and short experiments are rather quick to complete, therefore reducing the time and cost overhead on participants.

Research conducted about Human Error Theory and software development has focused on understanding what type of human errors have been made and testing whether prevention strategies work. The third objective of my doctoral research is to understand how industry software developers currently mitigate against human errors. The fourth objective of my doctoral research is to deliver a tool which aids industry software developers to mitigate against the most frequent forms of human error. I have learned how other researchers have used a variety of methods to gather qualitative

and or quantitative data to understand similar objectives. This allows me to consider whether a different methodology could provide a varying insight on the objectives I intend to address in my doctoral research.

### 3.1.4 Summary

The literature shows that empirical research is the common route with human factors research in software development, Human Error Theory research in software requirements and Human Error Theory research in software development. This provides a strong foundation to using empirical research methods when selecting the research methodology for my research. There is no clear leading choice of empirical research methodology for human factors research in Pirzadehs' findings. I will be considering the benefits and drawbacks of each method given the time/cost/access to industrial software engineers/other constraints of my doctoral research. Reviewing Human Error Theory research in software requirements and Human Error Theory research in software development, the scope of methods used is narrowed down to primarily surveys (questionnaire), case studies and design (experiment). Each research method has its advantages and disadvantages, given the time/cost limitations of my doctoral research, it is clear that a combination of empirical research methods may be best suited to addressing the research problem. This will allow me to utilise survey methods to elicit human errors from software developers and run experiments to see if safeguards are effective.

## 3.2 Research Questions

This section presents the research questions for my doctoral research. I used Easterbrook et al. (2008) to inform me while developing my research questions. In Section 1 I set out the aim and objectives, in order to reach these I ask the following research questions:

- **RQ1** What SB errors do industry software developers experience during development? *This research question allows me to understand what SB human errors software developers make and then start to form a virtual Swiss Cheese Model.*

- **RQ2** How do industry software developers mitigate the SB errors they experience during development? *This research question allows me to understand what software developers do to mitigate SB human errors make and then complete the virtual Swiss Cheese Model created in RQ1.*

- **RQ3** Do industry software developers make more slips/lapses compared to mistakes? *This research question aims to understand whether software developers do indeed make more SB errors i.e. slips and lapses vs RB and KB errors i.e. mistakes. This allows SE researchers to correctly target safeguards at more frequently occuring errors types.*

- **RQ4** Does the online training package on the OODA loop reduce the number of human errors that software developers make? *This research question allows me to understand whether the safeguard, in this case online training package, is effective at reducing human errors.*

- **RQ5** Do developers find the online training package easy and useful to use? *This research question aims to better understand developer experience of the online training package. Should developers struggle to use the online training package this is something that would need to be factored in when analysing the reduction of results.*

In Figure 3.1 I show when and where each research question will be answered alongside which research method(s) will be employed.

My doctoral research is broken into two key studies. The first looks to understand what human errors occur and how these are currently mitigated.

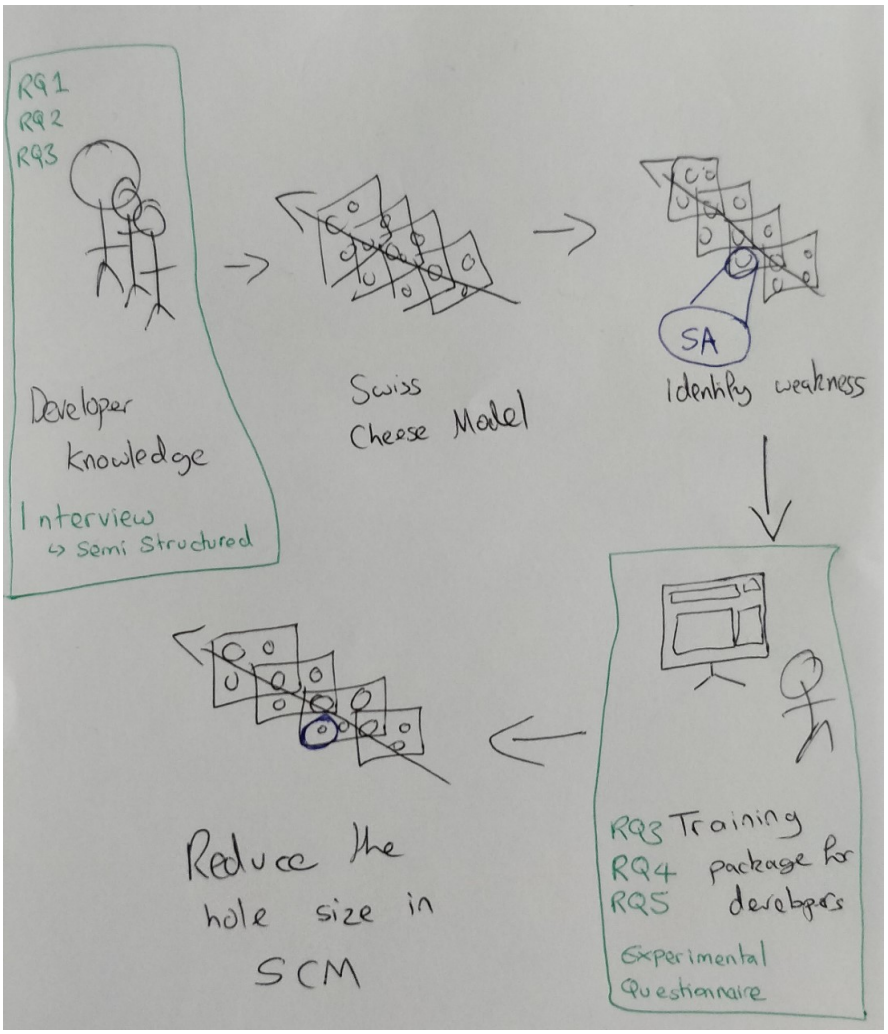



Figure 3.1: Research Methods to Conceptual Model

The second looks to evaluate a training package which employs SA. I conduct a survey study to answer RQ1 and RQ2 where I employ semi structured interviews. A survey instrument was selected so I could increase the number of participants within a limited time frame and make optimal use of the limited time each industrial software engineer could offer. I conduct an experimental study to answer RQ4 and RQ5 where I employ questionnaires, participant logs and training results. This experimental study was chosen to best compare the number and type of human errors industry software

developers make pre and post using the training package on SA. RQ3 is answered from the data obtained across both studies, as they both look to understand what types of human errors industry software developers make.

## 3.3 Study One: Survey Method

This study looks to answer RQ1, RQ2 & RQ3. These questions aim to under the types of human errors software developers make and the mitigation strategies that they employ to safeguard themselves against these errors.

The primary focus of this study is uncover information about developer human errors from developers. The could be done by using ethnographic research and studying them in the workplace, survey research and asking them to report on human errors. Ethnographic graphic research was immediately ruled out due to time constraints on access to industry software developers from my industrial collaborators.

This led me to surveys and exploring questionnaires vs interviews. While questionnaires could get a greater reach in terms of participants, the data collected could potential not be as descriptive as required which could lead to interpretation threats or me not being able to use the data. Interviews allow me directly question participants and clarify any misunderstandings or obtain additional contextual information if required.

I decided to use semi structured interviews as the data collection method for this study. By using a series of predefined questions I was able to ask each participant the same thing, while being able to probe where necessary to gain any other useful data from the participant. Semi structured interviews are however time consuming, costly and subject to a number of threats e.g. interview bias, descriptive and interpretation validity.

The vast majority of interviews were conducted face to face in participants work places. These interviews were conducted in meeting rooms at pre arranged times. Thought was given to the layout of the room so that

neither the interviewer or interviewee were distracted by anything going on outside the room. Some interviews had to be conducted virtually using platforms such as Skype. These were pre arranged with the participants. I was fortunate that my interviews didn't experience technical difficulties or interruptions that could be expected while working from home.

I conducted all the interviewees myself and was reminded myself of interviewer bias before each interview. By sticking to the pre arranged questions and only opening up the discussion when appropriate, I was able to limit any interviewer bias. I only further questioned a participants response when information was missing or unclear. I kept the interview style as semi restrictive so as to allow me to probe the participants responses as and when required. I kept the tone of the interview casual and relaxed so that participants did not feel intimidated when being asked about human errors they had made. This casual style is intended to try and strike up some rapport and allow participants to feel comfortable with sharing instances of human error.

When designing my interview questions I aimed to use open-ended questions, this would allow participants to respond with rich and descriptive answers. In order to simplify the process I am kept the language simple so that each participant was able to easily understand. I kept questions short and succinct and did not use negative phrasing.

In an ideal world, I would have like to use observations with a think aloud protocol to really understand exactly what is going on when a developer makes a human error. There are a number of factors that impact on a developer at any given time including personal stressors, which are not always apparent when asking industry software developers about a given error retrospectively.

Seaman (1999) reports that interviews are used to collect historical data, opinions or impressions about something, identify terminology used in a

given setting from interviewees. There are three main forms of interview, these are structured, semi structured and unstructured. Structured interviews are used to elicit answers to specific questions the interviewer has pre set. Unstructured interviews are used to gain answers to open ended questions and are ideal to gather as much information on a a broadly defined topic. Finally semi structured interview are designed to be a mix of both structured and unstructured as such they use a mixture of open ended and specific questions (Seaman 1999).

In addition to the three main forms of interview there is also narrative and life history interviews. In narrative interviews the key focus is on getting the interviewee to talk about an event or issue as a narrative. In these interviews, interviewees must be careful when telling the story as it is easy to jump through life stories along the way. Interviewers should refrain from asking questions during the narrative part and make notes about the important parts to ask questions on after (Morris 2015).

Interviews in which interviewees tell their life story while focusing on key aspects are called life history interviews. This interview technique maybe used to determine why the interviewee is in a certain state e.g. been in an abusive relationship. This interview technique does cross over with semi-structured interviews as the interviewer does provide some guidance where necessary. This interview technique can be hard to do well as it requires a lot of trust between the interviewer and interviewee (Morris 2015).

Seaman (1999) reports that interviewers should commence interviews with a brief introduction about the research being conducted. The content of this interview introduction should be carefully considered such that the interviewees do not bias unconsciously bias there responses by leaving out information they feel is not relevant to the interviewer. The interviewer must also keep track of an interview to ensure that it stays on track especially in the unstructured interviews. The interviewer must carefully guide the

interviewee back on track or interrupt them in a non abrupt or rude manner. Interviewers may feign ignorance to gain information from the interviewee, but also to build a rapport and eliminate any feeling that the interviewer is an expert (Seaman 1999).

The loss of information is something that should be avoided at all costs, where it does occur the researcher must consider at what point it will have the least bias on the findings. There are two main stages in which information loss can occur which are data collection and coding. Below I briefly explore each of these and what kind of information in addition can be lost in addition to the 'recoverable data' in each stage (Oppenheim 1992).

In the first instance we explore data collection, so if the answer to a question has been lost, we can simply re ask the question. While this is recoverable information we cannot guarantee the answer they give is the same as previously given as the thought process maybe different. Additionally the researcher has lost information relating to the tone of the candidates voice, facial expression, hesitations, digression's etc (Oppenheim 1992).

In the second instance we explore coding, which can take place in the field or in the office. Irrespective of whether coding occurs in the field or in the office, attention must be given to amount of categories created as this could affect the statistical analysis if categories are combined. Where coding occurs in the field there is a potential for bias as the interviewer has to fit the respondents response in to an appropriate category. This bias has the potential of being further increased if the interviewer is under pressure to conduct the interviews again and recollect the data as quickly as possible. On the other hand if coding occurs in the office then the coder has more time to review the recorded responses and categorise them appropriately (Oppenheim 1992).

## 3.4 Study Two: Field Experiments

This study looks to answer RQ3, RQ4 & RQ5. These questions aim to under the types of human errors software developers make and determine whether training software developer situation awareness can be a method to reduce the number of human errors made.

The primary focus of this study is evaluate whether a intervention has led to a reduction in the number of reported human errors by participants. The could be done by using field experiment or controlled experiment. A controlled experiment was ruled out quickly due to the time input required from developers each day i.e. it would have been too time and financially costly to put developers into a controlled setup for a two week period.

This led me to a field experiment, where developers would continue their daily activities as normal and only log instances of any human errors that occurred during the experiment. The intervention would be applied during the experiment and then I would be able to see if the number of human errors had reduced or not. A field experiment is an experiment conducted outside of laboratory / controlled conditions and typically occur in real world settings.

By allowing developers to continue their with there jobs, the time and financial overhead was significantly reduced as they only have to contribute to the study when they made a human error. This allows the results of the study to more accurately reflect the impact the intervention would have on developers i.e. they are continuing to develop real world solutions on real projects with real world problems surrounding development.

A drawback of using field experiments is that there is limited control over extraneous variables which could bias results. This could potentially make it harder for future researchers to replicate the study in an identical way.

A field experiment needs to be planned thoroughly to ensure the research

is conducted correctly and the appropriate tools and measures are used. A sample needs to be selected in my research this is a convenience sample. The research design needs to outlined alongside the tools for data collection and procedure to be used. Finally consideration needs to be given to any appropriate statistical analysis.

In addition to field experimentation I make use of a short online questionnaire to help answer RQ5. Given the short number of questions, and responses were predominately defined a questionnaire was the ideal selection. It meant that participants could complete the follow up questionnaire in there own time, and eliminate time in transcribing an interview for something that can easily be administered via questionnaire.

In this online questionnaire, I use a combination of free text and simple rating scales of 'Yes/No' and 'Easy/Average/Difficult', in favour of using more standardised Likert scale ratings. This was to help simplify the questionnaire and make it easy for participants to respond. By using a binary scale and then providing a freetext option later, participants are able to expand if they wish. Using 'Easy/Average/Difficult', instead a 5or 7 point Likert scale makes it very simple for participants to understand what they are selecting when completing the questionnaire.

In an ideal world, I would have like to use observations with a think aloud protocol to really understand exactly what is going on when a developer makes a human error. Not only would it allow me to better understand what is going on when a developer makes an error (RQ3), but it will give a rich insight into how developers are applying OODA loop training (RQ4). There are a number of factors that impact on a developer at any given time including personal stressors, which are not always apparent when asking industry software developers about a given error retrospectively.

Sjøberg et al. (2005) report that the method for identifying cause-effect relationships is to conduct controlled experiments. Shadish et al. (2002)

define experiment as a study in which an intervention is deliberately introduced to observe its effects.

Basili (2007) report that controlled experiments need to be replicated with varying conditions, designs and allowing for evolving questions to be answered. Due to limitations of time and cost, my doctoral research will not be replicated. By using developers from different companies, countries and experience levels I hope to have a data set which represents varying conditions.

Ko et al. (2015) report of 10 key components when designing a tool evaluation experiment. While my end solution is not a tool and is an online training package the concept is very similar. I carefully evaluated the key components and ignored two of them as I deemed them unnecessary. These were *group assignment* as the online training package is designed to completed alone and *training* as the online training package is easy to navigate. In addition the online training package went through a pilot study phase which confirmed the ease of navigation.

Sjøberg et al. (2003) report using professional system development tools in an experiment increases the realism, but requires careful planning, risk analysis and more resources. The simplicity of my online training package meant that participants had to follow a simple set of instructions which were delivered as a how to guide via email. This simplicity meant that I could afford not to add a system development tool and not add an extra system for the participants to interact with.

## 3.5 Analysis Techniques

The data I have collated throughout my doctoral research is predominately qualitative. In this section I discuss the coding techniques used.

The coding process of qualitative data becomes difficult when the data is subjective opposed to objective as the terminology used to describe it

varies and is difficult to interpret. During the coding process it is imperative that there is minimal information loss during the transformation phase thus maintaining a high level of accuracy. The researcher may find problems during the transformation when it comes to different words used to describe the same phenomenon or the same words used to describe different phenomenon (Seaman 1999).

Basit (2003) reports that coding is one of the most significant steps taken during analysis to organize and make sense of textual data. Raw data can be extremely interesting to look at however it is difficult to gain much with out conducting some for of systematic analysis on it. The process of coding involves subdividing the data as well as assigning categories (Dey 2003).

Coding can either be conducted as inductive or deductive. Inductive coding is where codes are derived from the data. Deductive coding is where codes are applied from a pre defined list (Linneberg & Korsgaard 2019). In my research there is a need to use both inductive and deductive coding methods, for example human errors are already pre defined and therefore deductive coding will be applied. While coding participant instances of these errors in the work they have conducted will be inductive coding.

Hoda et al. (2010) report that grounded theory is being increasingly used in the study of human aspects within SE. Stol et al. (2016) report using version control, project management and team communication systems to aid in the management of large amounts of heterogeneous data.

I conduct a thematic analysis (Braun & Clarke 2006) of the coded themes and sub-themes. My approach was very similar to that used by Meyer et al. (2019) in their analysis of reflective goal setting by developers. My supervisor and I extensively discussed commonalities between the sub-themes.

For each study coding was broken down in to various stages. Typically this took the form of verification that the described human error indeed matched that of the category it was described in. This is to say that when

coding omission's is the described human error an omission, if not move it to the correct category of human error. The next stage was to assign themes/sub-themes the data using predefined themes or generating our own. Following on from this both my supervisor and I had a discussion about each instance (e.g. a human error log in a log sheet). Where there was a disagreement in the themes/sub-themes applied an extensive discussion was had until agree ability was met.

As recommended by Kitchenham et al. Kitchenham et al. (2010) where any disagreement on classification occurred between my supervisor and I 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.

It is important to note that in some instances sub-themes may be allocated to multiple themes as where appropriate therefore figures may not tally up entirely.

## 3.6 Pilot Study

No version of a questionnaire is perfect on the first attempt, it is refined multiple times to rule out various issues and ensure it will aid in answering the research question / aim. Before putting a questionnaire to the actual group of target participants we should aim at targeting a subset of people who come from a similar background or are a subset of the target audience. By doing so we can ensure that the questions are appropriate and worded correctly, so that the respondents can correctly answer the questions.

To ensure the pilot study is effective in its execution we need to ensure that we target similar professionals. We have two options when it comes to select groups for the pilot study, the first being a study set of the actual

target group. The second is taking a group of similar professionals who are outside of the target group. Having reviewed the options and size of the desired target population we decided to use a similar group so as to not use the intended target.

By ensuring the pilot group is a similar group or a subset of the target group we can be sure that feedback gained from the pilot group will be useful feedback and make the questionnaire more understandable as the pilot respondents will come from a similar background. Dependant on feedback and the target audience of the questionnaire it may require multiple iterations of pilot study.

I decided to target PhD students, researchers, academics who have experience at writing code and active software developers internal to Brunel University London as my target audience for the pilot studies. The pilot candidates will be targeted using email or social media. This will be achievable as the target population are known colleagues of the research team. While the sample of participants to be used in the pilot study are easy to access, the typically aren't full time industry software developers. There is a definitive experience difference in the sample of participants given how some develop industry code, some write academic code / personal projects and others no longer actively write code.

Below I briefly explain how the pilot study was executed for Study 1 and Study 2 respectively in Sub Section 3.6.1 & 3.6.2.

### 3.6.1 Pilot of Study 1

I recruited 3 doctoral researchers from the Department of Computer Science at Brunel University to participant within my pilot study. All three of these participants have worked within industry prior to commencing their doctoral degrees.

I conducted the all the interviewees in an iterative manner, that is; con-

duct interview, discuss the interview with the participant, make appropriate changes to the script and then repeated with the next participant.

### 3.6.2 Pilot of Study 2

I recruited 3 doctoral researchers from the Department of Computer Science at Brunel University to participant within my pilot study. All three of these participants have worked within industry prior to commencing their doctoral degrees.

I condensed the time in which each experiment took to conduct. I eliminated all the days and focus on the contact points and activities that needed to be completed. I conducted a study introduction, ask my pilot participants to complete some errors in the log sheet. I then asked them to complete the intervention. Any necessary feedback on the log sheet will have already been obtained so I do not asked them to complete it post intervention. I then ask the pilot participants to complete the end of study follow up questionnaire. I conducted the all the experiments in an iterative manner, that is; conduct interview, discuss the interview with the participant, make appropriate changes to any study documentation, log sheets, intervention material, follow up questionnaire and then repeated with the next participant.

By iterating through participants in this cycling method I am able to build on feedback. If I had approached pilot study participants all at once, I could have got the same / similar feedback from all. This way more things can come to light and be addressed as small issues get fixed between pilot cycles.

## 3.7 Participant Type: Student, Professional or Both

What type of participant should be used for my study, student, professional or both. In this section I explore the literature to determine what the correct

approach is and also highlight the approach that I will use for my doctoral research.

Sjøberg et al. (2002) reports that students typically used in software engineering research as they are more accessible and easier to organise. Sjøberg et al. (2002) argue that there are a variety of differences between student and professional, examples include; experience and skill level, use of professional methods and tools and team work versus individual work. Sjøberg et al. (2002) report that differences amongst students and professionals may be large enough that the subject category could be irrelevant and the difference accounted for as one of many characteristics of being a software engineer.

Vegas et al. (2015) reports that within the area of empirical software engineering, lab experiments have become common practice. Currently it is unclear as to how generalizable the results of a lab experiment are given that they are simplifications of real world situations (Vegas et al. 2015). Vegas et al. (2015) reports that typically students are used over professionals, participants conduct exercises or create toy software instead of working on real projects and participants are typically putting what they have learned into practice instead of harnessing developer knowledge and experience. Vegas et al. (2015) reports of 15 difficulties regarding securing company involvement in running an experiment. Vegas et al. (2015) report that industrial environments have imposed more constraints than laboratory environments and professionals were under motivated and performed worse than students.

Salman et al. (2015) conducts an experiment to determine whether students are representatives of professionals in software engineering experiments. Salman et al. (2015) perform a Test Driven Development experiment on the subject groups; student (17 participants) and professional (24 participants). Salman et al. (2015) asks “How much does the code quality of a task produced by students using Test Driven Development differ from the code quality of a task produced completed by professionals using Test

Driven Development?" and "How much does the code quality of a task produced by students using Test Last Development differ fromt he code quality of a task produced by professionals using Test Last Development?".

Salman et al. (2015) report that there was a difference in code quality when Test Last Development was used, however both groups performed similarly when Test Driven Development was used. Salman et al. (2015) find that the difference in code quality may be due to the tasks the groups were assigned. Salman et al. (2015) conclude that there neither subject group outperform each other when applying a new technology during experimentation.

Falessi et al. (2018) report the opinions of a group of experts on the use of students and professionals as subjects in software engineering experiments. They ask three research questions which are 'What are the pros and cons of using professionals and students as subjects in software engineering experiments?', 'As a community, what do we agree and disagree on with respect to subjects in software engineering experiments?' and 'Is it possible to characterize subjects using a different classification than professionals or students?' Falessi et al. (2018) conducted a focus group during a session at ISERN 2014, in which 65 empirical researchers including the authors argued on the challenges with using students as subjects in software engineering experiments. The authors consolidated there findings in to 14 statements. These statements were evaluated by the ISERN session attendees in fall 2016 by completing a questionnaire which reported their personal level of agreement with the statements.

Falessi et al. (2018) find that neither subject category is always better than the other. Professionals come at a higher cost and are not as accessible as students, although professionals are more representative (Falessi et al. 2018). It is impossible to say that students or professionals will always be better than the other due to the number of pros and cons that each subject

category poses (Falessi et al. 2018). Falessi et al. (2018) report that the community agrees with statement nine and fourteen. The statements are "ST9 - We should think about population and validity already before conducting the experiment, at the time when we are planning to use convenience sampling" and "ST14 - The suitability and representativeness of students as proxies for professional developers change with different contexts and with different types of population.". Given that students and professionals are not mutually exclusive subject categories, Falessi et al. (2018), propose a scheme in which subjects should be categorized. This scheme categorizes subjects with respect to: real experience, relevant experience and recent experience (Falessi et al. 2018).

Höst et al. (2000) investigates the difference between student and professional groups by assessing the difference on ten different factors which affect the lead time of software development projects. Höst et al. (2000) reports that there are minor differences in conception and no significant differences in correctness between the two groups. Höst et al. (2000) reports that final year software engineering students are relevant as subjects in empirical software engineering research.

Reviewing the findings it becomes apparent that neither student or professional are better sources of subject in empirical software engineering research. Salman et al. (2015) ask questions of code quality differences between students and professionals. There is a similarity with what I plan to do and this, where I will ask developers to develop a solution using a tool and assess the number of faults detected after. This presents an argument to use a combination of both students and professionals. Falessi et al. (2018) report that the suitability of a group of subjects changes given the context and type of population. Falessi et al. (2018) propose a scheme which factors in real experience, relevant experience and recent experience.

As it is clear that neither group is better than the other, I now explore

the literature to see what subject category has been used. Hu et al. (2017, 2016), Anu et al. (2016*a*), Huang (2016), Huang et al. (2014), Anu et al. (2017, 2016*b*) use computer science students (70%) at undergraduate (50%) or postgraduate level (20%). Huang & Liu (2017), Huang et al. (2012), Hu et al. (2017a) use professionals (30%) made up of practitioners (10%) or combination of practitioners and researchers (20%). Of the research conducted using students 20% is focused on software development. Of the research using professionals 20% is focused on software development. This shows that human error research within software development uses professionals over students.

For my research I will be aiming to use research participants who have experience at writing code on industry projects. By doing so I am able to use participants who may have retrained e.g. become a researcher or participants who have gone back to study for postgraduate qualifications. When seeking participants for pilot studies I will be aiming to get participants who have experience at writing code.

## 3.8 Sampling

In any qualitative research a sample of the population is used and studied. Marshall (1996) reports that there are three sample strategies when it comes to selecting a sample for a qualitative study; convenience sample, judgement sample and theoretical sample. These samples can be defined as;

- A **convenience sample** involves selecting the most accessible subjects. (Marshall 1996)
- A **judgement sample** involves the researcher actively selecting the most productive sample. (Marshall 1996)
- A **theoretical sample** involves building interpretative theories from

> the emerging data and selecting a new sample to examine and elaborate on this theory. (Marshall 1996)

Having had keen industrial collaborators from the beginning of my research, I have used a convenience sample due to the ease of access to highly desirable participants. I did set out basic preconditions which said any participant should be a practising software developer on industrial projects. This is automatically met by using the convenience sample of industrial collaborators.

### 3.8.1 Sample Size

In this subsection I will consider the sample size required for doctoral research. The sample size will vary depending on what stage of research is being conducted i.e. is it a pilot or actual study, is the sample a group of experts or industry practitioners?

Kadam & Bhalerao (2010) reports that if a sample size is too small the results are not generalizable, as the sample is not an accurate representation of the target population. Kadam & Bhalerao (2010) reports that if a sample size is too big the researcher runs the risk of subjecting participants to unnecessary interventions and wasting valuable resources e.g. participant/researcher time.

In order to determine the sample size, we must first determine what the confidence level, confidence interval and population are. The confidence level is a percentage which indicates how sure you are that the results are going to be true. The confidence level is typically set at 95%. The confidence interval is the range within which the respondents would have picked the same answer. The confidence level is typically set at 5%. I define the population as software developers within an industry setting as this is the focal point of my doctoral research. Thibodeau (2013) estimates that there are 18.2 million software developers worldwide, rising to 26.4 million software

developers in 2019.

Using the sample size calculator provided by `www.surveysystem.com/sscalc.htm`, I set the confidence level to 95%, confidence interval to 5 and population to 26,400,000 (population as estimated by Thibodeau (2013)). The sample size required is 384. Wagner et al. (2020) report of a similar figure, they suggest a sample size of 400 to attain generalisability.

Barlett et al. (2001) report that researchers use inadequate sample sizes as they are faced with various constraints. These constraints include; time, personnel or other resource limitation. Barlett et al. (2001) report that in instances where inadequate sample sizes have been used, researchers should report the appropriate sample size, used sample size, reasons behind using an inadequate sample size and discuss the effect of using an inadequate sample may have upon the research study.

Trying to recruit 384 participants on two separate occasions is not going to be a trivial task. This presents me with a constraint as described by (Barlett et al. 2001). I will now looking at the literature to see what type of sample size has been used. Hu et al. (2017a) do not report how many professionals and researchers are used.

Looking first at the participant numbers where undergraduate students have been used. Huang et al. (2014) reports using 70 undergraduate students. Huang et al. (2012) reports using 55 undergraduate students. Hu et al. (2016) reports using 28 undergraduate students. Anu et al. (2016*a*) reports using 46 undergraduate students. Hu et al. (2017) reports using 31 undergraduate students. The mean number of participants where undergraduate students are used is 46.

Looking next at the participant numbers where postgraduate students have been used. Anu et al. (2017) reports using 34 postgraduate students. Anu et al. (2016*b*) reports using 17 postgraduate students. The mean number of participants where postgraduate students are used is 26.

Finally looking at the participant numbers where professionals have been used. Huang & Liu (2017) reports using 14 professionals. Huang (2016) reports using 15 professionals and 1 researcher. The mean number of participants where professionals are used is 15.

I have learned how the sample size vary depending on the participant group. Given that I aim to use industry software developers for my research I aim to realistically get 20 participants per study. 20 industry software developers would be more than any other study has used to investigate human error within software development to my knowledge. Having looked at the mean number of participants from studies in the area I believe 20 industry software developers is a sufficient number for a small scale study.

Connelly (2008) reports that a sample size for a pilot study should be 10% of the actual study. This means I need to get a minimum of 2 participants per pilot study.

To conclude I have learned that there are many factors that can influence an ideal sample size representing the population. I noticed that I am likely to encounter constraints on recruitment of participants during my research. I have explored what sample sizes other researchers have used and will base my sample size of similar values.

## 3.9 Empirical Validity

Careful attention must be paid to construct, internal and external validity and reliability in order to ensure that the conclusions drawn from the research can be seen as valid (Easterbrook et al. 2008). Kitchenham et al. (2002) highlight guidelines for conducting empirical software engineering research, which explore all angles of research. These guidelines aid in ensuring that construct, internal and external validity threats are minimised to acceptable levels.

Easterbrook et al. (2008) reports construct validity addresses whether theoretical constructs are interpreted and measured correctly. As such an example would be if an experiment designed to test the efficiency of a network, that the researcher's interpretation of efficiency is the same as other researchers.

Easterbrook et al. (2008) reports that internal validity focuses on whether the study has been designed and conducted properly. An example of an internal validity threat would be comparing a control group of average students taught to program using conventional teaching methods against an experimental group of excellent students taught to program using novel teaching methods.

Easterbrook et al. (2008) reports that external validity addresses whether claims for generality of results are justified. As such if a given piece of research was focused at developers within Capability Maturity Model Level 5 organisation, it would be difficult to convince readers that the results are applicable to first year undergraduate computer science students.

Reliability focuses on the repeatability of a given piece of research. As such it contains details of the conducted research at a level such that another researcher could conduct the same research and gain the same results. Easterbrook et al. (2008) highlight that potential problems could be identified if another researcher can not gain the same results, it could highlight that the researcher bias has been introduced. This could be because the researcher had a stake in that particular piece of research perhaps the requiring the successful evaluation of a tool.

Maxwell (1992) reports a number of threats to validity when it comes to qualitative research, some of these include; descriptive validity, interpretation validity, theoretical validity and generalisability.

Maxwell (1992) reports that descriptive validity addresses whether a researcher has accurately recorded what they heard a participant say and

or do. Walsh (2003) discusses the same concept at 'credibility'. Has the researcher accurately recorded what was said during the interview. If factors like stress and tone of voice are important, have these been recorded? In my research it is only important to ensure that the transcriptions accurately record what has been said.

Maxwell (1992) reports that interpretation validity addresses whether a researcher has captured the observation as interpreted by the participant being researched.

Maxwell (1992) reports that theoretical validity addresses whether the validity of the researcher's concepts and the theorized relationships among the concepts in context with the phenomena.

Auerbach & Silverstein (2003), Maxwell (1992) report that generalisability addresses whether you can apply the theory resulting from the study universally. Within qualitative research this is difficult as any body of research hones in on the themes and sub-themes identified within the sample group being studied. My research is exploratory in nature and would require a much larger sample size before generalisability can be considered.

## 3.10 Summary

In this chapter I explore the methods that have been employed in studies throughout Human Error Theory literature, examine the theory behind survey methods along side the appropriate analysis techniques. Key takeaways from this chapter are:

- **Historical Methods used in Human Error Theory research** An array of methods are used within Human Error Theory research including; survey, case study and experimentation. Some methods are used more often for eliciting human errors from participants e.g. surveys.

- **Methods to Used** I will employ survey methods to elicit example of human errors from industry software developers. I will employ controlled experiments to determine whether a safeguard (online training package on SA) is effective at reducing the number of human errors made.

- **Participant Types** While students are easily accessible and commonly used with SE Human Error Theory I will be using industrial software engineers. This is to prevent anomalous human errors (specifically human errors reported by students alone) being reported.

- **Sample Size** An ideal sample size would be 384 industry software developers. Recruiting such a number of software developers could be challenging, having looked at similar research I aim to recruit 20 participants in a study.

# Chapter 4

# What SB errors do industry sofwtare developers make and how can we mitigate these? - An Interview Study

This chapter develops an understanding of human errors within software developer. This will allows for the identification of frequently occurring errors and lead to reduce the number of defects introduced into code as a result of human error by industry software developers. I need to first understand which type of SB human error occurs most within the development phase of the SDLC and how industry software developers mitigate these SB errors. This will allow me to build a basic Swiss Cheese Model and identify which areas are in most need of attention.

Section 4.1 presents the research questions I ask in this study. Section 4.2 details the approach taken for the study. Section 4.3 presents the results for this study. Section 4.4 presents the threats to validity for this study. Section 4.5 presents a summary of this chapter.

## 4.1 Research Questions Addressed In This Study

In this chapter I address the following research questions;

- **RQ1** What SB errors do industry software developers experience during development?
- **RQ2** How do industry software developers mitigate the SB errors they experience during development?

## 4.2 Approach

To understand developer experiences of the errors they make during development I used qualitative research in the form of semi structured interviews. I iterate through the eight SB errors described by Reason, describing each error and a non software engineering related example. I then ask developers to describe instances where an error has occurred in the development work they have done. Additionally I ask developers to describe any mitigation factors they have subsequently put in place to safeguard themselves against the error in the future. (See Appendix B.2 for more detail on the definitions and associated real world examples provided) I implemented a grounded approach to coding the qualitative interview data collected. I now detail the study participants and their recruitment, the interview process and data analysis involved in our study.

### 4.2.1 Participants & Recruitment

The study was performed over a sixteen week period. During this time I interviewed 27 industry software developers. Those interviewed included: 4 women and 23 men aged between 18 and 54. Of these industry software developers 14 had more than ten years of industry experience. These industry software developers were a convenience sample and had been contacted by

word-of-mouth, email and face-to-face. Table 4.2 provides a full breakdown of the participants who were recruited for this study. In addition to this I shared details of the study and issued invitations to participate on social media, with posts on LinkedIn and Twitter.

Table 4.1 shows how many views, likes and shares each post gained. When reviewing the figures in the views column it is important to note that this reflects how many people have seen it. This is not an indication of how many people have read the post fully or followed up on details. I do not have any demographic information about the users that viewed the posts therefore I cannot be sure whether they are views from the intended target group i.e. industry software developers.

| | **Views** | **Like** | **Shares** |
|---|---|---|---|
| **Twitter** | 6340 | 18 | 19 |
| **LinkedIn** | 632 | 5 | 2 |

Table 4.1: Social Media Views For Participant Recruitment

Please see Table 4.2 for demographic data of participants of the interviews.

| | **Gender** | **Age Range** | **Years of Industry Experience** | **Primary Job Role** | **Primary Industry** | **Primary Language** | **OSS vs CSS** |
|---|---|---|---|---|---|---|---|
| $P_1$ | Male | 18-24 | 1-3 | Software Developer | Software Industry | Python | CSS |
| $P_2$ | Male | 35-44 | 4-7 | Software Developer | Software Industry | Python | CSS |
| $P_3$ | Male | 18-24 | <1 | Data Analyst | Healthcare | R, Bash and Matlab | CSS |

| | **Gender** | **Age Range** | **Years of Industry Experience** | **Primary Job Role** | **Primary Industry** | **Primary Language** | **OSS vs CSS** |
|---|---|---|---|---|---|---|---|
| $P_4$ | Male | 35-44 | 10+ | Lead Scala Developer | Fashion | Scala | CSS |
| $P_5$ | Male | 35-44 | 10+ | Senior Developer | Meetings and Events | PHP | CSS |
| $P_6$ | Female | 35-44 | 8-10 | Senior Developer | Meetings and Events | PHP and JavaScript | CSS |
| $P_7$ | Male | 25-34 | 4-7 | Developer | Meetings and Events | PHP and SQL | CSS |
| $P_8$ | Male | 35-44 | 10+ | Solutions Consultant | Meetings and Events | C#, Pascal, Fortran and C++ | CSS |
| $P_9$ | Male | 45-54 | 10+ | Technical Architect | Meetings and Events | PHP | CSS |
| $P_{10}$ | Male | 25-34 | 4-7 | Product Manager | Tech (Industrial Applications) | Python | CSS |
| $P_{11}$ | Male | 35-44 | 10+ | Director | Digital Innovation in Manufacturing | Python | OSS |
| $P_{12}$ | Male | 45-54 | 10+ | Python Developer | Contract programmer for aid organisation | Python | CSS |

| | **Gender** | **Age Range** | **Years of Industry Experience** | **Primary Job Role** | **Primary Industry** | **Primary Language** | **OSS vs CSS** |
|---|---|---|---|---|---|---|---|
| $P_{13}$ | Male | 35-44 | 10+ | Software Engineer | Finance | Java | CSS |
| $P_{14}$ | Male | 35-44 | 10+ | Developer | Finance | JavaScript | CSS |
| $P_{15}$ | Male | 25-34 | 4-7 | Oracle DBA Developer/Analyst | Inhouse IT | C++ | OSS |
| $P_{16}$ | Female | 45-54 | 10+ | Software Developer | Education | VB.Net | CSS |
| $P_{17}$ | Female | 35-44 | 8-10 | Software Developer | University (Software and Development Team) | .NET, c#, AngularJs, JavaScript, HTML, CSS | CSS OSS |
| $P_{18}$ | Male | 25-34 | 1-3 | Software Developer | IT Service Management | Scala | CSS |
| $P_{19}$ | Male | 25-34 | 10+ | Technical Architect | Meetings and Events | PHP | CSS |
| $P_{20}$ | Male | 25-34 | 4-7 | Agile JAVA Developer | Telecommunications | JAVA | CSS |
| $P_{21}$ | Male | 35-44 | 10+ | System Analyst | Education | JAVA | OSS |
| $P_{22}$ | Male | 35-44 | 10+ | Software Architect | Leisure | JAVA, Js | CSS |

| | **Gender** | **Age Range** | **Years of Industry Experience** | **Primary Job Role** | **Primary Industry** | **Primary Language** | **OSS vs CSS** |
|---|---|---|---|---|---|---|---|
| $P_{23}$ | Male | 25-34 | 8-10 | Backend Support Developer | Events Management | PHP | CSS |
| $P_{24}$ | Male | 25-34 | 1-3 | Research Assistant | Academia | C/C++ | OSS |
| $P_{25}$ | Male | 45-54 | 10+ | Principal Software Architect | Media & Entertainment | C++ Python | CSS OSS |
| $P_{26}$ | Male | 25-34 | 10+ | Engineering Manager | Broadcast Software | JavaScript | CSS |
| $P_{27}$ | Female | 35-44 | 1-3 | Software Developer | Software Development | Java | CSS |

Table 4.2: Demographic Data - Interview Participants

### 4.2.2 Interview Method

I describe the process taken during the data analysis phase as shown in Figure 4.1. In this study I used semi-structured interviews to question industry software developers about their errors during coding. I asked industry software developers how they mitigated their errors. Although the interviews had predetermined questions, there was an element of unstructured open ended questions that could be used as follow-ups. This was based on an interviewee's answers and the interviewer's interpretation.

Interviews were held between August 2018 and December 2018. I conducted all the interviews and so consistency was maintained across all in-

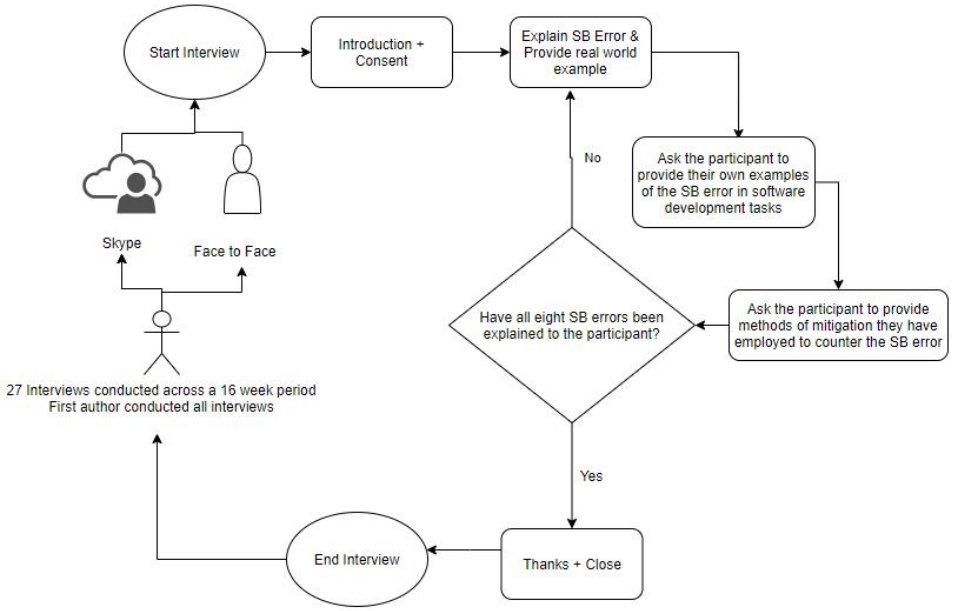



Figure 4.1: Interview Process for Interview Study

terviews. The interviews were either face-to-face, mostly at the developer's workplace, and video calls using Skype. The interviews lasted between 30 to 45 minutes. All interviews have been recorded and transcribed.

The structure of the interview was as follows;

1. Introduction and overview of logistics.
2. Explain each of the eight skill-based errors. Each explanation included providing a definition and a non software engineering-related real world example. Details of the eight skill-based errors and the non software engineering examples can be found in Appendix B.2 - Supporting Material For Study One.
3. Ask each developer about their error occurrences and mitigation strategies for each of the eight skill-based error types. A sample set of questions can be found in Appendix B.3 - Supporting Material For Study One.

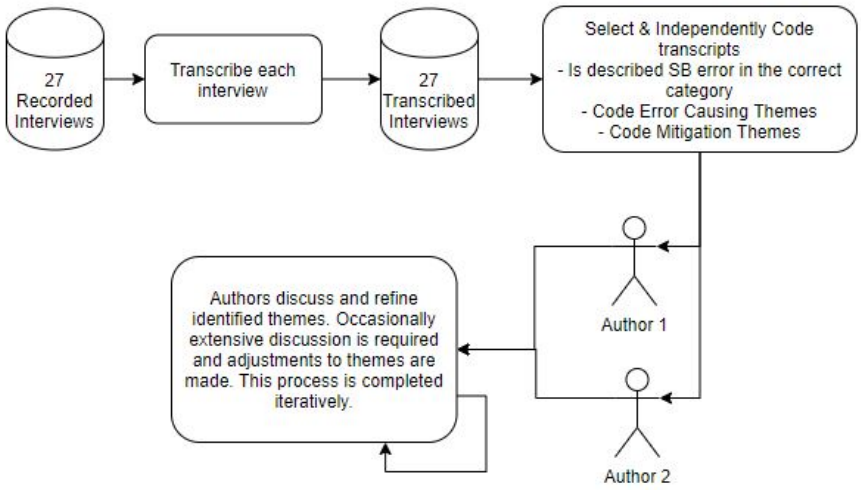



Figure 4.2: Data Analysis Process for Interview Study

4. Close and thanks. The demographic questions that I asked industry software developers can be found in Appendix B.4 - Supporting Material For Study One.

### 4.2.3 Data Analysis

Below I explain the data analysis process as depicted in Figure 4.2. Once the interviews were performed, each interview audio recording was then transcribed manually by myself to allow for coding and analysis.

I identified n sub-themes and grouped them into 4 themes for errors and mitigation strategies in the transcripts using Thematic Analysis (Braun & Clarke 2006). My approach was very similar to that used by Meyer et al. (2019) in their analysis of reflective goal setting by industry software developers. My supervisor and I extensively discussed commonalities between the sub-themes and noticed that these themes for mitigation strategies related well to the generic high level concepts identified by (Graziotin et al. 2017). These high level themes were: management, tools, processes and

the developer. Graziotin et al. (2017)'s scheme seemed a useful high level clustering of the mitigation strategies themes my supervisor and I extracted and so I adopted this theme structure and organised our mitigation strategy sub-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.

My supervisor and I open coded each reported error and mitigation strategy based on direct quotes from transcripts against the sub-themes established. To ensure that coding was conducted reliably my supervisor and I extensively discussed each error type prior to performing any coding to ensure their understanding was the same. Then my supervisor and I independently coded interview transcripts. When coding the transcripts my supervisor and I looked to:

1. Themes and supporting evidence for a given SB error.
2. Themes and supporting evidence for a given mitigation strategy.

I noticed that the error themes which emerged from the interview data were not all psychological events, i.e., were not errors in the true Human Error Theory 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 I had expected, it seemed to me that what industry software developers reported was nevertheless important. Consequently, I labelled each industry software developer reported 'error' as either: a reason for error, an error or the consequence of an error.

As recommended by Kitchenham et al. (2010) where any disagreement on classification occurred between my supervisor and I 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.

It is important to note that in some instances sub-themes may be allocated to multiple themes as where appropriate therefore figures may not tally up entirely.

Theme categorisation was documented using a custom kanban style board using Trello[1]. 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.

## 4.3 Results

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

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

The 27 interviews elicited 57 themes of errors across eight SB error types. Table 4.4[2] shows the number of error and mitigation strategies broken down by SB error type. It is important to be able to see if there is a significant difference in the number of inattention errors or over attention errors and or mitigation strategies being described by developers. It is also useful know if a given SB error type has a significantly larger number of mitiga-

[1] https://trello.com/en

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

| **Sub-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 4.3: Developer Error Themes (in ranked order)

tion strategies described compared with the errors described. A SB with a higher number of mitigation strategies to errors could be argued as better understood and or easier to safeguard against as developers know of more effective ways to safeguard themselves. Table 4.4 shows that Perceptual Confusion errors were mentioned most frequently with industry software 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 I 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.

| **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 4.4: Number of Theme Occurrences of each Skill Based Error Type

I also considered the errors mentioned by industry software developers in terms of developer error sub-themes as shown in Table 4.3. (Table 4.3 is a subset of a larger list of error sub themes, to view the list of all error themes see Appendix B.5). More details of each error theme in Table 4.3 (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). My results suggest that industry software developers focus more on the reasons for errors and their consequences, than the errors themselves (see Table 4.3). 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 4.3 suggests industry software 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 Peitek et al. (2020), but performing programming tasks within a complex development environment is likely to increase this cognitive load still further. For example, S1P17 told me that errors occurred when

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

S1P12 said

> 'typing commands into the wrong window'

caused errors and S1P10 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 4.3 shows that industry software developers also recognise that a frequent human error is their own concentration being impaired. Issues surrounding concentration, context switching, work pressure, rabbit hole[3],

[3]Going down a rabbit hole is a metaphor commonly used to indicate someone has gone

understanding and distractions were cited regularly by industry software developers which suggests that it is easy for industry software developers to lose their awareness of a given task:

- S1P26

  > 'You go and reply to the email and by the time you come back you have forgotten what you were going to do.'

- S1P13

  > '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...'

- S1P16

  > '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.'

- S1P22

  > '...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

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

> 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.'

- S1P16

> '...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 industry software developers as the consequence of an error. Duplicate code is a well-known bad smell so it is interesting that industry software 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 throughout the development process (for example, the London Ambulance System failure Beynon-Davies (1999)), so it is not surprising that industry software developers say these problems underpin some of the errors they make.

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

### 4.3.2 RQ2: How do industry software developers mitigate their Skill-based (SB) human errors?

I have classified the mitigation strategies mentioned by industry software developers into four themes which are the developer, processes, tools and management. Table 4.5 presents these themes and shows the number of

times each theme was mentioned by industry software developers during interviews. Table 4.5 shows that industry software developers see themselves as highly influential in mitigating errors, suggesting that industry software developers seem to take a great deal of personal responsibility for trying to prevent errors.

Table 4.6 shows the sub-themes of the developer mitigation strategies. Table 4.6 suggests that a large number of themes relate to the industry software developers' cognitive issues such as focus, concentration, discipline, attention, understanding and awareness. Most of these issues were discussed by industry software developers in terms of them using willpower to improve their coding behaviour. For example, industry software developers said:

- S1P10

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

- S1P16

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

- S1P12

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

- S1P16

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

- S1P12

  'Just by remembering.'

| **Sub-Theme** | **Developer** | **Process** | **Tools** | **Management** |
|---|---|---|---|---|
| Focus | 10 | | | |
| Concentration | 8 | | | |
| Use Headphones | 7 | | | |
| Awareness | 6 | | | |
| Discipline | 5 | | | |
| Learn | 4 | | | |
| Checklist | | 12 | | |
| Code Reviews | | 9 | | |
| Testing | | 8 | | |
| Note Taking | | 6 | | |
| Communication | | 6 | | |
| Documentation | | 6 | | |
| Pull Requests | | 3 | | |
| Automation | | | 9 | |
| Git | | | 5 | |
| Compiler | | | 4 | |
| Navigation Helper | | | 2 | |
| Planning | 3 | | | 3 |
| Training | | | | 2 |
| Best Practices | | | | 2 |
| Prioritisation | | | | 1 |
| Well formed, low level processes | | | | 1 |
| Awareness | | | | 1 |
| Other | 15 | 7 | 8 | |
| **Total Instances** | 58 | 57 | 28 | 10 |

Table 4.5: Mapping Mitigation Strategy Themes to Sub-Themes

These quotes suggest that industry software 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 4.6 shows a variety of developer-based mitigation strategies, it is surprising that reducing tiredness and taking breaks was not explicitly mentioned more frequently by industry software developers. The impact of tiredness on errors seems conventional wisdom. It is unclear why tiredness did not feature more directly in my results. It is clear that tiredness etc. can be a reason for industry software developers to lose their focus on a task Sarkar & Parnin (2017) so it is surprising that industry software developers did not mention this more often. Similarly it is surprising that interruptions were not mentioned more explicitly by industry software developers. Interruptions are widely thought to underpin errors Sykes (2011) but were not mentioned much in my study. Bailey & Konstan (2006) report that interruptions have a disruptive impact on completion time and error rate. Where interruptions were mentioned it was indirectly, with industry software developers talking about mitigation for errors like using headphones and turning emails off.

### Processes

Mitigation strategies related to processes were frequently mentioned by industry software developers. Most of these strategies are based on detecting the consequences of errors in terms of spotting faults in code. Table 4.7 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 re-

lation to faults. For example, some industry software developers mentioned:

- S1P13

  > '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.'

- S1P19

  > '...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 4.8 suggests that automation is an important element of mitigating developer errors. Industry software developers seem to rely on automation tools or tools that have elements of automation to aid in reducing their errors. For example, some industry software developers mentioned:

- S1P20

  > '...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.'

- S1P7

  > '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.'

**Management**

Although management did not feature hugely in mitigation strategies, Table 4.9 suggests that management need to better plan workloads, promote and provide training for industry software developers to use best practises. For example, some industry software developers mentioned:

- S1P11

  > '...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'

- S1P10

  > '...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 industry software developers are not very surprising. Industry software 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 industry software 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 industry software developers in effective error mitigation is likely to depend on better tools, management and processes.

Overall in response to RQ2 my results suggest that industry software developers predominately rely on trying to improve their own willpower to mitigate errors. Industry software developers use a variety of strategies to retain their focus and concentration in order to reduce the number of errors they make. Industry software developers also use mitigation strategies

| **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 4.6: Themes of Developer Mitigation Strategies

within the development process, often to detect faults (e.g., checklists), tools (e.g., automation) and management (e.g., planning).

## 4.4 Threats To Validity

As with any empirical research our study has several threats to validity. Below I explore the threats to validity as construct validity, internal validity, external validity and repeatability. In addition to these I explore the following threats to qualitative research as described by Maxwell (1992); descriptive validity and interpretation validity.

| **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 4.7: Themes of Processes Mitigation Strategies

| 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 4.8: Themes of Tools Mitigation Strategies

### 4.4.1 Construct Validity

Construct validity assesses our ability to measure an 'object' I intend measuring. The responses given to me by a developer have been processed by the developer, therefore, I 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. Participants' reactions to the researcher presence must also be considered. I 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 I recorded and transcribed the interviews. In addition I investigate only a subset of all human errors i.e. SB errors. Future work will investigate RB and KB errors.

| **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 4.9: Themes of Management Mitigation Strategies

### 4.4.2 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, I used the same interviewer throughout the study. This allowed me 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 industry software 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 Ericsson & Simon (1984) 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 I report.

### 4.4.3 External Validity

External validity assesses the ability to generalise our results. My 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 industry software 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 industry software 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.

### 4.4.4 Repeatability

Repeatability assesses whether if this study were to be repeated by another they would get the same results. I 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. I 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.

### 4.4.5 Descriptive Validity

Descriptive validity assesses whether a researcher has accurately recorded what they heard a participant say and or do. I transcribed all the recorded interviews. I was able to ensure that the transcribed all transcripts in the same way, to reflect what was said in each interview recording by; (a) re-playing each interview and reading what was transcribed to ensure a match, (b) including all stumbles/stutters in speech from both the researchers and participants.

### 4.4.6 Interpretation Validity

Interpretation validity assesses whether a researcher has captured the observation as interpreted by the participant being researched. During each interview I attempted to get each participant to be as descriptive as possible about each scenario so that during analysis, it would be clear and easy to interpret the scenario from the participants point of view. Where it was not clear I asked further questions to the participant until a clear picture had been captured.

## 4.5 Summary

In this chapter I presented preliminary work on the errors industry software developers make and how they mitigate these errors. I discuss the implications of these findings in Chapter 6. This study makes the following contributions:

- To the best of my knowledge this is the first study to provide empirical evidence about what SB human errors industry software developers make and how they currently mitigate errors. Future work is needed to understand the Rule-based (RB) and Knowledge-based (KB) human errors industry software developers make and mitigation for these error types.

- I uncover that industry software developers appear to take personal responsibility for their SB human errors. My results suggest that mitigation was also developer-centric, revolving around improving concentration, awareness and focus.

- *RQ1: What SB errors do industry software developers experience during development?* My results suggest that industry software developer make all eight SB errors. Perceptual confusions (13 mentions)

appear to be the most commonly occurring and double capture slips (4 mentions) being least occurring. Errors were labelled as reason for error, error or consequence. The most common reason for error was complexity of development environment (16 mentions), most common error was concentration (13 mentions) and most common consequence of error was duplications (10 mentions).

- *RQ2: How do industry software developers mitigate the SB errors they experience during development?* My results suggest that industry software developers are aware of methods to mitigate the SB errors they make. Industry software developers were able to suggest the most mitigation strategies for omission following interruptions (25 mentions) while the least amount of mitigation strategies for interference errors (7 mentions). Mitigation strategies were labelled as developer, tool, process and management. The most common developer mitigation strategy was focus (10 mentions), most common tool mitigation strategy was automation (9 mentions), most common process mitigation strategy was checklist (12 mentions) and most common management mitigation strategy was planning (3 mentions).

# Chapter 5

# Can improving industry software developers SA reduce the number of human errors they make? - An Exploratory Study

This chapter explores whether the number of industry software developer human error reduces after undergoing training on Situation Awareness (SA). This study focuses on looking at all human error error types; slips, lapses and mistakes. 10 industry software developers participate in a two week experiment where they log all human errors they make and undertake a training package on SA which employs the OODA loop by Boyd (1987). The OODA loop is critical thinking pathway, in which one follows the cycle of Observe - Orient - Decide - Act.

Section 5.1 presents the research questions I ask in this study. Section 5.2 details the approach taken for the study. Section 5.3 presents the results

for this study. Section 5.4 presents the threats to validity for this study. Section 5.5 presents a summary of this chapter.

## 5.1 Research Questions Addressed In This Study

In this chapter I ask the following research questions;

- **RQ3** What type of human errors do industry software developers make?
- **RQ4** Does the online training package on the OODA loop reduce the number of human errors that industry software developers make?
- **RQ5** Do industry software developers find the online training package easy and useful to use?

## 5.2 Approach

To aid industry software developers reduce the number of human errors they make, I propose introducing a defence mechanism to the Swiss Cheese Model in the form of an online training package targeting software developer SA at the developer layer of the Swiss Cheese Model.

In Chapter 4 I learn that 27 industry software developers do suffer with skill-based human errors. These industry software developers seemed aware of methods that could help them reduce the number of skill-based errors. It was noted that the industry software developers do not know how to implement all mitigation strategies e.g. how to improve their ability to focus on a task. The mitigation strategies reported by the industry software developers in the previous study (Chapter 4) were broken down in to four high level themes, these are; developer, process, tool and management.

Figure 5.1 shows the Swiss Cheese Model. The SB errors shown described by industry software developers in Chapter 4 can be seen as the holes in the

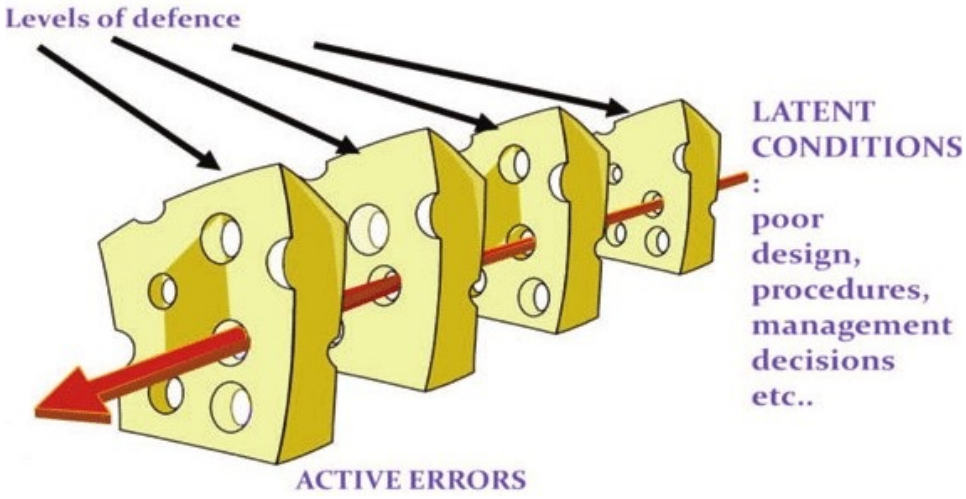



Figure 5.1: James Reasons's Swiss Cheese Model Reason (2008) adapted from Carthey (2013)

layers of cheese, the larger the hole, the more prevalent the error is. The identified themes (developer, process, tool and management) can be seen as the levels of defence. By strengthening a defence layer, you can reduce the size the of the hole in a defence layer and aid in reducing the likelihood of an error making its way through the layers of the Swiss Cheese Model.

I have chosen to focus this body of research on the layer of cheese with the most amount of hole identified within the Swiss Cheese Model. In Chapter 4 this has been identified as being the developer layer of cheese. Having conducted some preliminary background research in the identified holes of cheese, situation awareness started to become a common theme tying many of these holes together. As a result I focus my work towards the use of situation awareness within the designed intervention.

Endsley (1988) 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 industry software developers suggest lost situation awareness. This

is further discussed within Section 6.2.1.

I used qualitative data in the form of human error logs and survey responses, alongside quantitative data in the form of online training results. I implemented a grounded approach to coding the human error logs and survey response data collected. I now detail the OODA loop intervention, study participants and their recruitment, the experimental process and data analysis involved in my study.

### 5.2.1 The OODA Loop Intervention

When designing the intervention to be used to train software developer situation awareness there were multiple factors I had to consider. Some of these factors included ranged from the ease of use and reuse of the intervention, how much time would my industrial collaborators grant me with developers, intervention delivery e.g. games, classroom, virtual learning environment, etc and many more.

There are a number of cognitive training methods that could have been used instead of the OODA Loop, some of these include memory recall games, peripheral vision training, sound identification, person/intruder identification, spot the difference and many more. Some of these training methods are rather specific to combat related scenarios and immediately deemed inappropriate for my setting of training software developers. Other training methods were excluded due to time limitations from my industry collaborators.

The OODA loop is one of the most widely used training methods and is very easy to apply within software development. I discuss the background to the OODA Loop in Section 2.4.1, however a brief reminder of the OODA loop is as follows. The OODA loop is a cognitive training method designed to improve decision-making (Boyd 1987). The four stages of the OODA loop (Figure 5.2) are Observe - Orient - Decide - Act, which form the basis of

improving critical decision making.

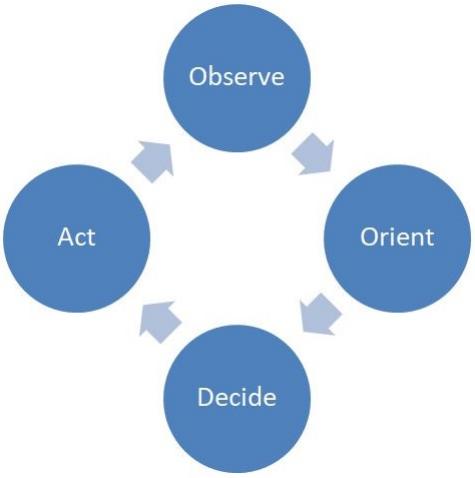



Figure 5.2: OODA Loop (Boyd 1987)

An example of day to day use of the OODA loop is the process of purchasing a meal. First you notice you are hungry, this the observation phase. You proceed to the phase of orientation which could be remembering its still breakfast hour in your local bakery and you can get a sausage roll. Next comes the decision making in which you actively decide to take a break from your current task and get a sausage roll or do nothing about your hunger. Finally comes the action phase in which you actual going to the bakery and purchase a sausage roll.

By using a simple but effective training method I was able to partner this is easy to access training delivery platform i.e. online videos. This was ideal for developers as it meant they could do the training at the beginning of their working days and not need to change flexible working hours to in a classroom based training setting at a specific time. In addition it significantly improved the reach of the training and kept it consistent for all participants as it was the same video and quiz.

The OODA loop encourages the maintenance of situation awareness (SA) by iteratively 'Observing' (level 1 of SA), 'Orienting' (level 2 of SA), and

'Deciding' before 'Acting'. By employing the use a simple loop software developers should be able to better maintain their situation awareness and reduce the number of holes within the developer layer of cheese in the Swiss Cheese Model.

### 5.2.2 Participant & Recruitment

The study was performed over a fifteen week period. During this time 10 industry software developers participated in the study, three of whom dropped out of the study due to withdrawal from the study or lack of engagement. All participants were men aged between 18 and 54. Of these industry software developers 4 had more than ten years of industry experience. These industry software developers were a convenience sample and had been contacted by word-of-mouth, email and face-to-face. Table 5.2 provides a full breakdown of the participants who were recruited for this study. In addition to this I shared details of the study and issued invitations to participate on social media, with posts on LinkedIn and Twitter.

Given this is a separate study, a new call for participants was made on social media. Table 5.1 shows how many views, likes and shares each post gained. When reviewing the figures in the views column it is important to note that this reflects how many people have seen it. This is not an indication of how many people have read the post fully or followed up on details. I do not know have any demographic information about the users that viewed the posts therefore I can not be sure whether they are views from the intended target group i.e. active industry software developers.

| | **Views** | **Like** | **Shares** |
|---|---|---|---|
| **Twitter** | 1790 | 6 | 2 |
| **LinkedIn** | 450 | 2 | 2 |

Table 5.1: Social Media Views For Participant Recruitment

Please see Table 5.2 for demographic data of participants of the experimental study.

| | **Gender** | **Age Range** | **Years of Industry Experience** | **Primary Job Role** | **Primary Industry** | **OSS vs CSS** | **Primary Language** |
|---|---|---|---|---|---|---|---|
| $P_1$ | M | 45-54 | 10+ | Software Developer | Software Development | CSS | Python |
| *$P_2$* | *M* | *45-54* | *10+* | *Technical Architect* | *Humanitarian Aid* | *CSS* | *Java* |
| *$P_3$* | *M* | *25-34* | *<1* | *Junior Developer* | *Software Development* | *OSS* | *Python* |
| *$P_4$* | *M* | *25-34* | *<1* | *Software Developer* | *Employed* | *CSS* | *Python* |
| $P_5$ | M | 25-34 | 4-7 | Software Developer | Humanitarian Aid | CSS | Python |
| $P_6$ | M | 25-34 | 8-10 | Senior Web Developer | Software Developer | CSS | PHP |
| $P_7$ | M | 18-24 | 1-3 | Web Developer | Software Development | CSS | PHP |
| $P_8$ | M | 25-34 | 10+ | Software Developer | Engineering | CSS | C/AL |
| $P_9$ | M | 35-44 | 10+ | Software Manager | Software Developement | CSS | PHP / .NET |
| $P_{10}$ | M | 25-34 | 1-3 | Software Engineer | SaaS Predictive Maintenance | CSS | Scala |

| | Gender | Age Range | Years of Industry Experience | Primary Job Role | Primary Industry | OSS vs CSS | Primary Language |
|---|---|---|---|---|---|---|---|

Table 5.2: Demographic Data - Experiment Participants

Rows in italics indicate that the participate withdrew from the study or was not engaging.

### 5.2.3 Evaluation Method

The method used for this study is a field experiment where participants were required to keep a log of all human errors they encounter over a two week (ten working day) period. The OODA loop intervention (described in Section 5.2.1 was administered half way through the study (the morning of day 6). This intervention takes form as an online training package in which industry software developers use a series of short online videos. In order to confirm participants basic understanding of the OODA loop they were asked to complete a brief quiz.

The exploratory study was broken down in the three key stages, firstly an introduction and week one, secondly the intervention and week two and finally the follow up. Participants were contacted via email at the beginning of each of these stages. Below I describe what participants were made aware of in each of these three stages.

In the first instance to kick start the study (Email One). Industry software developers were provided with the following documents:

1. Introduction To Study (Video): A short video explaining the study. This will be a hyperlink in the email and an optional resource for industry software developers to use. You can watch the same 'Intro To Study' video here: `https://www.dropbox.com/s/gjd9o0almn865sl/`

IntroToStudy.mp4?dl=0

2. Participant Information Sheet: Provided participants basic information about the study and highlights key contacts they may need to speak with during the study. This was attached to the email and is a mandatory resource for industry software developers to read.
3. Consent Sheet: Gains participants consent to participate in the study and ensures they agree and understand the necessary points. This was attached to the email and is a mandatory resource for industry software developers to complete.
4. About Them Questions: Basic demographic questions about the participant and some questions about how they deal with human error. This was attached to the email and is a mandatory resource for industry software developers to complete. The questions asked are available in Appendix C.2 - Supporting Material For Study Two.
5. Record Log: A log sheet to help industry software developers keep track of any human errors that occur during the working day. This was attached to the email and is a mandatory resource for industry software developers to complete.

In the second instance the participants were emailed to kick start the online training package (Email Two). Participants were provided with the following links:

1. Intro to situation awareness. You can watch the same 'Intro To Situation Awareness' video here: https://www.dropbox.com/s/j8b9aahy331ff8j/TrainingOneIntroToSA.mp4?dl=0 Screenshots of the slides used in this video can be viewed in Appendix C.4.1 - Supporting Material For Study Two.
2. Intro to OODA loop. You can watch the same 'Intro To OODA Loop' video here: https://www.dropbox.com/s/xz2pjh1t895gt1v/

`TrainingTwoIntroToOODA.mp4?dl=0` Screenshots of the slides used in this video can be viewed in Appendix C.4.2 - Supporting Material For Study Two.

3. Applied OODA loop in software development. You can watch the same 'Intro To Applied OODA LOOP' video here: `https://www.dropbox.com/s/kf67z508w3dv9n5/TrainingThreeOODAExamples.mp4?dl=0` Screenshots of the slides used in this video can be viewed in Appendix C.4.3 - Supporting Material For Study Two.

4. Quiz. his quiz will take form as an online questionnaire and results were emailed to industry software developers. The questions asked are in the Quiz in Appendix C.5 - Supporting Material For Study Two.

The final email requested that participants send their completed log sheets back to me and complete the follow up questionnaire. The questions asked in the follow up questionnaire can be viewed in Appendix C.6 - Supporting Material For Study Two.

In order to help my participants I sent them a daily message via Skype/Email (depending on the company setup and the level of access I can gain). This message served as a reminder to complete the log sheet every day and aimed to engage industry software developers by getting them to reply saying how many errors they logged that working day.

### 5.2.4 Data Analysis

Throughout the experimental study three key pieces of data were recorded; (1) human error logs, (2) results of the training package and (3) participant views on the training package. Below I explore what processing and analysis was performed each piece of collected data.

Firstly I look at the data collected in the human error logs. This data was coded using both inductive and deductive means to determine the human

| Participant Number | Initial Agreeability Between My Supervisor & I | Agreeability Between My Supervisor & I After Discussion |
|---|---|---|
| **$P_1$** | 57.14% | 100% |
| **$P_5$** | 50.00% | 100% |
| **$P_6$** | 83.33% | 100% |
| **$P_7$** | 57.89% | 100% |
| **$P_8$** | 57.14% | 100% |
| **$P_9$** | 43.75% | 100% |
| **$P_{10}$** | 80.00% | 100% |
| **Average** | 61.32% | 100% |

Table 5.3: Agreeability of Coding

error category and its theme. The human error categories were predefined therefore deductive coding was applied here. These were slip, lapse and mistakes and coded against the definitions provided by Reason (1990), these can be found in Appendix C.7. Coding of errors described by the developers in each log, was done in an inductive manner, in that the themes were derived from the data. The initial agreeability shown in Table 5.3 fluctuates between the two researchers (my supervisor and I), with high agreement for some participants (e.g. P6) and low agreement for others (e.g. P9). During discussion of disagreements between my supervisor and I, (recommended by Kitchenham et al. (2012)) it was clear that lack of contextual detail around error reports from some participants was the reason for these disagreements. Extensive discussion took place between the two raters (my supervisor and I) which resulted in some errors being multi-classified. Multi classifications were mainly across slips and lapses (both of which are execution errors so have some commonality) as it was difficult to retrospectively know whether an error was the result of a memory issue if this was not explicitly mentioned

by the developer. After this process of discussion, 100% agreement was achieved. I identified seven themes across the human errors which are:

- **Internal communication** Poor internal communication e.g. incomplete documentation.
- **External communication** Poor external communication e.g. failing to obtain full error details from end user.
- **Code structure/complexity** Poor code structure and or increased code complexity.
- **Complexity of development environment** Having many things running in the development environment at anyone time
- **Ordering/sequencing tasks** Executing a series of tasks in the incorrect order.
- **Syntax issues** Use the wrong syntax / syntax errors in newly written code.
- **Special cases** Unique errors which do not fit the other themes e.g. UI/UX design / functionality issues

The second piece of data was the results from the training package. This did not need processing as the results were binary i.e. pass / fail on each question and the results tallied up.

The final piece of data was the views of participants on the training. These were coded via sentiment with four potential codes: 1 = Positive, 0 = Neutral, -1 = Negative & - = No comment To Code. Many of the questions asks already had codes associated with them e.g. yes / no questions. As result I coded all responses and my supervisor reviewed the codes assigned. My supervisor and I had a 100% agreeability on these.

## 5.3 Results

The results from my small-scale snapshot study confirms that industry software developers make human errors manifesting as slips, lapses and mistakes. Training industry software developers to maintain situation awareness using the OODA Loop seems to lead to decreased developer errors. I will now present the results broken down by each research question posed at the beginning of the chapter.

### 5.3.1 RQ3 - Do industry software developers make more slips/lapses compared to mistakes?

Participants record all human errors they make during development work across a ten day period in a log sheet. Table 5.4, 5.5 and 5.6 shows each error for each developer classified two ways; first, errors in each of the seven themes, second, each error classified as either a slip, lapse or mistake. A small number of errors are multi-classified because they cut across human error classifications.

Table 5.4, 5.5 and 5.6 suggests that the distribution of human error types (i.e. slips, lapses and mistakes) varies between industry software developers with some reporting many more mistakes than others.

| Participant Number | Team Communication | | Code Structure / Complexity | Complexity of Development Environment | Ordering / Sequencing Tasks | Syntax Issues | Special Cases |
|---|---|---|---|---|---|---|---|
| | Internal | External | | | | | |
| **P1** | | M1 | L/M 1 | S 1 | S/L/M 1<br>S 1<br>L 1 | S/L 1 | |
| **P5** | | | | S 1<br>S 2<br>S 3 | L 1<br>S/L/M 1 | | S/L 1 |
| **P6** | S/L/M 1 | M 1<br>M 2<br>M 3 | S/L 1<br>S/L 2<br>S/L 3 | S/L 1 | | S/L 1<br>S/L 2<br>S/L 3<br>S/L 4<br>S/L 5<br>S/L 6 | |

Day 1 to 5: White Background & Day 6 to 10: Grey Background

Key: S = Slip, L = Lapse, M = Mistake, S/L = Slip/Lapse, L/M = Lapse/Mistake & S/L/M = Slip/Lapse/Mistake

Note: The number after the key is a counter for each participant for the type of error

Note: Participants 2, 3, 4 withdrew due to a change in workload or non - engagement

Table 5.4: High Level Themes (1/3)

| Participant Number | Team Communication | | Code Structure / Complexity | Complexity of Development Environment | Ordering / Sequencing Tasks | Syntax Issues | Special Cases |
|---|---|---|---|---|---|---|---|
| | Internal | External | | | | | |
| **P7** | M 1 | M 1<br>S/L/M 1<br>S/L/M 2 | S/L 1 | S/L 1<br>S/L/M 1<br>M 1 | | S/L 1<br>S/L 2<br>S/L 3<br>M 1<br>L 1 | S/L 1 |
| **P8** | M 1 | | S/L 1<br>S/L 2<br>S/L 3<br>S/M 1<br>S/L 4 | | | | |
| **P9** | M 1 | M 1<br>M 2 | S/L/M 1<br>S/L 2 | | | S/L 1<br>S/L 2 | M 1 |

Day 1 to 5: White Background & Day 6 to 10: Grey Background

Key: S = Slip, L = Lapse, M = Mistake, S/L = Slip/Lapse, L/M = Lapse/Mistake & S/L/M = Slip/Lapse/Mistake

Note: The number after the key is a counter for each participant for the type of error

Note: Participants 2, 3, 4 withdrew due to a change in workload or non - engagement

Table 5.5: High Level Themes (2/3)

| Participant Number | Team Communication | | Code Structure / Complexity | Complexity of Development Environment | Ordering / Sequencing Tasks | Syntax Issues | Special Cases |
|---|---|---|---|---|---|---|---|
| | Internal | External | | | | | |
| **P10** | | M 1 | L/M 1 | L 1 | S 1 | S 1 | |
| | | | | M 1 | L 1 | L 1 | |
| | | | | L 2 | | S 2 | |

Day 1 to 5: White Background & Day 6 to 10: Grey Background

Key: S = Slip, L = Lapse, M = Mistake, S/L = Slip/Lapse, L/M = Lapse/Mistake & S/L/M = Slip/Lapse/Mistake

Note: The number after the key is a counter for each participant for the type of error

Note: Participants 2, 3, 4 withdrew due to a change in workload or non - engagement

Table 5.6: High Level Themes (3/3)

Table 5.4, 5.5 and 5.6 also shows variation in the error themes to which industry software developers seem prone. For example, Participant 6 makes the most syntax errors, a number of which appear to be related to JavaScript and specifically the use of the keyword 'this'. Whereas Participant 8 seems to make more code structure / complexity errors. Participant 5 is the only developer who does not record any communication related errors. My analysis does not suggest experience or demographics explains why these industry software developers seem prone to these specific errors but it might be that the particular work tasks during the snapshot influenced the error themes (See Table 5.2 for more details). Future work is needed to investigate the relationship between errors and development context.

Table 5.4, 5.5 and 5.6 suggests that most communication based errors are mistakes rather than slips or lapses. Mistakes are usually more substantial errors than lapses or slips and can be more complex to correct. This confirms the importance of strong communication during development activities. Table 5.4, 5.5 and 5.6 also shows that syntax errors comprise mostly of slips or lapses. This suggests that industry software developers generally know syntax but make minor errors despite this knowledge. Most industry software developers reported errors related to the complexity of the development environment. For example Participant 10 says that they *'Forgot to increase the version of an updated dependency.'* This is because Participant 10 says they were *'Juggling three different tasks all at the same time. Performance research on one strand, bug fixing on two separate issues. Each with their own programming languages! (Python, Java and Scala)'.* Table 5.4, 5.5 and 5.6 shows only Participants 8 and 9 do not report such complex development environment errors. More work is needed to uncover any contextual factors explaining this variation.

### 5.3.2 RQ4 - Does the online training package on the OODA loop reduce the number of human errors that industry software developers make?

Participants undertook an online training package comprised of three videos followed by a quiz. Industry software developers engaged well with my OODA loop training. All but one of whom attained scores of eight or nine in the nine question quiz that concluded the training package.

Table 5.7 shows how many errors were self-recorded by industry software developers before and after OODA loop training. Although the numbers of errors in the snapshot are small, Table 5.7 shows an encouraging error reduction after training. In all cases there is either a reduction in errors (four industry software developers) or no change in the number of errors (three industry software developers) after the OODA loop training. I performed a paired T-test, which shows a significant difference of 0.0414 between the number of errors being made before and after the OODA loop training.

| | **Total Human Errors** | **Before Training Errors: D1- D5** | **After Training Errors: D6 - D10** | **Reduction Rate** |
|---|---|---|---|---|
| P1 | 7 | 6 | 1 | 83% |
| P5 | 6 | 3 | 3 | 0% |
| P6 | 14 | 9 | 5 | 44% |
| P7 | 14 | 8 | 6 | 25% |
| P8 | 6 | 5 | 1 | 80% |
| P9 | 8 | 4 | 4 | 0% |
| P10 | 10 | 5 | 5 | 0% |

Table 5.7: Numbers of logged Human Errors

I looked in more detail at the types of errors before and after OODA loop training and found that error reductions after training were predominately

in execution errors. For example Participant 6 makes 18 execution errors before training which reduces to four execution errors after training. This reduction in execution errors suggests that using the OODA loop during development helps industry software developers retain situation awareness of their code and maintain concentration sufficiently to reduce the slips and lapses that they usually make. A larger scale study is needed to establish whether this finding holds more generally for industry software developers and whether the effect lasts over time.

| | **Stage One Questions** | | | | | **Stage Two Questions** | | | | **S1** | **S2** | **Average** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Q1** | **Q2** | **Q3** | **Q4** | **Q5** | **Q1** | **Q2** | **Q3** | **Q4** | % | % | **Score** |
| **$P_1$** | Y | Y | Y | Y | Y | Y | Y | Y | N | 100% | 75% | 89% |
| **$P_5$** | Y | Y | Y | Y | Y | Y | Y | Y | Y | 100% | 100% | 100% |
| **$P_6$** | Y | Y | Y | N | N | Y | Y | Y | N | 60% | 75% | 67% |
| **$P_7$** | Y | Y | Y | Y | Y | Y | Y | Y | Y | 100% | 100% | 100% |
| **$P_8$** | Y | Y | Y | Y | Y | Y | Y | Y | N | 100% | 75% | 89% |
| **$P_9$** | Y | Y | Y | Y | Y | Y | Y | Y | N | 100% | 75% | 89% |
| **$P_{10}$** | Y | Y | Y | Y | Y | Y | Y | Y | Y | 100% | 100% | 100% |

Table 5.8: Participants Results from Training

### 5.3.3 RQ5 - Do industry software developers find the online training package easy and useful to use?

I asked industry software developers for their thoughts about the training. The sentiment of all responses can be viewed in Table 5.9. The sentiment is positive indicating that participants enjoyed participating in the study and found the online training tool easy to use. Participants said that they learned about the OODA loop and how to apply it in software development for the first time. Participant 7 said *'Yes, the idea of OODA was helpful when dealing with developing problems.'* All industry software developers found

the content actionable, Participant 2 saying *'Fully actionable especially in our work.'* Participant 9 says the OODA loop is *'Very helpful and is a need in our daily work in software engineering.'* While the majority of the participants found the OODA loop led to an improvement in their work, Participants 1 & 10 did not. All except Participant 1 report that they will continue to use the OODA loop in the their work. Participant 10 says that they will continue to use the OODA loop in their development, despite not finding any direct improvement.

| | **About OODA Training** | | | | | **About The Study** | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Q1 | Q2 | Q3 | Q4 | Q5 | Q1 | Q2 | Q3 | Q4 | Q5 |
| **$P_1$** | 1 | 0 | -1 | -1 | 0 | 1 | 1 | 1 | 0 | 0 |
| **$P_5$** | 1 | 1 | 1 | 1 | - | 1 | 0 | 1 | 1 | - |
| **$P_6$** | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | - |
| **$P_7$** | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| **$P_8$** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| **$P_9$** | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| **$P_{10}$** | 1 | 0 | -1 | 1 | 1 | 1 | -1 | 1 | 1 | 0 |

Table 5.9: Follow Up Questionnaire Coded By Sentiment

Key: 1 = Positive, 0 = Neutral, -1 = Negative & - = No comment To Code

## 5.4 Threats to Validity

As with any empirical research my study has several threats to validity. Below I explore the threats to validity as construct validity in Section 5.4.1, internal validity in Section 5.4.2, external validity in Section 5.4.3 and repeatability in Section 5.4.4.

### 5.4.1 Construct Validity

Construct validity assesses our ability to measure an 'object' I intend measuring. I was collating cognitive information from individual industry software developers in each human error log. Each participant provide me with a log sheet containing human errors which they have processed already, therefore, I can not be certain the data is accurate. Participants may falsify their responses to influence the impression of themselves that they are presenting. Accuracy is a common threat to validity in all interview and questionnaire studies. Given the duration of the study, industry software developers may forget to engage with the study and forget to log human errors day to day. I attempt to combat this by sending daily reminders and pro actively engaging with industry software developers by asking how many errors they logged per day.

### 5.4.2 Internal Validity

Internal validity assesses whether all elements of the study have been designed and executed correctly. I mitigate bias by asking the participants to complete the same activities. I use the same forms and questions when participates are asked to provide me with any responses. This allowed me to ensure all individual data captures had been conducted in the same way and the approach to asking questions were uniform throughout. During the analysis phase my supervisor and I independently coded all human error logs, upon comparison my supervisor and I came to the same results or had extensive discussion until mutual agreement was reached.

### 5.4.3 External Validity

External validity assesses the ability to generalise our results. Although the sample size of industry software developers is relatively small, the qualitative data obtained from the human error logs provided by the participants has

been detailed. Over half of my respondents have been practicing in industry for over 8 years. The sample size is not generalisable to the wider population however the responses somewhat represent that of 3 different companies. With such a small sample size, threats such as winner's curse (Ioannidis 2008, Marino 2018, Palmer & Pe'er 2017) need to be considered. The large effect size I have discovered could be significantly inflated. Future works should look to expand upon this exploratory study and include a larger sample set so that threats such as winner's curse are not an issue.

### 5.4.4 Repeatability

Repeatability assesses whether if this study were to be repeated by another they would get the same results. By providing an in depth explanation into how participants have been recruited, experiments have been performed and analysis has been conducted I hope that the study is replicable. Recruiting a similar group of participants could prove difficult as each researcher's access to potential participants varies. The demographics of the group recruited is important, as the type of human errors may vary based on the developer's experience, primary industry, technology use etc.

## 5.5 Summary

In this chapter I presented my preliminary study on reducing human error during development by providing industry software developers with cognitive training designed to improve their situation awareness. I discuss the implications of these findings in Chapter 6. My study makes the following contributions:

- To the best of my knowledge this is the first study to provide empirical evidence showing how the number of human errors made by industry software developers reduces when industry software develop-

ers are provided with training to improve their situation awareness. More work is needed to understand whether our results hold beyond this small scale study.

- *RQ3 - What type of human errors do industry software developers make?* My results show that industry software developers do make all three types of human errors (slips, lapses and mistakes) in their development work. My results suggest that slips and lapses occur more commonly than mistakes. Slips and lapses are likely to result in smaller defects which should be easier to mitigate.

- *RQ4 - Does the online training package on the OODA loop reduce the number of human errors that industry software developers make?* My results show that cognitive training using the OODA loop leads to a decrease in the number of human errors industry software developers make. I found that industry software developers particularly reduced the number of execution errors they made following OODA loop training. More work is needed to understand if this finding is generalisable and if the effect lasts over time.

- *RQ5 - Do industry software developers find the online training package easy and useful to use?* My results show encouraging signs that most of the industry software developers in my study were enthusiastic about the training they received in how to maintain their situation awareness. Most industry software developers found the training easy and useful in a topic they had no previous knowledge. The majority of the industry software developers said that they will continue to use the OODA loop in their work.I intend to explore further the use of focused training packages to educate and support industry software developers in their work.

# Chapter 6

# Discussion

This chapter discusses the findings of two studies which investigate human error within software development as detailed within Chapter 4 & 5. The chapter is presented by addressing each research question individually. Section 6.1 discusses SB errors reported in Chapter 4 (**RQ1**). Section 6.2 discusses mitigation strategies reported in Chapter 4 (**RQ2**). Section 6.3 discusses the type of human errors reported in Chapters 4 and 5 (**RQ3**). Section 6.4 discusses our findings on training developer SA in Chapter 5 (**RQ4**). Section 6.5 discusses developer experience of using the training tool in Chapter 5 (**RQ5**).

*Note:* This chapter discusses the results of both studies as a result participant numbers are prefixed with S1 or S2 to identify which study the participant belongs to, for example S1P2 relates to Participant 2 from Study 1.

## 6.1 RQ1: What SB errors do industry software developers make during development?

The results of my study suggest that all industry software developers make all 8 SB errors. SB errors indeed, slips/lapses or in other words execution

based errors. Many development activities fit execution based error well, therefore it is unsurprising that all 8 SB errors are mentioned by participants in my study. Double capture slips are mentioned the least and perceptual confusions are mentioned the most. It is important to better understand whether these findings are generalisable or not, as researchers will be able to better target the correct subset of problematic errors. This will allow researchers to better target efforts at either SB errors as a whole or focus more specifically at commonly occurring errors.

There are varying cognitive loads on software engineers, these can manifest in various ways, examples of these can include e.g. environmental stressors such as work place distractions, personal stressors such as family issues, health issues such as poor mental health etc. Taking these into account I am surprised that more participants in my studies did not cite more instances of trivial human errors e.g. omitting a semi colon. Trivial errors such as omitting a semi colon are safeguarded against by the compiler. Given the simplicity of these trivial errors participants may have felt these were superficial and not mentioned them. Further work should investigate what stressors / loads on industry software developers are measured.

In Table 4.3 I present developer error themes which were elicited from the interviews from study one. The themes are coded as reason, error or consequence. Further work is required to help industry software developers to identify the reason for the error and the consequence of the error. This would better allow industry software developers to employ appropriate methods to safeguard against their own frequently occurring errors.

In RQ2 I discuss the numerous methods that industry software developers employ to safeguard against SB errors.

## 6.2 RQ2: How do industry software developers mitigate the SB errors they experience during development?

My results suggest that developer-based mitigation strategies are the most frequently reported ways to reduce human errors. I find this interesting as I had believed that industry software developers may have leaned more towards using tool automation or reliance on process based mitigation strategies. Industry software developers seem to think that they individually should stop making errors that get through to production. Other disciplines, for example, health Abimanyi-Ochom et al. (2019), 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., industry software developers saying that they need to concentrate more). Many of the process, tool and management strategies could be described as 'external' to a industry software developer, and mostly focused on detecting consequential physical code faults. These external strategies are likely to aid industry software developers in implementing internal strategies to prevent the error. Figure 6.2 shows how 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 I suggest that the commonly mentioned mitigation strategies of using tools (e.g., automation) and processes (e.g., checklists) can support industry software developers to implement internal mitigation strategies (e.g., staying focused).

My results resonate with the Reason (2000) Swiss Cheese model of accident causation. The Swiss Cheese model (shown in Fig. 6.1) is an approach to building effective organisational defences against failure. The aim of the

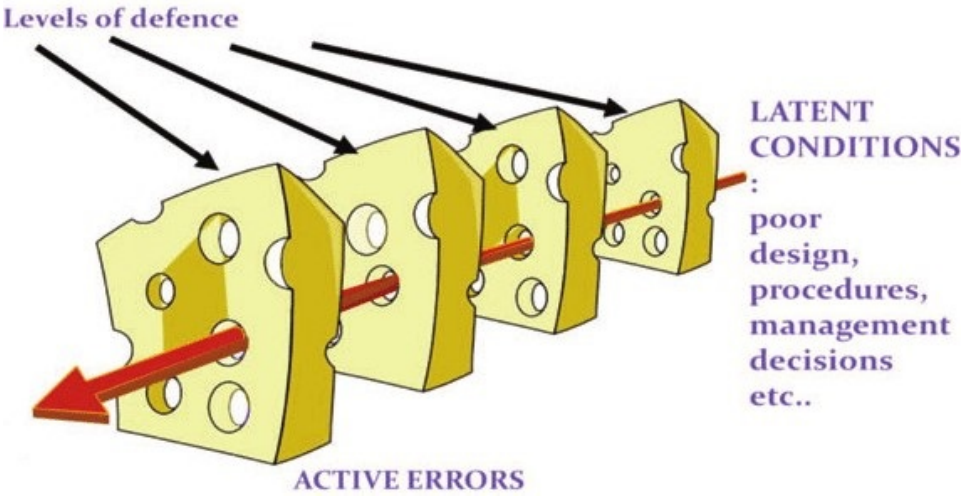



Figure 6.1: James Reasons's Swiss Cheese Model Reason (2008) adapted from Carthey (2013)

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[1]). Taking such an approach to error mitigation 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.

Few of the errors reported by industry software 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 industry software developers are also previously known approaches. For example, using testing, pull-requests and well

[1]https://www.oilfieldtechnology.com/special-reports/23042015/rallying-against-risk/

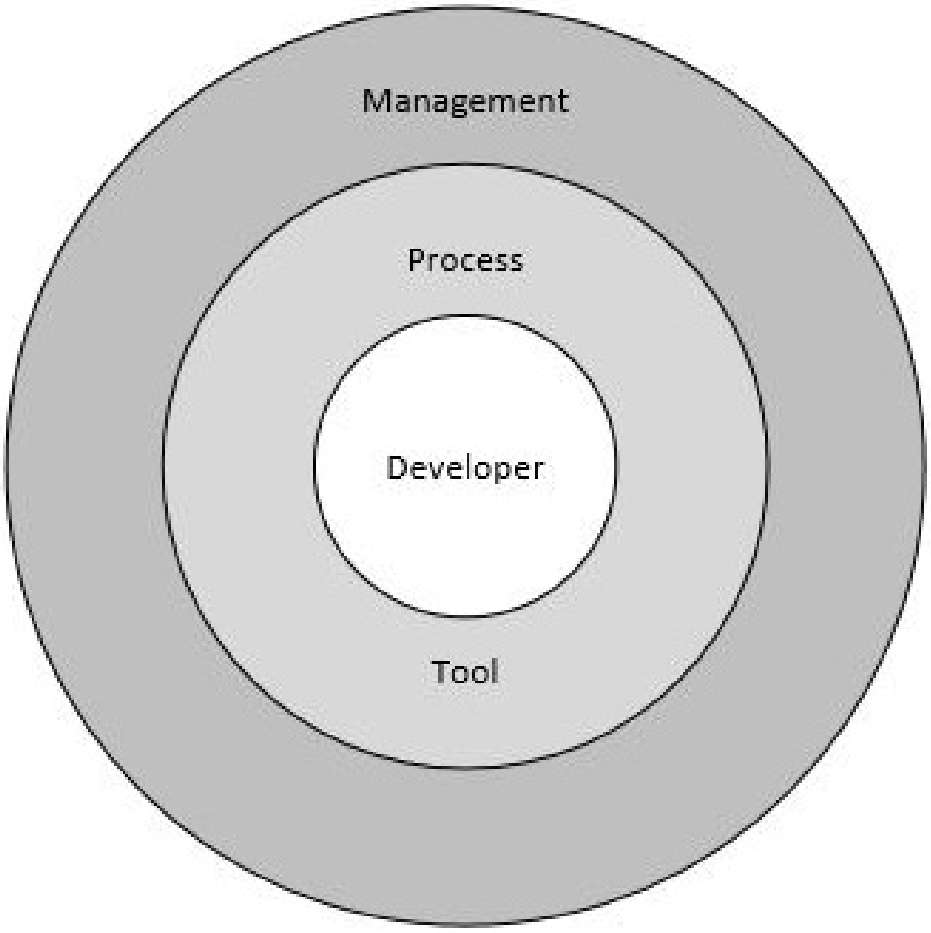



Figure 6.2: Mitigation Strategies

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.

### 6.2.1 Improving Situation Awareness

My results suggest that situation awareness is a problem for industry software developers. Endsley (1988) 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 industry software developers suggest lost situation awareness. For example, S1P12 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. Procida (2017) 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 (Petersen et al. 2019), medicine (Wright et al. 2004), transportation (Wickens 2002) and cyber security (Ioannou et al. 2019).

In software development coding on 'autopilot' can lead to the loss of situation awareness which can lead to industry software developers going down rabbit holes. The unplanned refactoring of code is usually such a rabbit hole. Industry software 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. I recommend that industry software developers conduct refactoring tasks as part of planned work rather than embedded in other tasks.

S1P9 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.'

My results suggest that industry software 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 industry software support developers in recognising when situation awareness is being lost need to be developed, perhaps in the form of training for industry software developers as such training in situational awareness is common in health and medical practice. Brennan et al. (2016) report that improving surgeons' ability to manage their awareness levels is an essential requirement to reducing medical error.

### 6.2.2 Improving Cognitive Skills

My results suggest that cognitive skills such as remaining focused, remembering, maintaining self discipline and attention all affect industry software developers. These cognitive skills also impact a developer's ability to maintain situational awareness. Industry software developers indicated that these were skills they wanted to improve. Related to this, our results also show

that many industry software developers said they need to concentrate better, pay more attention, focus more. S1P24 says

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

S1P16 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 industry software developers are affected by cognitive issues. For example, S1P16 says

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

Industry software developers also seem to take personal responsibility for their ability to deploy effective cognitive skills. The underlining assumption being that industry software 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) Boyd (1987) is often a part of military training and may prove worthwhile in helping industry software developers improve their cognitive skills.

### 6.2.3 Using Checklists

My results suggest that industry software 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. (2011) report that checklists provide an alternative to reliance on intuition and memory in clinical problem solving. For example, S1P21 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 industry software 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 industry software developers I 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 (Clay-Williams & Colligan 2015). Checklists have also been used in software development, for example, in risk management and extensively in software inspection Brykczynski (1999). Given the value of checklists that industry software developers I 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.

### 6.2.4 Tool use

My results suggest that using software tools can provide industry software developers with greater support. Tool related mitigation strategies were ranked second out of the four high level themes by the industry software developers I interviewed. Table 4.8 shows the specific tools and types of tools which industry software developers mentioned they used to mitigate errors. For example:

S1P9 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.

S1P18 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 industry software developers.

Some industry software 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, S1P7 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 industry software developers. Following the lead of the aviation industry, standardising the interaction of tools with industry software developers could help across multiple platforms and languages.

### 6.2.5 Faster feedback loops

Industry software developers mentioned process related mitigation strategies (see Table 4.7) 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 industry software developers can get feedback on the consequences of their error, the quicker they can make improvements.

Pair programming is a well established approach to industry software developers getting instant feedback on their code Williams et al. (2003), Tomayko (2002). S1P20 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 industry software 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. S1P9 says

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

The value of code review Sadowski et al. (2018) is reiterated by S1P1:

> '...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 industry software developers, but where they are, I 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 my results suggest that industry software developers value mechanisms where code is checked for defects, which have occurred as a consequence of their errors, and feedback quickly provided to them.

### 6.2.6 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. (2012) 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 me as I expected tiredness to be more problematic to software industry software developers. S1P11 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 industry software developers may recognise that they are fatigued but can deploy their own mitigation strategies to combat tiredness.

## 6.3 RQ3: What type of human errors do industry software developers make?

My results suggest that industry software developers make human errors across the three main types of human error i.e. slips, lapses and mistakes. The distribution of the human error types varied between industry software developers with some reporting many more instances of mistakes than others. While I suspect violations do occur within software development, I believe these to be rare. Unsurprisingly none of our participants self-report any violation human errors. Future work could employ use of ethnographic research to determine whether violations do occur, what form these take and why industry software developers make violation errors.

Table 5.4, 5.5 and 5.6 shows variation in the errors that different industry

software developers seem prone to during the two week snapshot of this study. For example Table 5.4, 5.5 and 5.6 suggests that S2P6 makes the most human errors related to syntax. Whereas S2P8 seems to make more code structure / complexity related errors. S2P5 is the only developer who does not record any communication related human errors. In relation to these participants my analysis (See Table 5.2) does not suggest anything about their experience or demographics which could explain why they seem prone to these specific errors. Other factors related to the project or development environment need further examination to identify any relationship with an over-representation of error-types.

Table 5.4, 5.5 and 5.6 also shows the type of human errors (slips, lapse, mistake) that occurred within each theme. Table 5.4, 5.5 and 5.6 suggests that all communication based human errors are mistakes rather than slips or lapses. Mistakes are usually more substantial errors than lapses or slips and can be more complex to correct. This confirms the importance of strong communication mechanisms during development activities. Table 5.4, 5.5 and 5.6 also shows that syntax errors comprise entirely of slips or lapses. This suggests that industry software developers generally know syntax but make minor errors despite this knowledge. Most industry software developers reported errors related to the complexity of the development environment. Table 5.4, 5.5 and 5.6 shows only S2P8 and S2P9 do not report such errors, for example S2P10 tells he they 'Forgot to increase version of updated dependency.' This is because S2P10 was 'Juggling three different tasks all at the same time. Performance research on one strand, bug fixing on two separate issues. Each with their own programming languages! (Python, Java and Scala)'.

When looking specifically at the seven identified themes within Chapter 5, we can immediately a diverse range of human errors. We know that planning based errors are mistakes while execution based errors are slips and

lapses. It is unsurprising to see errors from both execution and planning. The planning based errors come from the two communication related themes, while the execution related errors come from the other themes.

I find it note worthy that the majority of self reported errors described by the participants were execution errors i.e. slips and lapses. Typically planning errors i.e. mistakes were noted as communication errors with external stake holders. Future work should investigate whether execution errors are more prevalent than planning errors.

## 6.4 RQ4: Does the online training package on the OODA loop reduce the number of human errors that industry software developers make?

To check how well participants engaged with the on-line training, participants completed an after training quiz with nine questions. The majority of participants performed well in this quiz with all but one participant attaining scores of eight or nine. Only S2P6 recorded a lower score of six (A full breakdown of participant performance can be found in the Table 5.8.). Overall the quiz results suggest that participants engaged well with the OODA loop training.

Table 5.7 shows how many errors were self-recorded by industry software developers broken down into before and after OODA loop training. Although the numbers of errors across the snapshot are small Table 5.7 shows an encouraging error reduction rate after training. In all cases there is either a reduction in errors (four industry software developers) or no change in the number of errors (three industry software developers) after the OODA loop training.

I looked in more detail at the types of errors before and after OODA loop training (Full details of this analysis can be found in Table 5.7.) and

found that error reductions after training were predominately in execution errors. For example S2P6 makes 18 execution errors before training which reduces to four execution errors after training. This reduction in execution errors suggests that using the OODA loop during development helps industry software developers retain situation awareness of their code and maintain concentration sufficiently to reduce the slips and lapses that they usually make. A larger scale study is needed to establish whether this finding holds more generally for industry software developers.

The OODA focuses on improving level 1 and level 2 SA i.e. observing and orienteering. It lacks in level 3 SA i.e. prediction. This could explain why there is a more noticeable reduction on execution errors post completion of the online training package. A large study would help to confirm whether the findings are generalisable and if there is a noticeable reduction in the type of errors made. This larger study needs to build upon this study, therefore should use software developers practising within industry. The sample should be diverse and not come from a specific underlying population or sector. Given that the type of human errors may reflect differently based on certain demographic factors e.g. a junior developer in there first fews months could be more likely to make more execution based errors as they start to build patterns for repetitive tasks. In Section 6.2 I discuss a variety of mitigation strategies as highlighted by participants in my study. The OODA loop package helps to improve SA and cognitive skills of the industry software developers who complete the online training package. This helps to strength the skill set they have to guard against human errors.

Some interventions can be costly (financially and cognitively) and hard to implement or maintain within an existing set of guidelines for an organisation. The online training package that I have developed is quick to complete and reduces the number of human errors software developer makes. The speed in which it takes to complete and gain direct benefits from the train-

ing package provides initial motivation for management to get their industry software developers to complete the training package.

Participants in my study showed no reduction in terms of the number of errors they made before and after completing the online training package. There are no obvious indications as to why this could be. Looking at the data the participants are experienced industry software developers who performed well on the training quiz. Further work is required to understand the impacts of other factors on developers and how they may impact the number and type of errors made. Such further work must be carefully designed, as it would not be ethical and could be intrusive to collate certain data points about a industry software developers state e.g. mental health conditions, sleep, personal/emotional stressors, etc.

## 6.5 RQ5: Do industry software developers find the online training package easy and useful to use?

I asked our participants for their thoughts about the OODA training (Table 5.9 shows the coded results of participant feedback.). The sentiment of all responses are positive which indicates that participants enjoyed participating in the study and found the online training tool easy to use. Participants said that they learned about the OODA loop and how to apply it in software engineering for the first time during the online training. S2P7 said

> 'Yes, the idea of OODA was helpful when dealing with developing problems.'

All of our participants found the content actionable, S2P2 saying

> 'Fully actionable especially in our work.'

S2P9 says the OODA loop is

> 'Very helpful and is a need in our daily work in software engineering.'

While the majority of the participants found the OODA loop led to an improvement in their work, S2P1 & S2P10 did not. All except S2P1 report that they will continue to use the OODA loop in the their work.

S2P10 says that they

> 'will continue to use the OODA loop in their development, despite not finding any direct improvement'.

S2P10's comment suggests that they found some benefit in using the OODA loop in their work. Further work is required to understand whether other industry software developers find the OODA loop beneficial in their development work for reasons other than to reduce human errors.

Participants told me that the online training package was easy and useful to use. I should have followed up with the study participants as to why they found the online training package easy and useful to use. I assume this is partly attributed to using short structured videos, followed by a quiz to test a learners retention. Methods like these are used in common major open online courses e.g. Coursera.

In RQ4 I learn that human errors do decrease as a result of industry software developers completing the OODA loop training package. I can partly attribute this success to the ease and usefulness of the training package. This is to say, if the training package had not been developed using good practices, users may not have attained such high scores in their quizzes and may not have retained / applied the concept of the OODA loop to their work successfully.

## 6.6 Summary

I show that SB errors do occur in industry software development. Many industry software developers are aware of these and know some means of mitigation. industry software developers appear to hold themselves accountable for their own errors instead e.g. i need to concentrate better instead of blaming others e.g. the workplace is too loud or i have too many tasks.

I show that the online training package I have developed is able to reduce the number of human errors industry software developers make. This is a simple, quick and easy means of aiding many industry software developers in a short space of time. The nature of the online training package means industry software developers can revisit the resource at anytime if they feel the need to refresh their memory.

# Chapter 7

# Conclusion

This doctoral research investigated the use of Human Error Theory within software development. I have provided an insight to the Human Error Theory literature, relevant fields in which Human Error Theory has been applied in e.g. medicine and the preliminary work that has been conducted within software engineering in particular software requirements. I have investigated what SB human errors industry software developers make and how they mitigate these. I show that maintaining SA is an issue to industry software developers. I train industry software developers SA using an online training tool on the OODA Loop to reduce the number of human errors that industry software developers make.

Section 7.1 highlights the steps I undertaken to achieve the aims and objectives of my doctoral research. Section 7.2 details the research contributions my doctoral research makes. Section 7.3 highlights the key research limitations of my doctoral research. Section 7.4 presents future in software development using Human Error Theory based on my doctoral research.

## 7.1 Research Aims & Objectives

The aim of this doctoral research was to: *Deliver a training package for industry software developers aimed at reducing the number of human errors industry software developers make while working on development tasks.* This thesis delivers an online training tool which reduces the number of human errors industry software developers make. This is based on a two week experiment using ten industry software developers.

To allow for successful completion of the aim, I set out the following objectives:

- **Obj1** *To understand what SB human errors occur during software development.* I ask industry software developers to self report on human errors in two studies. In the first I interview industry software developers about SB human errors. In the second I get industry software developers to report on any human error that occurs within a two week window. The response across both studies show that SA is an area of significant struggle to industry software developers.

- **Obj2** *To develop an understanding of how industry software developers currently try to mitigate against SB human errors.* I ask developers to self report on mitigation strategies they employ to their human errors in the first study. Industry software developers listed a variety of strategies that could be classified as developer, tool, process or management focused at a high level. It became apparent these industry software developers viewed themselves as the focal point of many mitigation strategies. This is not surprising given how they are struggling with maintaining SA.

- **Obj3** *To establish whether industry software developers make more slips/lapses vs mistakes.* By asking industry software developers to report on all human errors that they make during software development

activities across a two week period I can determine which type of error is more frequent. I learn that SB or slips/lapses are more common within software development.

- **Obj4** *To deliver an educational tool which aids industry software developers to mitigate against the most frequent form of human error.* I develop a four part online training package which educates industry software developers on SA and the OODA Loop. Industry software developers self report human errors made across a two week period where industry software developers complete the training on day 6. I see a clear drop in the number of human error made after using the tool.

## 7.2 Research Contributions

My doctoral research makes the following contributions to knowledge;

- **Cont1** New understanding about typical human errors made by industry software developers. While all forms of human error occur, slips and lapses or SB errors are most commonly occurring for industry software developers. Causes of human errors during development activities include complex development environment, lack of concentration and going down rabbit holes. Additionally I identify seven themes which software developer human errors tend to fall within.

- **Cont2** New understanding about how industry software developers mitigate human errors during development activities e.g. focusing, using headphones, automation and planning. Maintaining focus and concentration were reported frequently as a means of mitigating against human error.

- **Cont3** The use of SA training using the OODA loop can lead to a

reduction in the number of human errors software developers make. Designed, implemented an online training tool which allows industry software developers to improve their situation awareness. The tool was trialled using industry software developers and showed a reduction in the number of human errors being made.

- **Cont4** Gain an initial understanding of the layers (developer, process, tool and management) in the Swiss Cheese Model. Identification of that situation awareness is a large route through for errors and developing a training tool which aids industry software developers in reducing the size of the hole in the slice.

My doctoral research makes the following contributions to practitioners;

- **Know your own weaknesses.** Every developer is different and struggles with different concepts. Our analysis shows a variety of types of errors that developers make. Developers becoming more conscious of the human errors they commonly make and actively checking for these can help reduce errors.

- **Use cognitive training.** We have shown that using cognitive training, like the OODA loop, seems to help decision making and can reduce the human errors a developer makes.

- **Simplify your workload.** One of the biggest causes of human error reported by the developers in our study was the complexity of the development environment. Reducing the cognitive load by simplifying the complexity of the development environment could reduce human errors. Actions such as minimising the number of simultaneous development tasks and closing down unnecessary tools and windows can help reduce the cognitive load.

- **Communicate carefully with stakeholders outside your team.** Our study suggests that a relatively high number of mistakes are related to communicating with stakeholders outside of the development team. Ensuring that communication is clearly understood seems important to reducing mistakes.

## 7.3 Research Limitations

Limitations of my doctoral research include;

- **Generalisability / Sample Size** My findings only provide an initial insight into human error within software development. The sample size could have been increased had there been more industry collaborators and or funding available to the participants of the study.
- **Participant Bias** Participants could easily hide part of or manipulate a response to what they believe I as the researcher is intending on hearing. I attempted to address this by keeping each example as high level as possible and providing little SE related examples so they could choose examples from across their experience. Participants may change their response because they fear their management finding out about the errors they are making in their work. I attempted to address this by providing reassurance that the data in raw form would not be shared with anyone outside the research team and only processed / annoymised data would be published.
- **Interview Bias** I conducted all the interviews to aid reducing in irregularities of interview style. There is a danger that I could detect trends as the number of interviews grew and try focusing participants answers on specific responses. I mitigated this by only allowing the free flowing nature of the interview to occur based on what a participant says.

## 7.4 Future Work

My doctoral research has provided a preliminary insight into causes of and mitigation strategies of human errors alongside the development and implementation of a tool to aid in human error reduction for industry software developers. This section explores some future work that could be conducted based of my doctoral research and is presented in three sections.

The below items relate to my first study detailed in Chapter 4, which investigated what SB errors industry software developers makes and how they mitigate them.

1. **What forms do other human errors and how are these currently mitigated?** Investigate what form RB and KB errors manifest as within software development. Additionally gain an understanding of how RB and KB errors are currently mitigated.
2. **Generalise Findings** Investigate whether a larger participant pool shows similar themes in relation to SB errors and mitigation methods.

The below items relate to my second study detailed in Chapter 5, which investigated whether undergoing SA training leads to a reduction in human errors.

1. **Further understand adverse issues of using the OODA loop** Develop an understanding as to why planning errors increased when industry software developers use OODA loop.
2. **Generalise Findings** Run the OODA loop study with a larger participant pool to determine if its use does indeed lead to a reduce in human errors.
3. **Do results last?** Extend the OODA study to investigate whether the intervention has long lasting effects on human error reduction.

4. **Are there other benefits?** Determine if there are any other benefits to industry software developers in using the OODA loop other than reducing the number of human errors.

The below items relate further work that is needed within the scope of reducing human error within software development.

1. **Blame Culture** Investigate the type of blame culture that exists within software development and its effect on human error.
2. **Use other models** Investigate does the use of other models within human error research lead to a decrease in the number of software developer human errors.
3. **Violations** Develop an understanding in what and why industry software developers commit violation human errors.
4. **Monitoring developer factors** I know many factors e.g. amount of sleep, mental/physical/emotional well being etc, can determine whether a developer makes errors in their work. Develop a list of which of the measures industry software developers are willing to monitor actively.

My doctoral thesis highlights the need for research to target the human errors industry software developers make. I have shown that there are layers of human error that need work. My doctoral thesis focused on SA from the developer layer. I have shown that simple safeguard measures can be introduced to aid industry software developers in their work. This simple and effective measure can easily be rolled out within industry aiding millions to improve their SA skills, in turn reducing the number of human errors they make.

# Bibliography

Abimanyi-Ochom, J., Bohingamu Mudiyanselage, S. & Catchpool, M. (2019), 'Strategies to reduce diagnostic errors: a systematic review.', *BMC Med Inform Decis Mak* **19**(1), 174.

Anu, V., Walia, G. & Bradshaw, G. (2017), 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', SIGCSE '17, ACM, New York, NY, USA, pp. 39–44.

Anu, V., Walia, G., Hu, W., Carver, J. & Bradshaw, G. (2016*a*), 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, pp. 531–536.

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

Auerbach, C. & Silverstein, L. B. (2003), *Qualitative data: An introduction to coding and analysis*, Vol. 21, NYU press.

Bailey, B. P. & Konstan, J. A. (2006), 'On the need for attention-aware

systems: Measuring effects of interruption on task performance, error rate, and affective state', *Computers in Human Behavior* **22**(4), 685 – 708. Attention aware systems.
**URL:** *http://www.sciencedirect.com/science/article/pii/S074756320500107X*

Barlett, J. E., Kotrlik, J. W. & Higgins, C. C. (2001), 'Organizational research: Determining appropriate sample size in survey research', *Information technology, learning, and performance journal* **19**(1), 43.

Basili, V. R. (2007), The role of controlled experiments in software engineering research, *in* 'Empirical Software Engineering Issues. Critical Assessment and Future Directions', Springer, pp. 33–37.

Basit, T. (2003), 'Manual or electronic? the role of coding in qualitative data analysis', *Educational research* **45**(2), 143–154.

Beynon-Davies, P. (1999), 'Human error and information systems failure: the case of the london ambulance service computer-aided despatch system project', *Interacting with Computers* **11**(6), 699–720.

Bird, C., Nagappan, N., Murphy, B., Gall, H. & Devanbu, P. (2011), Don't touch my code!, *in* 'Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering - SIGSOFT/FSE '11', ACM, p. 4.

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

Braun, V. & Clarke, V. (2006), 'Using thematic analysis in psychology', *Qualitative research in psychology* **3**(2), 77–101.

Brennan, P. A., Mitchell, D. A., Holmes, S., Plint, S. & Parry, D. (2016), '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* **54**(1), 3–7.

Brykczynski, B. (1999), 'A survey of software inspection checklists', *ACM SIGSOFT Software Engineering Notes* **24**(1), 82.

Carthey, J. (2013), 'Understanding safety in healthcare: The system evolution, erosion and enhancement model', *Journal of public health research* **2**, e25.

Clay-Williams, R. & Colligan, L. (2015), 'Back to basics: checklists in aviation and healthcare', *BMJ Quality & Safety* **24**(7), 428–431.
**URL:** *https://qualitysafety.bmj.com/content/24/7/428*

Connelly, L. M. (2008), 'Pilot studies', *Medsurg Nursing* **17**(6), 411–413.

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

Dey, I. (2003), *Qualitative data analysis: A user friendly guide for social scientists*, Routledge.

Easterbrook, S., Singer, J., Storey, M.-A. & Damian, D. (2008), Selecting Empirical Methods for Software Engineering Research, *in* 'Guide to Advanced Empirical Software Engineering', Guide to advanced empirical software engineering, Springer, pp. 285–311.

Ely, J. W., Graber, M. L. & Croskerry, P. (2011), 'Checklists to reduce diagnostic errors', *Academic Medicine* **86**(3), 307–313.

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

Endsley, M. R. (1995), 'Toward a theory of situation awareness in dynamic systems', *Human factors* **37**(1), 32–64.

Endsley, M. R. & Garland, D. J. (2000), Pilot situation awareness training in general aviation, *in* 'Proceedings of the Human Factors and Ergonomics Society Annual Meeting', Vol. 44, SAGE Publications Sage CA: Los Angeles, CA, pp. 357–360.

Ericsson, K. A. & Simon, H. A. (1984), *Protocol analysis: Verbal reports as data.*, the MIT Press.

Falessi, D., Juristo, N., Wohlin, C., Turhan, B., Münch, J., Jedlitschka, A. & Oivo, M. (2018), 'Empirical software engineering experts on the use of students and professionals in experiments', *Empirical Software Engineering* **23**(1), 452–489.

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

Graziotin, D., Fagerholm, F., Wang, X. & Abrahamsson, P. (2017), 'On the unhappiness of software developers', *CoRR* **abs/1703.04993**.
**URL:** *http://arxiv.org/abs/1703.04993*

Hightower, T. A. (2007), 'Boyd's ooda loop and how we use it', *Tactical Response* .

Hoda, R., Noble, J. & Marshall, S. (2010), Using grounded theory to study the human aspects of software engineering, *in* 'Human Aspects of Software Engineering', pp. 1–2.

Höst, M., Regnell, B. & Wohlin, C. (2000), 'Using students as subjects—a comparative study of students and professionals in lead-time impact assessment', *Empirical Software Engineering* **5**(3), 201–214.

Hu, W., Carver, J., Anu, V., Walia, G. & Bradshaw, G. (2016), Detection of Requirement Errors and Faults via a Human Error Taxonomy:

A Feasibility Study, *in* 'International Symposium on Empirical Software Engineering and Measurement', Vol. 08-09-Sept of *ESEM '16*, ACM, New York, NY, USA, p. 30:10.

Hu, W., Carver, J. C., Anu, V., Walia, G. & Bradshaw, G. (2017), 'Defect Prevention in Requirements Using Human Error Information: An Empirical Study', pp. 61–76.

Hu, W., Carver, J. C., Walia, G. & Anu, V. (2017a), 'Understanding Human Errors In Software Requirements : An Online Survey'.

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

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

Huang, F. & Liu, B. (2017), 'Software defect prevention based on human error theories', *Chinese Journal of Aeronautics* .

Huang, F., Liu, B. & Huang, B. (2012), '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.

Huang, F., Liu, B., Song, Y. & Keyal, S. (2014), 'The links between human error diversity and software diversity: Implications for fault diversity seeking', *Science of Computer Programming* **89**(PART C), 350–373.

Ioannidis, J. P. (2008), 'Why most discovered true associations are inflated', *Epidemiology* pp. 640–648.

Ioannou, G., Louvieris, P. & Clewley, N. (2019), 'A Markov Multi-phase Transferable Belief Model for Cyber Situational Awareness', *IEEE Access* .

Kadam, P. & Bhalerao, S. (2010), 'Sample size calculation', *International Journal of Ayurveda Research* **1**(1), 55.

Kitchenham, B. A., Society, I. C., Pfleeger, S. L., Pickard, L. M., Jones, P. W., Hoaglin, D. C., Emam, K. E. & Rosenberg, J. (2002), 'Preliminary Guidelines for Empirical Research in Software Engineering', *Main* **28**(8), 721–734.

Kitchenham, B., Sjøberg, D. I., Dybå, T., Brereton, O. P., Budgen, D., Höst, M. & Runeson, P. (2012), 'Trends in the quality of human-centric software engineering experiments–a quasi-experiment', *IEEE Transactions on Software Engineering* **39**(7), 1002–1017.

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

Ko, A. J., Latoza, T. D. & Burnett, M. M. (2015), 'A practical guide to controlled experiments of software engineering tools with human participants', *Empirical Software Engineering* **20**(1), 110–141.

LaToza, T. D., Venolia, G. & DeLine, R. (2006), Maintaining mental models, *in* 'Proceeding of the 28th international conference on Software engineering - ICSE '06', ICSE '06, ACM, New York, NY, USA, p. 492.

Leape, L. L. (1994), 'Error in medicine', *Jama* **272**(23), 1851–1857.

Li, S. Y. W., Blandford, A., Cairns, P. & Young, R. M. (2008), '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* **14**(4), 314 – 328.

Linneberg, M. S. & Korsgaard, S. (2019), 'Coding qualitative data: A synthesis guiding the novice', *Qualitative research journal* .

Marino, M. J. (2018), 'How often should we expect to be wrong? statistical power, p values, and the expected prevalence of false discoveries', *Biochemical pharmacology* **151**, 226–233.

Marshall, M. N. (1996), 'Sampling for qualitative research', *Family practice* **13**(6), 522–526.

Maxwell, J. (1992), 'Understanding and validity in qualitative research', *Harvard educational review* **62**(3), 279–301.

Meyer, A. N., Murphy, G. C., Fritz, T. & Zimmermann, T. (2019), *Developers' Diverging Perceptions of Productivity*, Apress, Berkeley, CA, pp. 137–146.

Morris, A. (2015), *A practical introduction to in-depth interviewing*, SAGE, Los Angeles.

Oppenheim, A. N. (1992), *Questionnaire design, interviewing and attitude measurement*, 2nd edn.

Palmer, C. & Pe'er, I. (2017), 'Statistical correction of the winner's curse explains replication variability in quantitative trait genome-wide association studies', *PLoS genetics* **13**(7), e1006916.

Peitek, N., Siegmund, J., Apel, S., Kästner, C., Parnin, C., Bethmann, A., Leich, T., Saake, G. & Brechmann, A. (2020), 'A look into programmers' heads', *IEEE Transactions on Software Engineering* **46**(4), 442–46.

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

Pirzadeh, L. (2010), 'Human Factors in Software Development: A Systematic Literature Review'.

Procida, D. (2017), Fighting the controls: Madness and tragedy in programming. DjangoCon Europe 2017.
**URL:** *https://www.youtube.com/watch?v=qI7NZV-rak0*

Rasmussen, J. (1983), 'Skills, rules, and knowledge; signals, signs, and symbols, and other distinctions in human performance models', *IEEE Transactions on Systems, Man, and Cybernetics* **SMC-13**(3), 257–266.

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

Reason, J. (1995), 'Understanding adverse events: human factors.', *BMJ Quality & Safety* **4**(2), 80–89.

Reason, J. (2000), 'Human error: models and management', *BMJ: British Medical Journal* **320**(7237), 768.

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

Ribeiro, G. d. S. R., Silva, R. C. d., Ferreira, M. A. d. A. A. A. & Silva, G. R. d. (2016), 'Slips, lapses and mistakes inthe use of equipment by nurses in an intensive care unit', *Revista da Escola de Enfermagem da USP* **50**, 419 – 426.

Richards, C. (2020), 'Boyd's ooda loop'.

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

Salman, I., Misirli, A. T. & Juristo, N. (2015), Are students representatives of professionals in software engineering experiments?, *in* '2015 IEEE/ACM 37th IEEE International Conference on Software Engineering', Vol. 1, pp. 666–676.

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

Seaman, C. (1999), 'Qualitative methods in empirical studies of software engineering', *IEEE Transactions on Software Engineering* **25**(4), 557–572.

Shadish, W. R., Cook, T. D., Campbell, D. T. et al. (2002), *Experimental and quasi-experimental designs for generalized causal inference/William R. Shedish, Thomas D. Cook, Donald T. Campbell.*, Boston: Houghton Mifflin,.

Sjøberg, D. I., Anda, B., Arisholm, E., Dybå, T., Jørgensen, M., Karahasanović, A. & Vokáč, M. (2003), Challenges and recommendations when increasing the realism of controlled software engineering experiments, *in* 'Empirical methods and studies in software engineering', Springer, pp. 24–38.

Sjøberg, D. I., Hannay, J. E., Hansen, O., Kampenes, V. B., Karahasanovic, A., Liborg, N.-K. & Rekdal, A. C. (2005), 'A survey of controlled experi-

ments in software engineering', *IEEE Transactions on Software Engineering* **31**(9), 733–753.

Sjøberg, D. I. K., Anda, B., Arisholm, E., Dybå, T., J"rgensen, M., Karahasanovic, A., Koren, E. F. & Vokác, M. (2002), Conducting realistic experiments in software engineering, *in* 'Proceedings of the 2002 International Symposium on Empirical Software Engineering', ISESE '02, IEEE Computer Society, Washington, DC, USA, pp. 17–.

Sommerville, I. (2015), *Software Engineering*, Always learning, 10th edn, ADDISON WESLEY Publishing Company Incorporated.

Stol, K.-J., Ralph, P. & Fitzgerald, B. (2016), Grounded theory in software engineering research: a critical review and guidelines, *in* 'Proceedings of the 38th International Conference on Software Engineering', pp. 120–131.

Sugden, C., Athanasiou, T. & Darzi, A. (2012), What are the effects of sleep deprivation and fatigue in surgical practice?, *in* 'Seminars in thoracic and cardiovascular surgery', Vol. 24, Elsevier, pp. 166–175.

Sykes, E. R. (2011), 'Interruptions in the workplace: A case study to reduce their effects', *International Journal of Information Management* **31**(4), 385–394.

Thibodeau, P. (2013), 'India to overtake U.S. on number of developers by 2017'.

Tomayko, J. E. (2002), 'A comparison of pair programming to inspections for software defect reduction', *Computer Science Education* **12**(3), 213–222.

Vegas, S., Dieste, O. & Juristo, N. (2015), Difficulties in running experiments in the software industry: Experiences from the trenches, *in* 'Proceedings

of the Third International Workshop on Conducting Empirical Studies in Industry', CESI '15, IEEE Press, Piscataway, NJ, USA, pp. 3–9.

Wagner, S., Mendez, D., Felderer, M., Graziotin, D. & Kalinowski, M. (2020), Challenges in survey research, *in* 'Contemporary Empirical Methods in Software Engineering', Springer, pp. 93–125.

Walia, G. S. & Carver, J. C. (2009), 'A systematic literature review to identify and classify software requirement errors', *Information and Software Technology* **51**(7), 1087–1109.

Walsh, K. (2003), 'Qualitative research: Advancing the science and practice of hospitality', *Cornell Hotel and Restaurant Administration Quarterly* **44**(2), 66–74.

Wickens, C. D. (2002), 'Situation awareness and workload in aviation', *Current directions in psychological science* **11**(4), 128–133.

Williams, L., Maximilien, E. M. & Vouk, M. (2003), Test-driven development as a defect-reduction practice, *in* '14th International Symposium on Software Reliability Engineering, 2003. ISSRE 2003.', IEEE, pp. 34–45.

Wright, M. C., Taekman, J. M. & Endsley, M. R. (2004), 'Objective measures of situation awareness in a simulated medical environment', *BMJ Quality & Safety* **13**(suppl 1), i65–i71.

# Appendix A

# Skill-based (SB) Errors

## A.1 Inattention

### A.1.1 Double-Capture Slips

Reason (1990) reports

> 'these are so named because they involve two distinct, though causally related, kinds of capture. First, the greater part of the limited attentional resource is claimed either by some internal preoccupation or by some external distractor at a time when a higher-order intervention (bringing the work-space into the control loop momentarily) is needed to set the action along the currently intended pathway. As a result, the control of action is usurped by the strongest schema leading onwards from that particular point in the sequence. Such slips are lawful enough to permit reasonably firm predictions regarding when they will occur and what form they will take. The necessary conditions for their occurrence appear to be (a) the performance of some well-practised activity in familiar surroundings, (b) an intention to depart from custom, (c) a departure point beyond which the

'strengths' of the associated action schemata are markedly different, and (d) failure to make an appropriate attentional check. The outcome, generally, is a strong habit intrusion, that is, the unintended activation of the strongest (i.e., the most contextually frequent) action schema beyond the choice point.'

### A.1.2 Omissions following interruptions

Reason (1990) reports

> 'in some instances, the failure to make an attentional check is compounded by some external event. For example: (a) "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." (b) "I walked to my bookcase to find the dictionary. In the process of taking it off the shelf, other books fell onto the floor. I put them back and returned to my desk without the dictionary." (c) "While making tea, I noticed that the tea caddy was empty. I got a fresh packet of tea from the shelf and refilled the caddy. But then I ommitted to put the tea in the pot, and poured boiling water into an empty kettle." Lapses (b) and (c) suggest that secondary corrective routines (rule-based solutions to regularly planned encountered 'hiccups' in a routine) can get 'counted in' as part of the planned sequence of actions, so that when the rule-based activity is over, the original sequence is picked up at a point one or two steps further along.'

### A.1.3 Reduced Intentionality

Reason (1990) reports

> 'it frequently happens that some delay intervenes between the formulation of an intention to do something and the time for

this activity to be executed. Unless it is periodically refreshed by attentional checks in the interim, this intention probably will become overlaid by other demands upon the conscious workspace. These failures of prospective memory lead to a common class of slips and lapses that take a wide variety of forms. These include detached intentions ("I intended to close the window as it was cold. I closed the cupboard door instead.") environmental capture ("I went into my bedroom intending to fetch a book. I took off my rings, looked in the mirror and came out again - without the book.") and multiple sidesteps ("I intended to go to the cupboard under the stairs to turn off the immersion heater. I dried my hands to turn off the switch, but went to the larder instead. After that, I wandered into the living room, looked at the table, went back to the kitchen, and then I remembered my original intention"). Sometimes these error take the form of states rather than actions i.e., lapses rather than slips): the what-am-I-doing-here experience ("I opened the fridge and stood there looking at its contents, unable to remember what it was I wanted." and the even more frustrating I-should-be-doing-something-but-I-can't-remember-what experience.'

### A.1.4 Perceptual Confusions

Reason (1990) reports

> 'the characteristics of these fairly common errors suggest that they occur because the recognition schemata accept as a match for the proper object something that looks like it, is in the expected location or does a similar job. These slips could arise because, in a highly routinised set of actions, it is unnecessary to invest the same amount of attention in the matching pro-

cess. With relatively unusual or unexpected stimuli, attentional processing brings noncurrent knowledge to bear upon their interpretation. But with oft-repeated tasks, it is likely that the recognition schemata, as well as the action schemata, become automatised to the extent that they accept rough rather than precise approximations to the expected inputs. This degradation of the acceptance criteria is in keeping with 'cognitive economics' and its attendant liberation of attentional capacity. Thus, perceptual slips commonly take the form of accepting look-alikes for the intended object ("I intended to pick up the milk bottle, but actually reached out for the squash bottle."). A closely-related variety involves pouring or placing something into a similar but unintended receptacle ("I put a piece of dried toast on the cat's dish instead of in the bin." "I began to pour tea into the sugar bowl.").'

### A.1.5 Interference Errors

Reason (1990) reports

> 'two currently active plans, or within a single plan, two action elements, can become entangled in the struggle to gain control of the effectors. This results in incongruous blends of speech and action ("I had just finished talking on the phone when my secretary ushered in some visitors. I got up from behind the desk and walked to greet them with my hand outstretched saying 'Smith speaking'.")'

## A.2 Overattention

### A.2.1 Omissions

Reason (1990) reports

> 'consider the situation in which one interrupts some reverie to enquire where one is in the tea-making sequence Mistimed checks such as these can produce at least two kinds of wrong assessment. Either one concludes that the process is further along than it actually is, and, as a consequence, omits some necessary step like putting the tea in the pot or switching on the kettle (omission). Or, one decides that is has not yet reached the point where it actually is and then repeats an action already done, such as setting the kettle to boil for a second time or trying to pour a second kettle of water into an already full teapot (repetition).'

### A.2.2 Repetitions

Reason (1990) reports

> 'consider the situation in which one interrupts some reverie to enquire where one is in the tea-making sequence Mistimed checks such as these can produce at least two kinds of wrong assessment. Either one concludes that the process is further along than it actually is, and, as a consequence, omits some necessary step like putting the tea in the pot or switching on the kettle (omission). Or, one decides that is has not yet reached the point where it actually is and then repeats an action already done, such as setting the kettle to boil for a second time or trying to pour a second kettle of water into an already full teapot (repetition).'

### A.2.3 Reversal

Reason (1990) reports that

> 'a rare but revealing kind of slip can appear in bi-directional sequences. An inappropriately timed check can cause am action sequence to double back on itself (reversal), as in the following cases. (a) "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." (b) "I got the correct fare out of my purse to give to the bus conductor. A few moments later I put the coins back into the purse before the conductor had come to collect them."'

# Appendix B

# Supporting Material For Study One

## B.1 Study Introduction

### B.1.1 Participant Information Sheet

**A Participant Sheet**

**Study title**

What skill based human errors do developers make during the development of software

**Invitation Paragraph**

My name is Bhaveet Shah, I am a second year PhD student at Brunel University London in the Department of Computer Science. I would like to invite you to take part in a research study. Before you decide you need to understand why the research is being done and what it would involve for you. Please take time to read the following information carefully. Ask questions if anything you read is not clear or would like more information. Please take time to decide whether or not to take part.

**What is the purpose of the study?**

I am trying to see if we can determine which skill based human errors occur when developers a writing code and if developers try to mitigate against any errors.

**Why have I been invited to participate?**

You have been invited to participate in this study as you are known to a member of the research team as someone that has participated in the development phase of software e.g. developer or manager.

**Do I have to take part?**

As participation is entirely voluntary, it is up to you to decide whether or not to take part. If you do decide to take part you will be given this information sheet to keep and be asked to sign a consent form. If you decide to take part you are free to withdraw at any time and without giving a reason.

**What will happen to me if I take part?**

Nothing will happen to you personally as a result of this research study. Firstly you and your team will be briefed about skill based human errors. I will ask you about any skill based errors you experienced. To aid with recall any errors I will provide a crib sheet of error. The study will not require you to visit the university as part of this research study.

**What do I have to do?**

You will be required to read this information sheet, attend the skill based human error training, be observed while developing code and participate in the post development interview.

**What are the possible disadvantages and risks of taking part?**

There are no anticipated risks with participating in this study. However, if you experience any distress following participation you are encouraged to inform Bhaveet Shah on Bhaveet.Shah@brunel.ac.uk.

**What are the benefits of taking part?**

There are no direct benefits to yourself from taking part in this research study. However you will aid the research team to determine which skill based human errors occur during the development phase of software. The research team will be able to share the results of the study, if published.

**What if something goes wrong?**

If you have any complaints about the project in the first instance you can contact Bhaveet Shah on Bhaveet.Shah@ brunel.ac.uk. If you feel your complaint has not been handled to your satisfaction you

can contact Tracy Hall on t.hall3@lancaster.ac.uk. Any complaints/comments may be forwarded to the chair of University Research Ethics Committee (UREC). They can be contacted by email at res-ethics@brunel.ac.uk.

If you are harmed by taking part in this research project, there are no special compensation arrangements. If you are harmed due to someone's negligence, then you may have grounds for a legal action but you may have to pay for it.

**Will my taking part in this study be kept confidential?**

All information which is collected about you during the course of the research study will be kept strictly confidential and not shared outside the research team. Your responses will be anonymised, by having personal information removed so that an individual cannot be identified from it.

**What will happen to the results of the research study?**

The results from the observations and post development interview will be analysed and used to guide the phase of research. The anonymised results may be used if the findings are published.

**Who is organising and funding the research?**

Bhaveet Shah is organising the research as part of his PhD in Computer Science at Brunel University London. The research is funded by Brunel University London.

**What are the indemnity arrangements?**

Brunel provides appropriate insurance cover for research which has received ethical approval.

**Who has reviewed the study?**

The College of Engineering, Design and Physical Sciences research ethics committee has reviewed this study.

**Include a passage on the University's commitment to the UK Concordat on Research Integrity**

Brunel University is committed to compliance with the Universities UK Research Integrity Concordat. You are entitled to expect the highest level of integrity from our researchers during the course of their research.

**Contact for further information and complaints**

For further information on this study please contact Bhaveet Shah on Bhaveet.Shah@brunel.ac.uk. The supervisor of the research project is Professor Tracy Hall and can be contacted on t.hall3@lancaster.ac.uk. Any complaints/comments may be forwarded to the chair of UREC. They can be contacted by email at res-ethics@brunel. ac.uk.

### B.1.2 Consent Sheet

**CONSENT FORM: What skill based human errors do developers make during the development of software?**

*The participant should complete the whole of this sheet*

*Please tick the appropriate box*

| | YES | NO |
|---|---|---|
| **Have you read the Research Participant Information Sheet?** | ☐ | ☐ |
| **Have you had an opportunity to ask questions and discuss this study?** | ☐ | ☐ |
| **Have you received satisfactory answers to all your questions?** | ☐ | ☐ |
| **Who have you spoken to?** | | |
| **Do you understand that you will not be referred to by name in any report concerning the study?** | ☐ | ☐ |
| **Do you understand that you are free to withdraw from the study:** | | |
| • **at any time?** | ☐ | ☐ |
| • **without having to give a reason for withdrawing?** | ☐ | ☐ |
| • **(where relevant, adapt if necessary) without affecting your future care?** | ☐ | ☐ |
| **(Where relevant) I agree to my interview being recorded.** | ☐ | ☐ |
| **(Where relevant) I agree to the use of non-attributable direct quotes when the study is written up or published.** | ☐ | ☐ |
| **Do you agree to take part in this study?** | ☐ | ☐ |

**Signature of Research Participant:**

**Date:**

**Name in capitals:**

## B.2 Explanation of Skill-based (SB) errors

**Omission**

*Description of Error Type:* Omissions are when you conclude the process is further along than it actually is, and, as a consequence, omit a necessary step.

*Real World Example of Error Type:* Forgetting to turn the kettle on in the tea making process.

**Repetition**

*Description of Error Type:* 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.

*Real World Example of Error Type:* Setting the kettle to boil for a second time.

**Reversal**

*Description of Error Type:* Checking something outside a sequence, causing you to double back on the sequence.

*Real World Example of Error Type:* 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 interruptions**

*Description of Error Type:* Forgetting something due to an external event.

*Real World Example of Error Type:* 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**

*Description of Error Type:* Unintentionally activating a strongly related action pattern.

*Real World Example of Error Type:* I intended only to take my shoes off, but took my socks off as well.

**Reduced Intentionality**

*Description of Error Type:* Some delay intervenes between the formulation of an intention to do something and the time for this activity to be executed.

*Real World Example of Error Type:* I opened the fridge and stood there looking at its contents, unable to remember what is was I wanted.

**Perceptual Confusion**

*Description of Error Type:* 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.

*Real World Example of Error Type:* I intended to pick up the milk bottle, but actually reached out for the squash bottle.

**Interference Errors**

*Description of Error Type:* Two active plans, or two parts of a single plan can become entangled.

*Real World Example of Error Type:* 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'.

## B.3 Sample Questions to ask Interviewee

1. Omissions are when you conclude the process is further along than it actually is, and, as a consequence, omit a necessary step. An example of an omission is forgetting to turn the kettle on in the tea making process. Please describe instances of omissions that you have encountered during your development work.
2. What mitigation strategies did you use / have you started to use for the described omissions(s)?
3. 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. An example of a repetition is setting the kettle to boil for a second time. Please describe instances of repetitions that you have encountered during your development work.
4. What mitigation strategies did you use / have you started to use for the described repetitions(s)?
5. Reversals are checking something outside a sequence, causing you to double back on the sequence. An example of a reversal is when 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. Please describe instances of reversals that you have encountered during your development work.
6. What mitigation strategies did you use / have you started to use for the described reversals(s)?
7. Omission following interruption is forgetting something due to an external event. An example of omission following interruption is when 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. Please describe in-

stances of omission following interruptions that you have encountered during your development work.

8. What mitigation strategies did you use / have you started to use for the described omission following interruptions(s)?

9. Double-capture slips are unintentionally activating a strongly related action pattern. An example of double-capture slips are when I intended only to take my shoes off, but took my socks off as well. Please describe instances of double-capture slips that you have encountered during your development work.

10. What mitigation strategies did you use / have you started to use for the described double-capture slips(s)?

11. Reduced intentionality is when some delay intervenes between the formulation of an intention to do something and the time for this activity to be executed. An example of reduced intentionality is when I opened the fridge and stood there looking at its contents, unable to remember what is was I wanted. Please describe instances of reduced intentionality that you have encountered during your development work.

12. What mitigation strategies did you use / have you started to use for the described reduced intentionality(s)?

13. Perceptual confusion is when 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. An example of perceptual confusion is when I intended to pick up the milk bottle, but actually reached out for the squash bottle. Please describe instances of perceptual confusion that you have encountered during your development work.

14. What mitigation strategies did you use / have you started to use for the described perceptual confusion(s)?

15. Interference error is when two active plans, or two parts of a single plan can become entangled. An example of interference error is when 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'. Please describe instances of interference errors that you have encountered during your development work.
16. What mitigation strategies did you use / have you started to use for the described interference error(s)?

## B.4 Demographic Questions

1. What gender do you identify as? [**Male, Female, Other**]
2. What age band do you fit into? [**18-24, 25-34, 35-44, 45-54, 55-64, 65+**]
3. How many full years of industry experience do you currently have? [**<1, 1-3, 4-7, 8-10, 10+**]
4. What is your current job role (If you have multiple roles, please state your primary role. If you have left industry, please state your primary role while in industry.)? [**Freetext**]
5. What industry do you currently work with (If you are self employed / contracting / employed by a software house, what is the industry your contract currently lies with. If you have left industry, please state the primary industry you worked in within while in industry.)? [**Freetext**]
6. What is the current project type you are working on? [**Open Source System (OSS) / Closed Source System (CSS)**]
7. What is the primary programming language you use? [**Freetext**]

## B.5 List Of Error Themes

Below is the list of all identified error themes;

1. Missing documentation
2. Misunderstood the requirement
3. Database development omissions
4. Incorrect sequencing
5. Test All cases
6. Incomplete requirements
7. Time Pressure
8. Forgetting environmental changes
9. Tiredness
10. Repeated Testing
11. Learning Systems
12. Inserting duplicate code
13. Redoing a body of work
14. Replicating error handling
15. Work Pressure
16. Communication Gap
17. New feature, breaks master branch
18. Refactoring code
19. Modifying the wrong file
20. Being distracted by someone else
21. Going down a rabbit hole
22. Resolving merge conflicts

23. Context Switching
24. Crisis causing immediate shift from BAU
25. Assumptions
26. Distractions from things around the workbench
27. Task Urgency
28. Autopilot
29. Not planning properly
30. Understanding the work
31. Thinking you have found a bug
32. Something goes wrong in my mind
33. Inexperience
34. Multiple workstations
35. Reusing old commands
36. Not concentrating
37. Mistaking x for y
38. Using the wrong environment
39. Version inconsistencies of software
40. Problems transplanting code
41. Waiting for things to build
42. Forgetfulness
43. Hardware failure
44. Requirements change during development
45. Auto complete
46. Thinking you know everything

47. Multiple workspaces
48. Planning a workflow
49. Language switching
50. Copy and pasting incorrectly
51. Debugging multiple versions of the same software
52. Unfamiliar with tools
53. Looking in the wrong place
54. Version inconsistences of code
55. Being stressed
56. Being impatient with the development tools
57. Noise

# Appendix C

# Supporting Material For Study Two

## C.1 Study Introduction

### C.1.1 Participant Information Sheet

**A Participant Sheet**

**Study title**

Can training software developers on human error reduce the number of skill based errors they make in their development work?

**Invitation Paragraph**

My name is Bhaveet Nagaria, I am a third year PhD student at Brunel University London in the Department of Computer Science. I would like to invite you to take part in a research study. Before you decide you need to understand why the research is being done and what it would involve for you. Please take time to read the following information carefully. Ask questions if anything you read is not clear or would like more information. Please take time to decide whether or not to take part.

**What is the purpose of the study?**

I am trying to see if we can reduce the number of skill based human errors developers make during development.

**Why have I been invited to participate?**

You have been invited to participate in this study as you are known to a member of the research team as someone that has participated in the development phase of software e.g. developer or manager.

**Do I have to take part?**

As participation is entirely voluntary, it is up to you to decide whether or not to take part. If you do decide to take part you will be given this information sheet to keep and be asked to sign a consent form. If you decide to take part you are free to withdraw at any time and without giving a reason.

**What will happen to me if I take part?**

You will be asked to record any human errors that occur during a two week period. At the beginning of week two you will be provided training on situation awareness. The study will not require you to visit the university as part of this research study.

**What do I have to do?**

You will be required to read this information sheet and answer some demographic questions. Every day for a two week window we ask you to keep track of any human errors (specific to software development) that occur. Examples of human error can be seen in the table below. To assist you in keeping this record we have supplied you with a record sheet with some worked examples. At the beginning of week two you will complete training on situation awareness.

| |
|---|
| Forgetting to turn the kettle on in the tea making process. |
| Setting the kettle to boil for a second time or trying to pour a second kettle of water into an already full teapot. |
| 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. |
| 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. |
| I intended only to take my shoes off, but took my socks off as well. |

| |
|---|
| I opened the fridge and stood there looking at its contents, unable to remember what is was I wanted. |
| I intended to pick up the milk bottle, but actually reached out for the squash bottle. |
| I had just finished talking on the phone when my secretary ushered in some visitors. I got up from behind the desk and walked to greet them with my hand outstretched saying 'Smith speaking'. |

**What are the possible disadvantages and risks of taking part?**

There are no anticipated risks with participating in this study. However, if you experience any distress following participation you are encouraged to inform Bhaveet Nagaria on Bhaveet.Nagaria@brunel.ac.uk.

**What are the benefits of taking part?**

By taking part in this research study you will be trained in a method of improving your situation awareness. You will aid the research team to determine whether this intervention can reduce the number skill based human errors which occur during the development phase of software. The research team will be able to share the results of the study, if published.

**What if something goes wrong?**

If you have any complaints about the project in the first instance you can contact Bhaveet Nagaria on Bhaveet.Nagaria@ brunel.ac.uk. If you feel your complaint has not been handled to your satisfaction you can contact Tracy Hall on Tracy.Hall@lancaster.ac.uk. Any complaints/comments may be forwarded to the chair of University Research Ethics Committee (UREC). They can be contacted by email at res-ethics@brunel.ac.uk.

If you are harmed by taking part in this research project, there are no special compensation arrangements. If you are harmed due to someone's negligence, then you may have grounds for a legal action but you may have to pay for it.

**Will my taking part in this study be kept confidential?**

All information which is collected about you during the course of the research study will be kept strictly confidential and not shared outside the research team. Your responses will be anonymised, by having personal information removed so that an individual cannot be identified from it.

**What will happen to the results of the research study?**

The results from the records and training will be analysed to determine if there is a change in skill based errors. The anonymised results may be used if the findings are published.

**Who is organising and funding the research?**

Bhaveet Nagaria is organising the research as part of his PhD in Computer Science at Brunel University London. The research is funded by Brunel University London.

**What are the indemnity arrangements?**

Brunel provides appropriate insurance cover for research which has received ethical approval.

**Who has reviewed the study?**

The College of Engineering, Design and Physical Sciences research ethics committee has reviewed this study.

**University's commitment to the UK Concordat on Research Integrity**

Brunel University is committed to compliance with the Universities UK Research Integrity Concordat. You are entitled to expect the highest level of integrity from our researchers during the course of their research.

**Contact for further information and complaints**

For further information on this study please contact Bhaveet Nagaria on Bhaveet.Nagaria@brunel.ac.uk. The supervisor of the research project is Professor Tracy Hall and can be contacted on Tracy.Hall@lancaster.ac.uk. Any complaints/comments may be forwarded to the chair of UREC. They can be contacted by email at res-ethics@brunel. ac.uk.

### C.1.2 Consent Sheet

**CONSENT FORM: Can training software developers on human error reduce the number of skill based errors they make in there development work?**

*The participant should complete the whole of this sheet*

*Please tick the appropriate box*

| | YES | NO |
|---|---|---|
| **Have you read the Research Participant Information Sheet?** | ☐ | ☐ |
| **Have you had an opportunity to ask questions and discuss this study?** | ☐ | ☐ |
| **Have you received satisfactory answers to all your questions?** | ☐ | ☐ |
| **Who have you spoken to?** | | |
| **Do you understand that you will not be referred to by name in any report concerning the study?** | ☐ | ☐ |
| **Do you understand that you are free to withdraw from the study:** | | |
| • **at any time?** | ☐ | ☐ |
| • **without having to give a reason for withdrawing?** | ☐ | ☐ |
| • **(where relevant, adapt if necessary) without affecting your future care?** | ☐ | ☐ |
| **(Where relevant) I agree to my interview being recorded.** | ☐ | ☐ |
| **(Where relevant) I agree to the use of non-attributable direct quotes when the study is written up or published.** | ☐ | ☐ |
| **Do you agree to take part in this study?** | ☐ | ☐ |

**Signature of Research Participant:**

**Date:**

**Name in capitals:**

## C.2 Demographic Questions

1. What gender do you identify as? [**Male, Female, Other**]
2. What age band do you fit into? [**18-24, 25-34, 35-44, 45-54, 55-64, 65+**]
3. How many full years of industry experience do you currently have? [**<1, 1-3, 4-7, 8-10, 10+**]
4. What is your current job role (If you have multiple roles, please state your primary role. If you have left industry, please state your primary role while in industry.)? [**Freetext**]
5. What industry do you currently work with (If you are self employed / contracting / employed by a software house, what is the industry your contract currently lies with. If you have left industry, please state the primary industry you worked in within while in industry.)? [**Freetext**]
6. What is the current project type you are working on? [**Open Source System (OSS) / Closed Source System (CSS)**]
7. What is the primary programming language you use? [**Freetext**]
8. Can we contact you to follow up / clarify any details? If yes, please specify your email. [**Freetext**]

## C.3 Logging Sheet

| Participant Name: | |
|---|---|

| | Details of Human Error | | Context |
|---|---|---|---|
| | Human Error | Consequence of Error | |
| *Example Error One: Day One* | *Communication failure between two of us working in the same area at the same time* | *Merge conflicts* | *Normal working conditions*<br>*Pair programming while error occurred* |
| *Example Error Two: Day One* | *Hot fixing on the live server, failing to run any tests* | *Live system went down; total failure* | *Time Critical*<br>*High Pressure* |
| *Example Error One: Day Two* | *No human errors detected today* | *N/A* | *N/A* |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

## C.4 Training Package Slides

### C.4.1 Video 1: Introduction to SA

Intro to situation awareness. You can watch the same 'Intro To Situation Awareness' video here: `https://www.dropbox.com/s/j8b9aahy331ff8j/TrainingOneIntroToSA.mp4?dl=0` Screenshots of the slides used in this video can be viewed below.

# Situation Awareness

1

## What is it?

- Basic
  - Knowing what is going on around us
- Advanced
  - The perception of the elements in the environment within a volume of time and space, the comprehension of their meaning and the projection of their status in the near future

2

## Why does it matter?

- We cannot remain aware of everything which is going on around us.
- We need to employ methods to help us maintain a high level of awareness.

3

## Example use of Situation Awareness in Aviation

- Fitts & Jones (1947) found that pilots were confusing wing flap and landing gear controls.
- Dekker (2005) reports the solution was to attached a rubber wheel to landing gear control & a small wedge to the flap control.
- Dekker (2005) reports that the solution went on to become a certification requirement.

Fitts, P.M. & Jones, R.E. (1947), analysis of factors contributing to 460 'pilot-error' experiences in operating aircraft controls, Aero Medical Laboratory Wright-Patterson AirForce Base, OH.

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

4

## Time to watch the next video!

Link to the next video is in the email

5

### C.4.2 Video 2: Introduction to the OODA Loop

Intro to OODA loop. You can watch the same 'Intro To OODA Loop' video here: `https://www.dropbox.com/s/xz2pjh1t895gt1v/TrainingTwoIntroToOODA.mp4?dl=0` Screenshots of the slides used in this video can be viewed below.

# OODA Loop

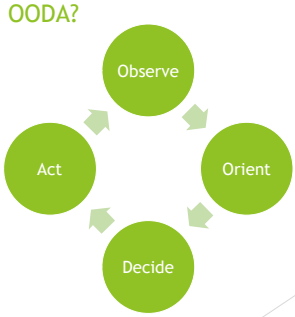
1

## What is the OODA Loop?

- Tool originally developed by military strategist John Boyd
- To explain how individuals and organizations can win in uncertain and chaotic environment
- Four step loop to help with critical thinking

2

## So what is OODA?




3

## OODA - Observe

- Gather as much information as possible.
  - Use as many sources as possible

4

## OODA - Orient

- Analyse the information you have to establish:
  - Where you are
  - Where you need to be

5

## OODA - Decide

- Develop plans of action
- Decide on the most appropriate plan

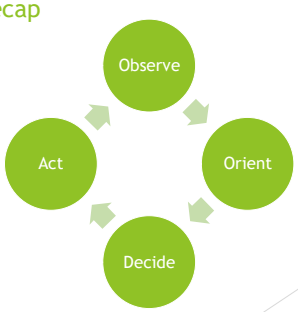
6

## OODA - Act

- Implementing the selected plan.

7

## OODA - Recap




8

## Example use of the OODA Loop

- Scenario: You are in the middle of making a cup of tea to take on your journey to work. You realise that you have run out of tea bags and only have coffee left.
- Observe
  - Recognise that you wish to make a mug of tea.
  - Recognise that you only have coffee.
- Orient
  - Do I really want tea or will coffee do? Do I need to have tea now?
- Decide
  - Make a mug of coffee instead.
  - Run to the shop, buy tea bags and risk being late for work.
  - Make tea at work / buy a tea from the train station café.
- Act
  - Make tea at work / buy a tea from the train station café.

9

## Time to watch the next video!

Link to the next video is in the email

10

### C.4.3 Video 3: Applied OODA Loop

Applied OODA loop in software development. You can watch the same 'Intro To Applied OODA LOOP' video here: `https://www.dropbox.com/s/kf67z508w3dv9n5/TrainingThreeOODAExamples.mp4?dl=0` Screenshots of the slides used in this video can be viewed below.

# OODA Loop Examples in Software Development

1

## Example One: Refactoring

- Scenario: You are in the flow of implementing a new piece of functionality. You have to touch someone else's code and notice it has not been written efficiently.
- Observe
  - Recognise you are implementing new functionality.
  - Recognise you have seen a potential refactoring opportunity.
- Orient
  - Is the refactoring essential to the implementation of the new functionality?
- Decide
  - Do the refactoring while you are in the codebase.
  - Make a note to come back and refactor the code as separate task after implementing the new functionality.
- Act
  - Continue implementing the new functionality without any refactoring.

2

## Example Two: Interruption

- Scenario: You are in the flow doing your work and your boss interrupts you. They ask you to do a piece of work for them.
- Observe
  - Recognise you are doing a piece of work.
  - Recognise you have a request to do some other work.
- Orient
  - Where does the new work fit in relation to the existing work?
- Decide
  - Do the bosses work immediately to keep them happy.
  - The bosses work is low priority, continue with current work.
- Act
  - Continue doing the existing piece of work.

3

## Example Three: Refactoring

- Scenario: You are fixing a bug in a client server application which controls a carpark. The processInput logic in the CarParkState class seems to be skipped every time the application is run.
- Lets take a look

4

# Time to complete the quiz!

Link to the quiz is in the email

5

## C.5 Quiz for SA Training Package

**Section 1 – About OODA**

Q1) What is the loop you should use in critical thinking?
- Orient, Observe, Act, Decide
- Observe, Decide, Orient, Act
- Observe, Orient, Decide, Act
- Orient, Decide, Observe, Act

Scenario: You notice that you are hungry. You remember there is a Greggs down the street and it's before 11am, which means they are still serving sausage rolls. You go to Greggs. You eat a delicious sausage roll (or maybe two, why not, you're already there).

Q2) What is the 'Observe' stage of the scenario?
- You remember there is a Greggs down the street and it's before 11am, which means they are still serving sausage roll.
- You notice that you are hungry.
- You go to Greggs.
- You eat a delicious sausage roll (or maybe two, why not, you're already there).

Q3) What is the 'Orient' stage of the scenario?
- You eat a delicious sausage roll (or maybe two, why not, you're already there).
- You notice that you are hungry.
- You go to sausage roll.
- You remember there is a Greggs down the street and it's before 11am, which means they are still serving sausage roll.

Q4) What is the 'Decide' stage of the scenario?
- You go to Greggs.
- You notice that you are hungry.
- You eat a delicious sausage roll (or maybe two, why not, you're already there).
- You remember there is a Greggs down the street and it's before 11am, which means they are still serving sausage roll.

Q5) What is the 'Act' stage of the scenario?
- You notice that you are hungry.
- You eat a delicious breakfast sub (or maybe two, why not, you're already there).
- You go to Greggs.
- You remember there is a Greggs down the street and it's before 11am, which means they are still serving sausage roll.

**Section 2 – Applied OODA in SE**

Scenario: You notice that there is a major defect in the live system. You remember that there is a backup to the old version. You locate the old backup. You restore the old version while the defect is fixed.

Q6) What is the 'Act' stage of the scenario?
- You hotfix the defect on the live server
- You fix the defect on your local and deploy it to the server
- You restore the old version while the defect is fixed.
- You do nothing and leave the system with a major defect.

Q7) What is the ‘Orient’ stage of the scenario?
- You establish how severe the defect is.
- You remember that there is a backup to the old version.
- You recall your boss telling you that should not roll back versions on live.
- You remember the passwords to the live server so a hotfix can be performed.

Q8) What is the ‘Observe’ stage of the scenario?
- You see on social media that the system is not working as intended.
- You get a call from the client telling you the live system is down.
- You go to use the system and find that it is down.
- You notice that there is a major defect in the live system.

Q9) What is the ‘Decide’ stage of the scenario?
- You could do a hot fix.
- Rolling back the system to the last stable position.
- You find the backup.
- You are not senior enough to make the call.

**Section 3 – About You**
Help us to get feedback to you individually. The data will be removed and anonymised once we have delivered feedback to you.

Q10) What is your name?

Q11) What is your email address?

## C.6 Follow Up Questions

Participants complete a two part online questionnaire to allow us to better understand how they experienced learning about the OODA loop and participating in the study. Below I detail the questions we ask our participants and they type of responses I allow them to provide us with.

Part one asks them about the OODA Training.

1. Did you learn anything new? If no, where did you learn this? [**Freetext**]
2. How actionable was the information you received in the OODA training package? [**Freetext**]
3. Has the OODA training led to an improvement to your work? [**Yes/No**]
4. Will you continue to use the OODA loop in your work? [**Yes/No**]
5. Do you have any comments on the OODA training? [**Freetext**]

The second part of the questionnaire asks the participants about the study.

1. How easy did you find the study to complete? [**Easy/Average/Difficult**]
2. Were you able to log all human errors encountered over the study? If no, why? [**Freetext**]
3. Did you find the daily email reminders helped? [**Yes/No**]
4. Did you find the log sheet helped you to record all human errors that occurred? [**Yes/No**]
5. Do you have any comments about the study? [**Freetext**]

## C.7 Coding of Human Errors

Below are the definitions of slips, lapses and mistakes are provided by Reason (1990).

**Slip** A slip is a result of carelessness or inattentive actions for example day-to-day activities such as fat fingering.

**Lapse** A lapse is a result of forgetfulness or a failure of memory, examples could include intending to do task A but not doing so due to an interruption and then resuming with another task.

**Mistake** A mistake is the most severe type of error and a result of lack of knowledge during the planning stage of an activity. An example could include misdiagnosing a patient due to lack of experience and or not exploring their signs/symptoms properly.

# Appendix D

# Snippets of Raw Data

This appendix contains example snippets of data collected throughout my doctoral research. Figure D.1 & D.2 shows examples of Transcripts from Study 1. Figure D.3 shows a screen dump of part of the Trello board used for coding of the transcripts. Figure D.4 shows a log sheet from a participant in Study 2. Figure D.5 shows a screen dump of part of the results from the online training quiz. Figure D.6 shows a screen dump of part of the results from the follow up questionnaire.

17:57 - Double-capture slips

I: We are looking at double-capture slips here. So unintentionally activating a strongly related action pattern. The example is you only intend on taking your shoes off but you take your socks off as well.

P26: Okay.

I: At a low level, something maybe like doing a git commit as well as a git add. That may help.

P26: Yup. Okay.

I: Not to worry if nothing is coming to mind.

P26: Okay, well nothing is coming of the top of my head.

I: We will skip on and something does spring up you can always let me know later.

Figure D.1: Snippet 1 of Raw Data from Transcript

## 12:57 - Reduced Intentionality

I: The next one is reduced intentionality, there are two examples here. The first example is you are feeling so cold so you go to close the window but on your way you close the cupboard door. The





second example is you are staring into the fridge blankly thinking why I am here. What happens here is there is a plan that formulates in your head, there is some delay between the formulation of this plan and the execution. This causes you to execute the plan incorrectly.

P13: Yeah, I'm searching my mind. I'm I've certainly seen examples where of my own behaviour where I am planning to implement a particular change in my mind and I've got a good understanding of what that is. I spend an hour or two to implement it and then I come across something that will trigger me to identify the fact that I have gone down a blind alley. What I am implementing was not in the spirit of what I initially intended to implement, but that will be where you have got a delays, minimum days.

I: So not quite the instantaneous delay its.

P13: Yeah I can't think of any examples.

Figure D.2: Snippet 2 of Raw Data from Transcript

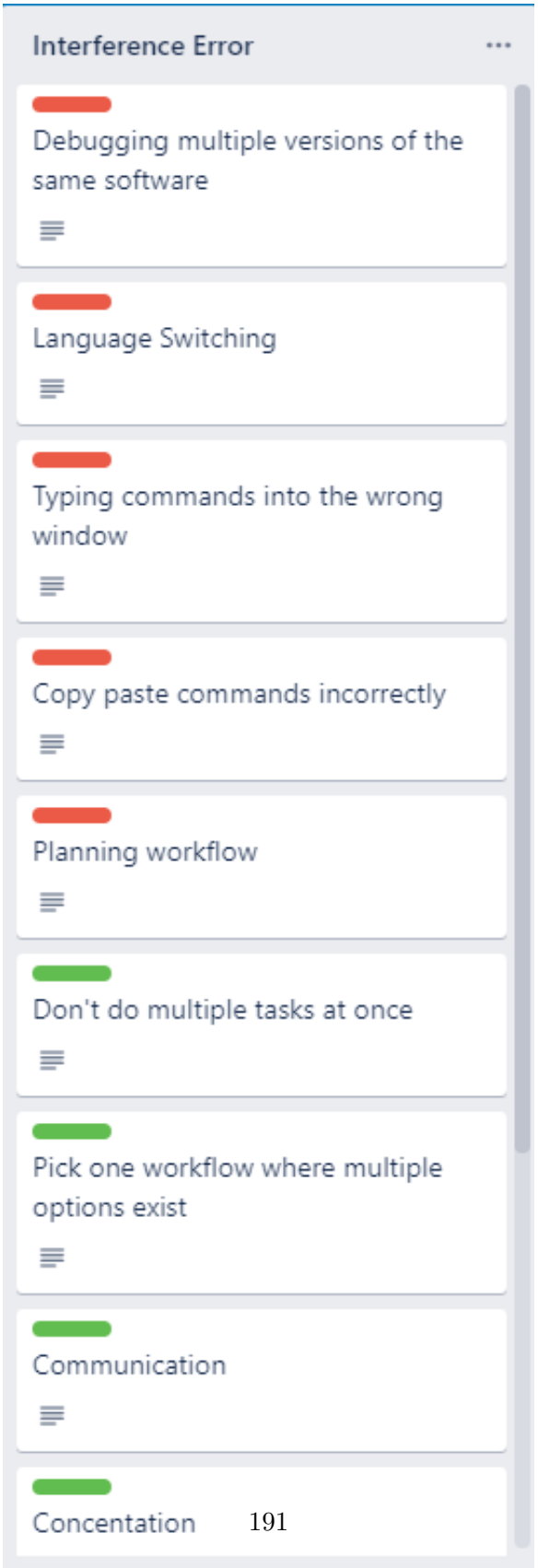





Figure D.3: Snippet of Coded Data from Trello

| | Details of Human Error | | Context |
|---|---|---|---|
| | Human Error | Consequence of Error | |
| Day One | Ran `sudo shutdown now` by choosing wrong entry in terminal history | None – sudo prompted me for a password and I cancelled the command at that point. | I use a window manager called i3 which has no shutdown button. So I run `sudo shutdown now` when I need to shutdown and this gets stored in my bash history. Perhaps there is a better approach that could avoid this potential error. |
| Day Two | No human errors noticed today | | |
| Day Three | Marked a JIRA ticket as merged without merging the PR | None as I happened to notice the PR was still open when I checked my GitHub notifications later – but this could have led to a feature being listed in the release notes but not actually being present. | We use JIRA for tickets and GitHub for pull requests. The 2 are not automatically synced, so when a PR is merged, the merger has to manually mark the JIRA ticket as merged. |
| Day Four | No human errors noticed today | | |
| Day Five | Tested a data migration on a table that turned out to be empty, invalidating the test | None – I picked up on this during a later similar test. Worst case – this could have blocked a deployment to our production environment. | Data migration: set of SQL statements that changes the structure of a database table<br><br>Empty table: table with no rows, where any structure changes are permitted (vs a table with data which restricts the possible changes) |
| Day Six | Merged a git branch which had an import error in some test files, wasn't caught by CI because I ran tests directly on the feature branch instead of "base branch + feature branch" (the latter approach would have caught this) – so what I should have done is push "base branch + feature branch" to CI instead of "feature branch" - this requires some manual fiddling though and isn't usually necessary. In an ideal world this process would be automated to always run "base branch + feature branch". | 2 failing tests on our CI – fixed shortly after | |
| Day Seven | Tested a ticket on the wrong git branch | Thought the change I was testing was broken, but turned out to be my mistake | |
| Day Eight | Started an Amazon EC2 instance with image "CI Server" instead of "CI Worker" as intended. | Confusion, lost time. In the end I recreated the EC2 instance after realising the issue. | |

| | | | |
|---|---|---|---|
| Day Nine | No human errors noticed today | | |
| Day Ten | No human errors noticed today | | |

| Participant Name: | P5(Named removed and replaced with participant number by BN) |
|---|---|

Figure D.4: Example of Raw Data from a Log Sheet

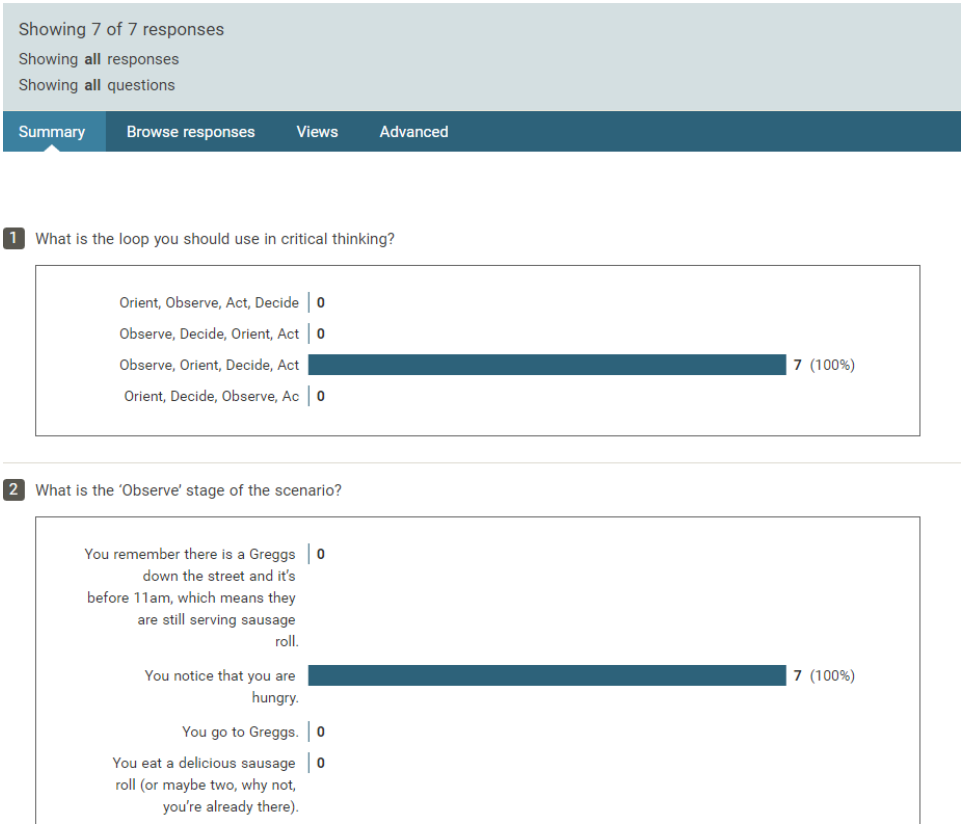
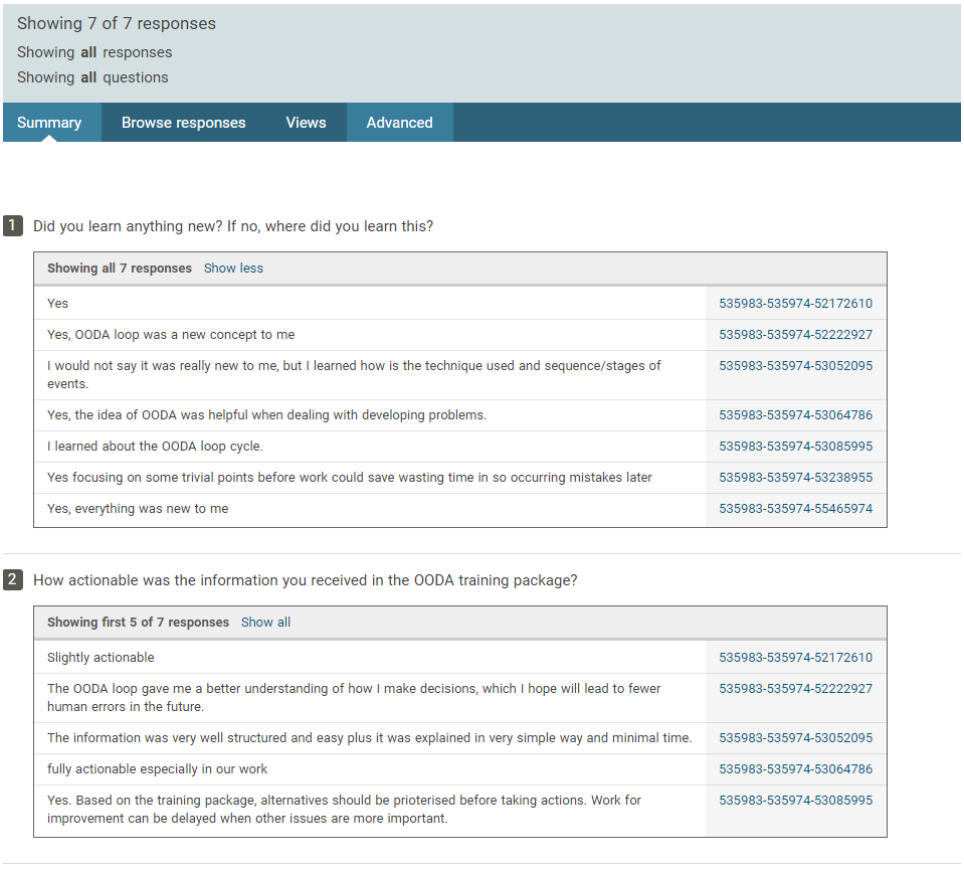



Figure D.5: Snippet of Raw Data from Quiz



Showing 7 of 7 responses

Showing **all** responses

Showing **all** questions

Summary | Browse responses | Views | Advanced

1 Did you learn anything new? If no, where did you learn this?

**Showing all 7 responses** Show less

| | |
|---|---|
| Yes | 535983-535974-52172610 |
| Yes, OODA loop was a new concept to me | 535983-535974-52222927 |
| I would not say it was really new to me, but I learned how is the technique used and sequence/stages of events. | 535983-535974-53052095 |
| Yes, the idea of OODA was helpful when dealing with developing problems. | 535983-535974-53064786 |
| I learned about the OODA loop cycle. | 535983-535974-53085995 |
| Yes focusing on some trivial points before work could save wasting time in so occurring mistakes later | 535983-535974-53238955 |
| Yes, everything was new to me | 535983-535974-55465974 |

2 How actionable was the information you received in the OODA training package?

**Showing first 5 of 7 responses** Show all

| | |
|---|---|
| Slightly actionable | 535983-535974-52172610 |
| The OODA loop gave me a better understanding of how I make decisions, which I hope will lead to fewer human errors in the future. | 535983-535974-52222927 |
| The information was very well structured and easy plus it was explained in very simple way and minimal time. | 535983-535974-53052095 |
| fully actionable especially in our work | 535983-535974-53064786 |
| Yes. Based on the training package, alternatives should be prioterised before taking actions. Work for improvement can be delayed when other issues are more important. | 535983-535974-53085995 |

Figure D.6: Snippet of Raw Data from Follow Up Questionnaire