Helping developers to help each other: a technique to facilitate understanding among professional software developers

Gail Ollis

Faculty of Science and Technology
Bournemouth University

August 2019

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

Abstract

Helping developers to help each other: a technique to facilitate understanding among professional software developers

Gail Ollis

Much of a professional software developer's work involves amending or extending pre-existing software; creating new software from scratch represents only a small proportion of their time. Even in a brand new project they still need to make sense of work others have done in the emerging software system. This research addresses how developers are helped or hindered in their own daily tasks by the actions of their peers.

The literature commonly focuses on a specific aspect of the work such as code comprehension or the processes by which a project is run. This research instead takes a holistic view that considers all the activities involved in the job, but from a single uniting perspective: rather than conventional measures such as coding errors or delivery timescales, the criterion here is how a developer personally experiences their own productivity to be affected by their peers.

The research used one-to-one interviews to identify common behaviours that help or hinder fellow software developers. Experienced software developers discussed team-friendly (and otherwise) behaviours across the breadth of their typical workplace tasks. The key themes to emerge from this qualitative data make a contribution to the understanding of software development by giving a comprehensive, developer's-eye view of behaviours that help or hinder them across the whole range of tasks that fill their days.

These themes laid the foundation for a practical application of the research. Techniques which had proved engaging and useful in the interviews were adapted into a continuing professional development workshop designed to encourage team discussion on issues of local resonance, selected by participants from those which the interviews had shown to be important. The topics resonated in a way which reinforces the validity of the interview findings. Participants enthusiastically identified useful actions for their own teams and would recommend the workshop to others.

They also saw potential for future development into different workplace scenarios. The workshop is just one application of the understanding contributed by the

research. The principles of good practice from the human perspective that have been identified also offer an empirical foundation that could be of practical use in appraisals, recruitment and any other scenario which requires an understanding of software developers not just as computer programmers but as people.

Contents

List of Figures

List of Tables

Relevant papers and presentations

Conferences Beards, T. Ollis, G., 2018. Cyber security needs cyber neurodiversity. In: PyCon UK 2018 15-19 September 2018 Cardiff, UK.

Ollis, G., 2016. Folklore and fantasy in the information age. In: PyCon UK 2016 15-19 September 2016 Cardiff, UK.

Ollis, G., 2016. Helping programmers get what they want. In: 27th Annual Workshop of the Psychology of Programming Interest Group - PPIG 2016 7 September-10 August 2016 St. Catharine's College, University of Cambridge, UK. Psychology of Programming Interest Group (PPIG 2016).

Ollis, G., 2014. "What Programmers Want". In: ACCU 2014 9-12 April 2014 Bristol.

Ollis, G., 2012. In search of practitioner perspectives on 'good code'. In: 24th Annual Workshop of the Psychology of Programming Interest Group - PPIG 2012 21-23 November 2012 London Metropolitan University, UK. Psychology of Programming Interest Group (PPIG).

Posters

Ollis, G., 2012. Why did they do THAT? The hidden impact of programmers' psychological differences on software development. In: BCS 8th London Hopper Colloquium.

Magazine Articles

Ollis, G., 2009. Santa Claus and Other Methodologies. CVu: The magazine of the ACCU, 20 (6), 3-7.

<u>Formative</u>

Pilot event for the evaluation study.

Ollis, G., 2016. Users are not the only people. In: PyCon UK 2016 15-19 September 2016 Cardiff, UK. Conference workshop to pilot the workshop procedure and materials for the evaluation study. Included a brief presentation of the completed exploratory study.

Summative

Presentations in which research findings were disseminated.

Ollis, G., 2019. Helping programmers to help each other. In: ACCU 2019 11-13 April 2019 Bristol. Peer-reviewed industry conference presentation and workshop. Presentation as for Ollis (2018) followed by a hands-on workshop for delegates.

Ollis, G., 2018. Helping programmers to help each other. In: Qcon San Francisco 2018 5-7 November 2018, San Francisco. Invited industry conference presentation on the completed research.

Ollis, G., 2016. Helping programmers get what they want. In: 27th Annual Workshop of the Psychology of Programming Interest Group - PPIG 2016 7 -10 September 2016 Cambridge, UK. Psychology of Programming Interest Group (PPIG 2016). Peer-reviewed industry conference presentation on results of the exploratory study.

Ollis, G., 2014. "What Programmers Want". In: ACCU 2014 9-12 April 2014 Bristol. Peer-reviewed industry conference presentation and workshop on the findings of the exploratory study. The response also informed the design of the evaluation study.

Ollis, G., 2012. In search of practitioner perspectives on 'good code'. In: 24th Annual Workshop of the Psychology of Programming Interest Group - PPIG 2012 21-23 November 2012 London Metropolitan University, UK. Psychology of Programming Interest Group (PPIG). Peer reviewed academic conference presentation on the early research design.

Ollis, G., 2012. Why did they do THAT? The hidden impact of programmers' psychological differences on software development. In: BCS 8th London Hopper Colloquium. Poster presentation on the early research design.

Complementary

Presentations not directly about the research but contributing to my profile in speaking about human aspects in software development.

Beards, T. Ollis, G., 2018. Cyber security needs cyber neurodiversity. In: PyCon UK 2018 15-19 September 2018 Cardiff, UK. Peer-reviewed industry conference presentation on the benefits to computer security of a neurodiverse team.

Ollis, G., 2016. Folklore and fantasy in the information age. In: PyCon UK 2016 15-19 September 2016 Cardiff, UK. Invited keynote to an industry conference on taking a pragmatic, humanly achievable approach to software methodologies.

Ollis, G., 2009. Santa Claus and Other Methodologies. CVu: The magazine of the ACCU, 20 (6), 3-7. Industry magazine article summarising the material of Ollis(2008), below.

Ollis, G., 2008. Santa Claus and Other Methodologies. In: ACCU 2008 2-5 April 2008 Oxford. Peer-reviewed industry conference presentation on taking a pragmatic, humanly achievable approach to software methodologies.

Ollis, G., 2007. Advocating Agility. In: ACCU 2007 11-14 April 2007 Oxford. Peer-reviewed industry conference presentation on the difficulties of introducing a new methodology in a software development organisation.

Acknowledgements

My deepest thanks to my supervisory team for all their sage advice and for complementing each other so well. Each of them has made this work better in their own way and I am grateful to have had such a diverse and knowledgeable team supporting me. In alphabetical order, my gratitude goes to:

Dr Sherry Jeary, who saved me from myself on many occasions, asked difficult but incredibly useful questions and taught me so much about spinning a Golden Thread.

Professor Keith Phalp, with whose help and enthusiasm the research stayed firmly focused on delivering a technique of practical use to software developers.

Dr Jacqui Taylor, who applied psychological expertise. As lead supervisor Jacqui also kept the whole show on the road and arranged essential pit-stops.

Dr Kevin Thomas, who also brought psychological expertise to bear and asked good questions. He played a crucial role in helping me to write more clearly about software by being the audience in my mind's eye. Kevin, this was quite literally written for you!

I would also like to thank Russel Winder, whose enthusiasm and public support helped me to forge a confident identity as a software development psychologist.

To Paul, the other half of a great team, without whose loving, patient and enthusiastic support this work could never even have started. You have held my hand as I followed my heart. Now it's your turn.

In loving memory of my parents, who passed away during the course of my research. Its completion owes so much to the faith they always showed in me and they would have been so very proud.

Chapter 1

Introduction

1.1 Programmers are people

This research concerns the behaviour of programmers who develop professional software. There is a long history of studying software from the perspective of the user; Grudin (2012), for instance, charts the evolution of Human Computer Interaction (HCI) from the days of vacuum tube computers all the way through to 'ubiquitous computing', where interaction now occurs not only with computers but also with the everyday devices in which digital technology is embedded. However end users are not the only people with human experience of computer programs. There is also a large but less visible audience of fellow programmers who maintain and extend the original author's source code, from which the executable program is built. My own experience as a professional programmer made me curious about the psychology of programming: my colleagues were skilled, intelligent and experienced; all of them could make a program behave as required; but each had a "fingerprint" in their work which had a profound effect on the ease with which others could understand that work. This research examines the nature of software development from the perspective of programmers, and asks how the behaviour of their peers affects them as users not of the end product but the raw code and its supporting infrastructure of tests, version control, builds and bug reports — features which are invisible to end users.

1.2 Terminology

Throughout this document people who write software for a living are described interchangeably as programmers, software developers, developers and software engineers. Within the professional community there is debate over terminology, as illustrated by community bloggers such as Kaplan (2014), Skorkin (2010) and Vlatko (2015) all trying to define the difference between terms. Their posts reflect not just the diversity but also the semantic complexity of official job titles and the job descriptions that people who program use for themselves. Rather than engage in the semantic debate, this document makes no distinction between these contested terms. Similarly programming, development or coding is used for the computer programming task they do and program, software, system or code for what is produced.

1.3 Software engineering is complex

There are both rewards and frustrations for programmers. There is, for instance, pleasure in craftsmanship; Lingel and Regan (2014) noted embodied perceptions of programming that are akin to those for physical crafts, e.g., ”It’s in your fingertips”. There is also frustration in failing to solve a problem. It is telling that Spraul (2012, pp.21–22) included a section entitled “Don’t Get Frustrated” in chapter 1 of his book on how to think like a programmer. But regardless of the pain or pleasure involved, professional software gets written because it is paid for. It might be an off-the-shelf software product for which a vendor has identified a market. In the past this was typically “shrink-wrap” software bought on storage media and installed on the buyer’s computer, but increasing numbers of applications are hosted in the cloud, where customers can subscribe to use “software as a service” such as Microsoft Office 365. According to Mintel (2018) the cloud computing market now attracts more than twice the revenue of five years ago. Software can also be bespoke, created specially for commercial customers or “internal” customers wanting software for use in-house. It might be “invisible”, embedded into a diverse range of specialised hardware from cardiac pacemakers to radars. While its presence is not always apparent, software is pervasive in the modern world.

Creating a working software system is not a simple task, so projects often struggle to deliver on time and within budget. The proportion of failing projects is unclear. Glass (2003), for example, acknowledged that there are many such projects but considered the so-called “Software Crisis” to be overstated, later exploring the claims for failure rates of 70% and suggesting that the truth was closer to 10% (Glass, 2005). Even the definition of “failing” may be contested. In a case study of a software development project that had gone over time and over budget (Linberg, 1999), perceived as a failure by the business, all the software developers interviewed reported it as the best or second best project they had ever worked on due to the interesting challenge, the performance of the team and the quality of the product. From personal experience there can indeed be great job satisfaction in a difficult project, but it is not always the case.

Regardless of high or low job satisfaction for the developers, difficulties clearly exist when the outcome of a failing project can be a complete cancellation before it sees the light of day. Projects which continue to the end may deliver working software which is late, costs more than expected, or compromises on the originally intended features. For example, Charette (2005) reported high-profile projects spanning 14 years which were abandoned and/or cost their North American companies millions of dollars in losses. In the UK, examples include a new air traffic control system which finally opened over-budget and six years late (BBC, 2002). Failure can also occur after deployment, sometimes with a catastrophic and embarrassingly public error such as NatWest's inability to process its millions of banking transactions for several days in 2012 (Arthur, 2012) and TSB's even longer outage in 2018 which led to a loss of over one hundred million pounds in a 6-month trading period (BBC, 2018; TSB, 2018).

The suggested causes of failure are wide-ranging; common factors include ill-defined requirements, poor project management and immature technology (Charette, 2005). Creating software is a labour-intensive intellectual task of figuring out how to achieve the required behaviour from available tools: hardware, programming languages, libraries, and software tools. These are complex technological tools wielded by human beings. Furthermore, it is not a solitary task; other people need to understand how the solution works.

1.4 Research questions

The subjective experience of people using software is described in the field of Human Computer Interaction (HCI) as "User eXperience" (UX). Similarly, "Developer eXperience" (DX) can be used to signify how software developers experience their work in creating software. It is usually used in connection with the usability of developer tools and Application Programming Interfaces (APIs) through which developers use the features of a software library or service (Rangel, 2015, is a typical example). The underlying principle is that, like end users, "dev[eloper]s are people too" (Baker, 2016).

Writing software is rarely a single-person effort; the keystrokes entered by a developer alone at their keyboard are just one part of an interconnected system of small pieces of code which must all work in concert within a common infrastructure. Because so much software development builds upon code that already exists, even that one small part is rarely independent of what others have previously done. This is manifest not only in tools and APIs provided by third parties but also in the actions of peers past or present in the same organisation. These actions also play a part in the wider Developer eXperience.

This premise underpins the research, the motivation for which stems from my own experience as a professional programmer over the course of 20 years. Over this time, two things became apparent:

1. Software rarely starts with a blank slate. It is common to be building on some existing software, maintaining or evolving it.

2. The difficulty of such tasks is affected by the way in which the previous work was done. Some programmers consistently produce work that is easy to follow but others do not, and making sense of what they have done is a source of frustration.

Influenced by this personal experience, the research seeks to build a picture of just which peer behaviours are most significant to professional software developers. This is the intent behind the first research question.

Research Question 1: What is the perception of experienced programmers on the peer behaviours that help or hinder them in their own tasks?

This is investigated in the first, exploratory phase of research (Chapters 4 and 5) whose objectives are:

1. To collect opinions from experienced programmers working in a variety of application domains about the helps and hindrances that affect them in the breadth of their everyday work.

2. To identify common themes in their responses.

Having collected this understanding of behaviours which most help or hinder other programmers, the research aims to offer practical applications of the findings. The goal of promoting "team-friendly" software development practices is reflected in the second research question.

Research Question 2: Can experienced programmers' accounts of the peer behaviours that most help or hinder them in their own tasks be used to help others in their practice?

This is explored in the second, evaluative phase of research, whose objectives are:

1. To present the common "helps and hindrances" themes collected in the first phase to groups of professional software developers for discussion.

2. To evaluate the usefulness of such discussion.

3. To identify other avenues through which the material could be useful.

1.5 Outline of the thesis

Difficulties in getting a working system delivered on time and within budget are an abiding part of software development in commercial settings. Many solutions have been proposed, each shaped by a particular perspective on the nature of software development. Chapter 2 gives an overview of the different perspectives. The one consistently successful solution appears to be the rather impractical one of putting the best people on the task:

> "Purely people factors predict project trajectories quite well, overriding choice of process or technology." (Cockburn, 2007, p.43).

Instead of treating individual differences as a confound which interferes with the efficacy of a favoured tool or process, such differences could serve as a constructive starting point rather than a somewhat rueful conclusion. This research therefore takes a cross-disciplinary approach of applying a psychological perspective to a software engineering problem: only some programmers are those "best people" of the idealised team, but what could be done to help all the people behave more effectively?

The behavioural aspects of software engineering have long been under-represented in the literature. Nearly 3 decades ago Curtis and Walz (1991) bemoaned the lack of a social rather than individual psychological perspective. A systematic review by Lenberg, Feldt, and Wallgren (2015) showed that research into 'softer, human aspects' has grown in the intervening years, but still leaves gaps in the field they describe as behavioural software engineering. The research reported here contributes to that field by understanding how the work experiences of programmers are influenced by the actions of others — what behaviours help or hinder them most — and by finding a technique to help them to communicate about these ideas. The approach to these research tasks is described in Chapter 3.

Professional programming is not an individual and independent activity; technical decisions are rarely made with a completely clean slate. Verhoef (2000) and Glass (2003) both gave figures of up to 80% for the effort typically spent on maintenance rather than initial construction of a software product. The work typically demands some degree of integration with what is already there, whether fitting changes to older software or conforming to decisions already made on a new project. Existing code and its supporting infrastructure can help the programmer by clearly signalling how it works and what it is for, or it can hinder them as they struggle to make sense of it. These differences, merely cosmetic from the computer's perspective, render the program more articulate or more confusing to the humans who read it, saving or costing them time and thus saving or costing the business money. Just as importantly, frustration also makes programmers unhappy. The findings of a survey by Ford and Parnin (2015) categorised many causes of such frustration including programming tools, unexpected complexity and peers. The consequences can be external ones such as low productivity and poor quality code, and internal ones such as impaired mental health (Graziotin, Fagerholm, Wang, & Abrahamsson, 2017).

An Exploratory Study, set out in Chapters 4 and 5, focuses on this socio-technical aspect: the interaction of human behaviour with the technological demands of building software. The study asked which of their peers' behaviours affect experienced professional programmers the most. The results represent an analysis of the answers from 28 interviews with programmers who between them have amassed 462 years of experience in the industry. This is an original and holistic inventory of the practices that affect them, spanning as it does the whole breadth of their professional activities but focusing uniquely on one assessment criterion, the effects that the programmers themselves experience. It identifies issues not in terms of general principles of good practice, nor indirect measurements such as numbers of errors in the code, but by how professional programmers perceive that such issues impact upon their progress.

While the picture that emerges from the Exploratory Study reflects some established tenets of good practice, it also illuminates an unanticipated subtle connection between the social and structural aspects of software engineering. Rather than inviting new good practice paradigms (or stronger emphasis on existing ones) at the detailed level of writing code, this work suggests the need for attention at a more macro level, with novel approaches to programmer education and continuing professional development (CPD) to encourage reflection and communication. Chapters 6 and 7 describe an Evaluation Study which tests a technique designed to tackle this challenge: a practical workshop which applies the discoveries made in the Exploratory Study of programmer experiences.

The outcome of the studies suggests scope for continuing research with professional programmers and also for extending the workshop technique to a wider audience. These are discussed in Chapter 8.

Chapter 2

The nature of software development

2.1 Introduction

This chapter explores the many facets of creating a software system and the approaches which are taken in pursuit of doing it better.

The complexity of creating most commercial software demands that it is done not by an individual but a team: software is a joint enterprise (§2.2). There is debate over the kind of skill the team exercises (§2.3); is it art, craft or science? The outcome, regardless, is an executable program achieved by the creation of a written text: the source code which sets out instructions to the computer (§2.4). Just as with a literary text, there is more than one way to characterise it (§2.5). One approach is to make empirical measurements of size and structure. Another is to consider from the perspective of the reader how meaning is deduced from the text. Both angles have been taken in studies to analyse the complexity of source code. Complexity is an issue which makes it difficult for the original author to express their intentions correctly but also affects future readers trying to fix mistakes or add functionality. This means that good programming is not only a technical task of instructing the computer to do something. Doing it well includes a social element (§2.6) in which the needs of future human readers, not just the compiler, are catered for.

It is unsurprising in view of the cost of software development and the scope for human error described in Chapter 1 that ideas for improving software development abound. The social element that is the subject of this research is not entirely overlooked but much of the effort focuses on processes and technical practices (§2.7).

2.2 Software as a joint enterprise

Given the complexity of software projects, the task of developing software is usually a team undertaking.

The need to divide the work among a team and integrate it into a working whole means that each programmer's work is constrained and influenced by the work of their peers. This extends beyond the immediate development of a new project; much of a programmer's time is spent on some kind of maintenance, reuse or further development of existing software. Estimates of just how much time vary. Collating data from 10 studies spread across three decades, Verhoef (2000) cited estimates of software maintenance costs from 50% to a little over 80% of the overall software life-cycle. Similarly, Glass (2003) put the figure at typically 40% to 80% and Grubb

(2003) at 40% to 70% of lifecycle costs. The figures depend in part on the definition of maintenance. It can be a misleading term since it is not used consistently. Tripathy, Naik, Tripathy, and Naik (2014), noting that there is no standard definition of "maintenance", devoted an entire chapter to a detailed explanation of concepts and terms associated with working on existing software. They broadly classified work with existing software under two headings: "maintenance" is post-delivery bug fixing and "evolution" is a process of continual change to implement new requirements. Figures vary between companies and between projects, but one characteristic is consistent: when there is relevant intellectual capital invested in their existing codebase, organisations are unlikely to start a project completely from scratch. Whether fixing a bug, updating and adapting an existing system or even working in a team where they must integrate their brand new component with those written by colleagues, programmers are frequently faced with understanding others' code and the infrastructure around it.

2.3 Software as art, craft or science

This distinction between the requirements of the computer (adherence to strict, unambiguous syntax rules) and human readers of the code (whose needs are more semantic) is reflected in the way software development has been portrayed as an art, a craft, a science and combinations of these. During the early days of computing in the 1950s and 1960s scientific and mathematical portrayals dominated, as reflected in the algebra and logic problems set by the aptitude tests that were common at the time (Abbate, 2012). Computing is still described, although not exclusively, as a discipline similar to mathematics. The national curriculum for computing (Department for Education, 2013), for example, characterises the subject as having "deep links with mathematics, science and design and technology".

The scientific skill of logical reasoning continues to be essential in working out how to achieve a desired output, but for human programmers themselves it is also important how the goal is achieved: what Case and Piñeiro (2006) described as the "intrinsic value" of the code. This is the professional value placed by programmers on how well code is crafted. Such aesthetic values of art and craft play no part in the "instrumental value" to the organisation of code which delivers correct program logic on time and within budget.

The scientific portrayals often focus on production of software: the logical problem-solving task of how to achieve a particular outcome from the computer (and thus deliver instrumental value) using the strict syntax of the languages available. Working software can be achieved without being well-crafted; instrumental value requires only that it does the job, although lack of concern about how it does so may come at a later cost by slowing the delivery of future fixes or enhancements. Portrayals of software development as an art or craft value craftsmanship and take account of how the intrinsic value of the work is perceived by human beings. This may include the process of creating it as well as the finished work; programmers interviewed by Lingel and Regan (2014, p.302) spoke of the pride in making something and the craft of working out how to construct it. Its eventual shape is of no interest to the computer, which simply follows the instructions it is given, but matters to human readers who need to understand the meaning of those instructions.

The creative process characteristic of an art or craft is sometimes construed in software development as something imprecise and uncontrolled which should be eschewed in favour of a process subject to predictable scientific laws. This is illustrated, for example, by the desire of early proponents of structured programming to ”advance programming from an art ...to a science” (Ceruzzi, 2003, p.104, my emphasis). Knuth (1974) noted that from as early as 1959 the literature promoted a drive for computer programming to be transformed from an art to a disciplined science. He frowned upon such a scientific-aesthetic dichotomy, suggesting instead that both are valuable: the rigour of science and the artistry that delivers this with elegant economy rather than ugly complexity.

Knuth went on to say that his books (Volume 1 of The Art of Computer Programming, 2009, was first published in 1968) were intended to help people to write ”beautiful programs”. Elegance is not only satisfying to achieve but also, according to Knuth, a quality that can be recognised and appreciated when reading someone else’s work. While aesthetic qualities may be hard to quantify and some may perhaps be in the eye of the beholder, Fowler (1999, p.75) suggested a number of ”bad smells” that provide a more concrete means of identifying ugly code which would benefit from the refactoring process described in his well-respected book. Although refactoring changes only the structure of the program, and not its behaviour, its purpose is not merely aesthetic. Feathers (2005), in another standard text respected in the industry, asserts that improving the code’s structure makes it easier to understand and maintain.

Author(s)	Subject
Ko, Myers, Coblenz, and Aung (2006)	Information-seeking behaviour during code maintenance
Buse and Weimer (2010)	Metrics for code readability
Abbes, Khomh, Guéhéneuc, and Antoniol (2011)	Impact of anti-patterns on comprehension
Haiduc, Aponte, and Marcus (2010)	Automated code descriptions to aid comprehension
LaToza and Myers (2010)	Questions developers ask during coding
Roehm, Tiarks, Koschke, and Maalej (2012)	Use of comprehension strategies and tools

Table 1: Examples of program comprehension research

2.4 Software as a document

The role of structure is explored in more detail within program comprehension research such as the examples listed in Table 1. These examples are of a kind characterised by Storey's (2006) meta-analysis of program comprehension research as empirical research that seeks to understand cognitive processes (in contrast to technology research that focuses on developing support tools). Understanding a program is not like understanding a linear text, in which the reader can start at the beginning and read through to the end. Indeed the inputs which the program processes — anything from human actions at a user interface to machine-generated messages in an automated monitoring system — are themselves rarely sequential and predictable. The program will be partitioned into many separate modules; modularity is an established design heuristic (cited, for example, in industry textbooks such as McConnell, 2004; Goodliffe, 2006; Hunt & Thomas, 2000) to hide unnecessary details and group data together with the methods that use it. This approach creates silos of information and behaviour that are called upon as needed to deal with the events that the program is designed to respond to.

This modularity has long been a necessary part of making a program manageable; the terms "cohesion" and "coupling" first came into computing parlance in the 1970s as principles of Structured Design (Yourdon & Constantine, 1979). Functional cohesion calls for lines of code in a module to share some common goal; if instead they perform multiple tasks the increased complexity makes it harder to understand what the module is for and how it is achieved. Closely associated with the principle of cohesion within a module, the principle of loose coupling calls for limiting the complexity of the relationship between modules and thus the extent to which one depends on another. The dependencies in a tightly coupled system can create an avalanche of necessary revisions in other modules whenever one module is changed. Paradigms may change but structure continues to be important in program comprehension. In Object Oriented Programming, for instance, low cohesion can manifest in the form of a “Blob”: a class which takes responsibility for too large a part of the program’s functionality. It is one of the antipatterns whose impact on comprehension was investigated by Abbes et al. (2011).

Rather than a simple walk from A to B along a single linear path, comprehension is an exploration through an edifice of communicating modules. This is evident from the kind of information that developers seek when trying to navigate unfamiliar code and understand it sufficiently well to make a change. The questions asked by the professional developers observed by Sillito, Murphy, and De Volder (2006) as they worked on real tasks in their codebase would be readily recognised by their peers in the industry. Sillito et al. analysed these questions into four categories according to the increasing scope of the enquiry: finding a point in the code which appears relevant to the task in hand, finding out more about the structural entity to which that point belongs, exploring relationships with other entities, and at the broadest level, understanding the behaviour achieved by these related entities. Simply finding the right section of code to examine can be a non-trivial exercise requiring search skills to narrow down the possibilities. Sillito et al. observed many cases where participants identified a relevant section of code by actually running the program, using a debugger to flag up whether selected lines were executed in the course of a particular behaviour. The difficulty of determining relevance is evident in one participant’s explanation of this strategy: ”I thought maybe these classes are not even relevant, even though they look like they should be” (Sillito et al., 2006, p.27, my emphasis). It is little wonder that Wilde and Casey (1996) talk about comprehension techniques in terms of ”reconnaissance”.

Theories about the different strategies used in code comprehension tasks have fallen into two main categories characterised by Soloway, Adelson, and Ehrlich (1988) as “top-down” and “bottom-up”. R. Brooks (1983), for example, theorised a top-down approach in which readers use knowledge of the problem domain to hypothesise about the likely processes involved. This resembles the initial breaking down of the problem into computable steps that occurs in the original design of the program (the approach usually associated with experts; Detienne, 2001), but in this case the reader applies their expectations to a search for candidate sections of existing code. Conversely, bottom-up strategies start by looking at lines of code to ascertain their function at a low level, working out what they do algorithmically and building upwards from there to a higher level at which this functionality can be mapped to a useful behaviour in the problem domain. Some researchers have linked differences in strategy to expertise (experts talk about code in abstractions whereas novices talk about individual statements; LaToza, Garlan, Herbsleb, & Myers, 2007) or domain knowledge (all used a form of top-down strategy but those familiar with the domain also included some bottom-up processing; O'Brien & Buckley, 2001). A third strategy is to search by "domain concept" — a feature of interest that is part the program (Rajlich & Wilde, 2002). Comprehension is needed in the context of a specific task to change the behaviour of the program in some way, and only a subset of code will be relevant to the task. If identifiers (the names assigned by programmers to constructs they create within a program) are sufficiently descriptive of the domain concept, a text search can help a programmer home in on locations which may be relevant to their task.

Other research has been concerned not with the direction of the search but the effectiveness of it. Robillard, Coelho, and Murphy (2004), for example, observed approaches to understanding code in a realistically complex modification task (making 5 specified changes to a feature in a codebase of 301 Java classes in 20 packages comprising 64,994 lines of code). Approaches were characterised as either systematic and methodical or ad hoc and opportunistic. The successful participants in the task showed the methodical approach much more often, but not to the exclusion of opportunism. The participants were all students or recent graduates and completed the tasks within a two hour time limit, so this study gives no insight into how increasing familiarity with a codebase or more years of general experience might affect strategy choices.

Hansen, Goldstone, and Lumsdaine (2013) illustrated the importance of experience with an experiment comparing the comprehension of quite subtly different versions of short Python programs which produced identical output. In most cases participants with more experience were faster and more accurate in predicting the output of a program, and better able to spot common errors. But the advantage of experience decreased when presented with code which violated their expectations — for example, a function call add_1(num) contradicted the impression created by its name because a deliberate scope error in the code to increment an integer num meant that there was no effect on the value. This finding chimes with Soloway and Ehrlich's (1984) principle concerning "rules of programming discourse" — conventions which, if broken, make the meaning hard to follow for readers who have absorbed the rules. Novices, not yet so familiar with the rules, are less likely to be affected by violations of them. These experiments on code structure have shown that small manipulations of notation, variable names and even superficial features such as whitespace, none of which is significant in terms of measurable program complexity, nonetheless influence comprehension.

If code comprehension, already quite a demanding task, can be made harder still by code style variations which might appear trivial to a layperson, it is not surprising that a sizeable proportion of the work of changing someone else's software is in understanding it. Abran (2004), drawing on a variety of papers, found figures of between 40% and 60% cited for the proportion of maintenance task time given over to understanding what is already there. Grubb (2003) referred to different papers but reached a similar conclusion: about 50% of the effort involved in a change is understanding the code. Some of that effort will depend on how adept the reader is at applying appropriate comprehension strategies, and some on the articulacy of the code. Boehm et al. (2000, p.23) explicitly incorporated the latter into his process for software cost estimation in the form of a "Software Understanding Increment". This contributes to a multiplication factor in estimates involving code reuse; evolution of existing code that is unfamiliar, obscure, poorly structured or lacks a clear mapping to the application domain costs more.

2.5 Software as a measurable entity

In the pursuit of reducing maintenance costs there have been numerous efforts to measure characteristics that make it harder for people to understand the software when they work on it in future. Much of the attention has focused on those that are most amenable to objective measurement: the structural qualities rather than aspects like obscurity or domain mapping (§2.5.1). The push for a quantitative scientific approach is exemplified by McCabe (1976) calling for a mathematical technique to measure the testability and maintainability of software modules.

There is a clear incentive for businesses to seek simple objective measurements. Identifying testability and maintainability issues is the first step to addressing them, creating the possibility of lower maintenance costs and thus being more competitive and more profitable. The perceptions of the programmers themselves are crucial in understanding the difficulty of software maintenance, but subjective difficulty is less easy to quantify than metrics based on the execution of the code. Even measurements based on more subjective criteria (§2.5.2) have failed to provide a good indication.

Evaluation of software metrics has depended on proxy measures of maintainability rather than an important but more elusive concept, difficulty of the maintenance task. Difficulty may indeed mean that more mistakes are made, but also that the task takes longer, whether or not it is completed without error.

2.5.1 Objective measurements

Studies have evaluated code metrics such as static analysis (Menzies, Greenwald, & Frank, 2007) and process metrics such as the history of changes to a file (Moser, Pedrycz, & Succi, 2008) to see if they are predictive of the number of defects found in the software. Defects are errors, also commonly known as bugs, which cause a program to behave incorrectly. For example the Y2K bug, widely reported in the news in the months before January 2000, was an error in systems designed to store years in two digits rather than four (e.g., 99 rather than 1999). Surviving systems still using this approach were susceptible to unintended behaviour as the assumption of the initial 19 ceased to be valid, although in the event the effort devoted to preventing the Y2K bug meant that there were few manifestations of it (National Geographic Society, 2011). Examples of bugs rarely reach the public domain or this level of notoriety, but companies developing software routinely track all reported defects within internal bug tracking databases.

Defects are thus comparatively easy to count, but they are not a particularly well-calibrated indicator of maintainability because there lacks a consistent means of identifying their relative size and significance; ”a defect is nowhere near as stable a concept as “body fat” or “body water”, for which you can rely on standardized definitions and off-the-shelf instruments” (Bossavit, 2012, pp.113–114). Even an apparently simple measurement of the number of defects can be problematic. Some organisations, for example, create unintended consequences in defect reporting behaviour by assessing performance based on this number (Kaner & Bond, 2004), thus influencing decisions on whether a particular symptom reflects one or many underlying problems. But the lack of a standardised measurement may not be the only problem. In a meta-analysis of 42 studies proposing the predictive power of metrics for software defects, Shepperd, Bowes, and Hall (2014) found significant differences between studies in the performance of the measurements used; by far the largest component of the variance (30%) was accounted for by “researcher group”, i.e. who did the work rather than which measurements they used upon which codebase. They therefore suggested a need for better reporting and sharing of expertise concerning the exact techniques used and also blind analysis done without knowledge of the condition used.

Defect counts can in any case measure the quality of the software only to the extent that they indicate extra work which would not have been necessary if the software were already working perfectly, but they are not indicative of the degree of difficulty of working with that software. Each reported defect is simply a record of a known error; estimates of its size are guesses, data about the effort actually expended to fix it is not necessarily recorded, and there is no standardized definition (Bossavit, 2012) which would permit defects to be compared.

It is hard to measure, but the variation in time taken to effect a change could be considered a more direct sign of the effort involved in making sense of the software than the number of defects. Many metrics do no better at predicting this effort than can be achieved using very simple measures such as the number of lines (Lind & Vairavan, 1989), perhaps in part because they measure only the features that play an executable part. For example: the number of possible paths through the code (McCabe, 1976), the number of operators and operands (Halstead, 1977), the flow of data through the system (Henry & Kafura, 1981), and measures specific to object oriented software such as depth of inheritance (Chidamber & Kemerer, 1994). Such metrics do not measure surface features such as whitespace, line length and identifier naming, which have no effect on the execution of the program.

Curtis, Sheppard, Milliman, Borst, and Love (1979) found that the correlation between complexity metrics and time taken to complete a change, while still no better than the correlation with the program's length, is much more pronounced for programmers with no more than three years' experience. Rather than confound the problems of complexity with those of being a novice, it would be interesting to consider which aspects are the ones to continue to cause difficulties even when programmers have become experienced. For a variety of reasons, programming is not easy to learn. Guzdial and Guo (2014), for example, cited as an issue the "learnability" of languages not designed with their users in mind. Teague et al. (2012) highlighted the difficulty of progressing from concrete examples to a more generalisable abstract understanding and have demonstrated, using a think-aloud protocol, that correct answers do not necessarily reflect correct understanding. Pears et al. (2007) have provided a comprehensive survey of the literature on introductory programming education that spans three decades, and concluded that despite the wealth of research attention it has received the problem remains unsolved. Given the difficulty of teaching and learning the subject, novices already face a degree of difficulty even before being challenged still further by complexity. Experienced programmers, even if not expert, should be past the first of those problems. Experts are observably different in what they want from a programming language (Petre, 1991), how they visualise their task (Petre & Blackwell, 1997) and numerous other ways (Petre, Hoek, & Quach, 2016). Their practices help them to do great work; issues which still nonetheless affect them could be considered the inherent difficulties of the problem.

2.5.2 Subjective measurements

In contrast to the metrics measuring the code from a compiler's perspective, Buse and Weimer (2010) proposed a metric specifically for the readability of code, drawn up from participants' assessments of 100 snippets of open source software. Using this metric in a longitudinal study of evolving open source projects, they showed a relationship between defect density (a ratio of the number of defects to the size of the software) and readability. While the study is a welcome example of actually testing the human perspective directly, rather than a computerised proxy for it, there are reasons for concern about the method used to create the metric. The small size of the snippets (just three lines each) eliminated any clues or misleading cues that exist when code is presented in context. Nor was there any goal for participants other than to report their assessment of readability. They faced no real requirement to understand the code so their responses may reflect something more superficial than the deeper reading that would be required if faced with a complete program and a realistic task to perform. Real life readers of code approach it with a job to do: finding a bug; making a change; checking for mistakes as part of a code review. Eliminating the wider context was, however, a considered choice; Buse and Weimer (2010) were explicitly exploring the readability of low-level details.

The correlation of the Buse and Weimer (2010) metric with defect density suggests that it may measure some comprehensibility characteristics which can make the programming task harder to do correctly. But defect counts, even if they could be standardised, are an imperfect indicator of the impact on programmers. Hall, Zhang, Bowes, and Sun (2014) measured instances of five "code smells" (Fowler, 1999) in three open-source codebases and found that the smells did not consistently influence defects; when they did have a significant effect, the effect size was small. This does not mean that smells are not detrimental, but the nature of their impact is hard to measure. Metrics allow objective measurement of code characteristics, and the number of defects reported in that code can be counted, albeit with the limitations already discussed. It is altogether less easy to count the cost in terms of a programmer's expensive time or to account for characteristics that cannot be measured, such as a colleague's decision to introduce a new third-party library into the system or write code for some imagined future requirement.

It is therefore important to distinguish between the inherent complexity of the problem and what F. Brooks (1987) described as "accidental complexity", a feature not of the problem itself but of the way in which available tools are used to solve the problem. Readability is just such a feature. Buse and Weimer (2010) claimed that it is largely independent of inherent complexity and supported their claim by showing, across 15 different open source projects, that there was little or no correlation between measurements of readability (using the metric they created) and cyclomatic complexity (a software complexity metric which counts the number of possible different paths through the program; McCabe, 1976). This accords with other research exploring the correlation between calculated complexity metrics and human perceptions. Katzmarski and Koschke (2012) found some consistency in the ratings made by over 200 programmers whom they asked which of a pair of Java methods (taken from open source projects) was the more complex, but little agreement between these opinions and the outcome of standardised metrics. These two studies involved undergraduate and postgraduate students and, in the latter case, academics — many with ”substantial” programming experience, although no details of experience are given. It is possible that their assessments might not match those of experienced professional programmers, but for these participants at least the metrics were a poor measure of how human beings perceived the complexity of the code.

2.6 Software development as a social activity

The nature of software development involves not just a computer being made to do things but also human beings understanding what it does. The great majority of professional programmers (90% according to McConnell, 2004) work in teams of three or more so however well-architected the system may be, being able to understand the behaviour of components written by others is essential. Communication – a skill not typically taught to learner programmers (Brechner, 2003) – is needed both between contemporaries and between the present author and future maintainers and enhancers of their code. Knuth (1984) suggested that instructing the computer is not the main task. Rather, programmers should concentrate on writing a program that can explain the intention to other humans.[1].

This is not to understate the technical difficulties of making a program work correctly. But the cognitive demands of creating a program (see Green & Petre, 1996, for a comprehensive summary) have already been much studied. The inherent technical and cognitive difficulties have tended to some extent to overshadow the social ones, yet other people play a significant part in the tasks of professional programmers.

[1]This was paraphrased by Scott (2009, p.10) as ”Programming is the art of telling another human being what one wants the computer to do”, a phrase widely mis-attributed as a direct quotation from Knuth (e.g., Tekir, 2012, p.33).

It is therefore no surprise that colleagues' performance is a factor in the happiness of software developers. Among the top 10 causes of unhappiness identified by Graziotin et al. (2017) in a survey of over 1300 developers (75% of them professionals), issues with colleagues featured several times: bad code quality and coding practice (no.3), under-performing colleague (no.4), and bad decision making (no.8). These are all issues where a colleague's behaviour has the potential to make one's own job harder. They may therefore also contribute to the number one cause of unhappiness, "being stuck in problem solving". Personal experience as a programmer suggests that problem solving can be very rewarding, but not when it involves being stuck in figuring out what a poor quality piece of code is doing.

2.7 Improving software development

The big problems to be solved — software delivered late, over budget, with serious defects, or never delivered at all — have encouraged hopeful grasping of big ideas that purport to solve them. No measures have entirely lived up to the hype surrounding them. F. Brooks (1987) suggested that the nature of software development (its complexity, the constraints of conforming to predefined interfaces, ongoing change over its lifespan and the difficulty of visualising its invisible structure) make it unlikely that there will be any "silver bullets". A seemingly obvious place to look for some modest solutions is in the education that programmers receive (§2.7.1), but there is a vast amount of complex technical material to cover and perhaps the basics need to be mastered to some degree before considering how to go about applying the technical skills that have been learned. This shifts attention to the workplace, where many approaches to the processes of software development have been tried (§2.7.2). There is also advice about the practices an individual can usefully follow (§2.7.3), but often this emphasises the technical and neglects the social. There continue to be attempts to find "silver bullets", but rather fewer attempts to understand the behaviour of those who are expected to use them.

2.7.1 Programmer education

Programmer education has a lot of ground to cover, and in teaching the application of its theoretical knowledge to practical skills it tends to emphasise the production of code from scratch: the transformation of requirements into new, working software, but not the transformation of existing software to meet new requirements. The

BCS[2] guidelines on course accreditation (BCS, 2015) do not require or even suggest that undergraduates should experience looking at other people's code. A BCS core requirement for honours programmes is that graduates should have "the ability to specify, design or construct computer-based systems" but only in the additional requirements for chartered professional status (CITP, CEng and CSci) are "deploy, verify and maintain" added to this list (BCS, 2015, p.10). Curriculum guidance from ACM[3]/IEEE[4] is better in recognising that modifying existing programs "can be more like real-world experience" than constructing a whole program from scratch (ACM and IEEE Computer Society, 2013, p.41). The ACM/IEEE Computer Science guidelines (ACM and IEEE Computer Society, 2013) explicitly include knowledge of "Software Evolution" (including program comprehension, code search and simple refactoring) as a core concept, while the Software Engineering guidelines go further by explicitly including "Working with legacy systems" (ACM and IEEE Computer Society, 2015, p.35).

Anything from 40% (Glass, 2003) to over 80% (Verhoef, 2000) of software development costs represent some form of work with existing code, so the ability to do so is a core skill that is essential in industry. But even when a professional body validating undergraduate courses does require software evolution as a core part of the curriculum, it can only be one of many such essential elements. An undergraduate's exposure to examples which would help them to understand how their task can indirectly be made harder or easier by other programmers' earlier decisions is limited; as a participant in the Exploratory Study (Chapter 5) observed, "When you're learning programming you don't tend to get the opportunity to look at a load of existing code, you don't get that impact." Developing this skill in any detail is a postgraduate topic, as for example in a "Software Engineering for Industry" course (Imperial College, 2017) teaching tools and techniques for working with large, existing software systems.

Teaching that includes exposure to existing software may do more than help students to deal effectively with the consequences of decisions taken in the past. In experiencing the consequences themselves, there is also potential for them to become more aware of the possible significance to others of their own actions as programmers.

[2]BCS, The Chartered Institute for IT
[3]Association for Computing Machinery
[4]Institute of Electrical and Electronics Engineers

2.7.2 Software process improvement

Over the years many ideas have been proposed for doing programming better; see Boehm (2006) for an extensive history from the 1950s through to the 21st century. To cite just a few examples of tools, processes and methods the industry has seen: the Waterfall process (Royce, 1970), Computer Aided Software Engineering (CASE) tools designed to support new analysis and design methodologies (Avison & Fitzgerald, 2006), and Agile software development (Agile Alliance, 2001a). The ideas address many different aspects of the problems of software development; doing it “better” can be defined in terms of meeting deadlines, faster debugging, fewer errors in the code, customer satisfaction and so on.

The criteria against which the performance of such interventions can be assessed are not easy to measure or compare in the absence of standardised benchmarks. Programmer productivity, for example, is an elusive and ill-defined concept. Bossavit (2012) noted the variation in the unit of measurement between studies by prominent authors who claimed to have found a tenfold difference between the most and least productive programmers: time to complete a task, lines of code per hour, or a manager’s evaluation. There was similar variation in the activity measured: experimental tasks to write code, experimental tasks to debug it, or case studies of real projects.

To some extent the importance of the people in the software development process has become more widely recognised through the Agile Manifesto’s value of “Individuals and interactions over processes and tools” (Agile Alliance, 2001a), but communication skills and people skills remain somewhat overlooked (Ahmed, Capretz, Bouktif, & Campbell, 2013). Such skills, often described as “soft skills” (a contentious term which can appear vague and dismissive; Watkin, 2017, presents some alternatives), are not automatically created merely by the adoption of a methodology that gives more emphasis to people than before. Indeed Agile methods intended to foster communication can even be counter-productive. Ghobadi and Mathiassen (2016) analysed interviews with project managers, developers, testers and user representatives at two companies with Agile software teams to identify perceived barriers to knowledge sharing. These they classified into seven categories: three related to teams (diversity; perceptions; capabilities, both technical and social) and four to projects (communication; organization; task setting; technology). Team perceptions of the Agile methodology appear to have created barriers through perceived restrictions. For example, one participant observed: ”If they don’t have a story written it seems to inhibit their initiative to actually be proactive and do something about it …I’d say why this wasn’t done, why wasn’t this raised, and someone goes: we didn't have a story". Some developers also feel anxious and intimidated about speaking in the intensive environment of daily stand-up meetings (Conboy, Coyle, Wang, & Pikkarainen, 2011), to the extent that they may actually dread it.

2.7.3 Orthogonal technical and social practices

Discounting obstacles that novices might introduce through their own lack of experience (e.g., violating expectations; Hansen et al., 2013), difficulties in software development can also be caused by the way some other, experienced developer has tackled the task. While almost all solutions are eventually successful in the technical goal of getting the computer to do something, they nonetheless differ in how easy it is for peers to understand what the computer is meant to do, and why, and how. This is why refactoring (changing the internal structure of software without changing its external behaviour) exists as a concept. "The compiler doesn't care whether the code is ugly or clean. But when we change the system, there is a human involved. And humans do care. A poorly designed system is hard to change" (Fowler, 1999).

Fowler describes the characteristics that indicate when to refactor as "bad smells". Many people can write working software, but not all manage to make it "smell good". Experience and technical ability do not necessarily mean expertise in achieving this "team-friendly" quality, and it is entirely possible to meet the technical goal without it. The number of identifiers and number of characters per line, for instance, may be significant for readability (Buse & Weimer, 2010), but neither plays any part in the execution of the program. There are two orthogonal skills in programming. One is the cognitive skill of analysing a problem, understanding how it can be solved in computable steps and explaining those steps to a computer, a syntactically pedantic audience where grammar is all-important and vocabulary is irrelevant but for a handful of keywords and a few orthographic restrictions. Giving these instructions correctly achieves the most commonly monitored and valued result for the business: delivered software. Less obvious, and less easy to measure, is the social dimension: the impact of that work on others. The decisions made about how to build it, its internal structure, the libraries to use, the interface it presents to other components, its tests and testability; all these and more have an effect on other members of the project and on future maintainers.

Habits and attitudes are important, yet are often treated as secondary in advice for programmers if mentioned at all. Maguire (1993), for example, referred to them only after seven chapters of explaining technical practices, as "the other necessary ingredient" to preventing and detecting bugs. This imbalance is not always the case; Hunt and Thomas (2000) are a fine example of treating both with equal importance and including them throughout the book as part of a more holistic approach to good practice. But historically and currently, there has been more focus on the role that tools or methods can play in the task of delivering working programs.

2.8 Conclusion

Characteristics which make instructions to the computer also serve well as an explanation to a human being have not proved particularly amenable to metrics. Measurements of the code cannot in any case bring together the whole breadth of things that a programmer needs to understand to work on an existing system (e.g., how it is built, tested, tracked, deployed etc). Nor can the processes under which software is developed provide a "silver bullet", though some have moved closer to addressing the importance not just of the code but of communication.

The motivation of the research reported in this thesis is to address the impact of the social dimension more directly. The research emphasises the importance of the fact that most of the time, a programmer's job involves working on a system that someone else — perhaps many others over time — has already had a hand in. It investigates the impact of the behaviour of that "someone else" by learning from software developers themselves what they find most and least helpful in the way their peers approach the job. This understanding is then applied in the design of an intervention to help promote positive behaviours. Social factors can make doing a good job more demanding than merely writing working code, so the breadth of all activities involved in a software developer's work is examined. But all activities are scrutinised with the same focused lens: the perceptions of the professionals themselves.

Chapter 3

Research methodology

3.1 Introduction

The purpose of this research is to learn from programmers themselves about how others' behaviour impacts upon them, and to give practical application to what is learned. The research questions (reiterated later in this chapter for convenience) therefore ask what matters to programmers most in the behaviour of their peers and whether this understanding can be used to help improve software engineering practice.

This chapter explains the philosophy that underpins how the research questions are addressed (§3.2). RQ1 is investigated by an exploratory study to collect programmer perceptions (§3.3). This feeds into an Evaluation Study (§3.4) that tests how the findings can be applied in practice by professional software developers (RQ2). These sections explain the considerations in designing the methodology for the project; a complete outline is given in Table 2.

Programmer perceptions (Exploratory Study)

Information sought	Methods considered	Chosen method(s)
Participant demographics: overall picture of career experience	Questionnaire	Online questionnaire, anonymous. Minimum 5 years' career experience
Elicitation: Perceptions of the impact of peer behaviours	Case studies Ethnography Participant observation Questionnaire Interviews Focus group Card sorting Q sort	Interviews by researcher, using: • Open discussion • Card sorting task • Structured discussion
Analysis of common themes	Template analysis Interpretative phenomenological analysis (IPA) Grounded theory Matrix analysis	Template analysis, using card sorting material for a priori themes

Practical application (Evaluation Study)

Information sought	Methods considered	Chosen method(s)
Working environment	Observation Questionnaire Interviews Focus group	Tuckman questionnaire, online, prior to discussion
Delivery of material and facilitation of discussion	Presentation Written report Focus group Workshop Activities Props	Workshop with text-based activities and props
Evaluation of delivery	Observation Discussion Questionnaire	Observation Questionnaire
Evaluation of usefulness	Observation Discussion Questionnaire	Questionnaire

Table 2: Methodology considerations

3.2 Research philosophy

It is hard to measure objectively the extent to which the actions of their peers can help or hinder programmers' work, nor is there yet a comprehensive catalogue of the behaviours which are relevant. Understanding the impact of those behaviours therefore calls for eliciting the subjective perceptions of those who are affected: experienced software developers. It is an inherent ontological assumption of the research that they are able to articulate how peer behaviours affect them; personal experience of programmers' outbursts of frustration suggests that, at least for negative behaviours, they are indeed conscious of the impact these have.

The research adopts an element of post-positivist philosophy in that the concept of peer behaviours is seen as an "independent reality" (Gray, 2013, p.23) rather than, for instance, a construction of each individual participant. It is not so much the individuals' personal interpretations of their experiences that are sought as the commonalities between their accounts. These point towards recognisable patterns of behaviour that occur in many workplaces, with similar effects.

However the research is also strongly influenced by its goal of practical outcome rather than by a particular epistemology. This is a pragmatist philosophy that considers "the research question to be more important than either the method ...or the worldview that is supposed to underlie the method" (Tashakkori & Teddlie, 1998, p.21). In other words, pragmatism chooses an approach not because it conforms to a particular ideology but because it fits the purpose of the research. Creswell (2013, p.6) described it as "problem-centered"; it uses whichever data offers the best understanding of the problem, be that quantitative, qualitative or a mixed methods approach which enables the problem to be analysed from different angles. Bryant (2004), for example, took such an approach in proposing a methodology for studying behaviour in the software development practice of Extreme Programming, and Di Penta and Tamburri (2017, p.499) presented the merits of using qualitative methods to address "why- and how- type questions" to explain quantitative findings about human behaviour such as "so-called 'bad practices' or human/organizational barriers" in software engineering research.

The Exploratory Sequential mixed methods approach chosen for this research is outlined in Figure 1 and the methods for each phase explained in §3.3 and §3.4.

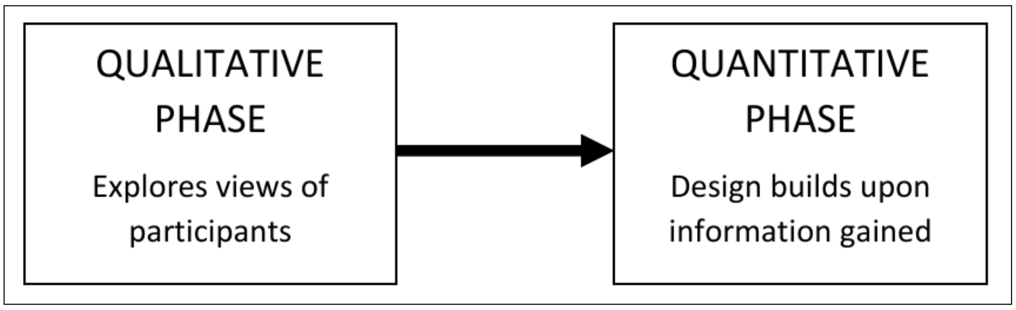

QUALITATIVE PHASE
Explores views of participants
QUANTITATIVE PHASE
Design builds upon information gained

Figure 1: Exploratory sequential mixed methods approach

3.3 Exploratory Study: Eliciting programmer perceptions of peer behaviour

Research Question 1: What is the perception of experienced programmers on the peer behaviours that help or hinder them in their own tasks?

Objectives:

1. To collect opinions from experienced programmers working in a variety of application domains about the helps and hindrances that affect them in the breadth of their everyday work.

2. To identify common themes in their responses.

This section outlines the design considerations for a study to address the objectives of Research Question 1, repeated here for convenience. The resulting design is a study in which experienced professional programmers are interviewed about real-life experiences in their work and asked to reflect on how the work is affected by the behaviours of their peers. This study, referred to throughout as the Exploratory Study, is described in Chapters 4 and 5.

3.3.1 Participant demographics

Research Question 1 seeks to identify patterns in the opinions of experienced programmers. New graduates do not yet have all the skills needed to work effectively in a commercial software development team (Radermacher, Walia, & Knudson, 2014; Johnson & Senges, 2010), so to eliminate any issues that are specific to novices the participants need to have enough experience to have developed a degree of “developer fluency”. This is a term used by Zhou and Mockus (2010), who suggested that fluency continues to grow for at least the first three years if taking into account the increasing difficulty of the tasks that can be assigned to novice developers as they gain experience. Job advertisements for "experienced" programmers do not always specify how much experience; of those that do, some demand at least five years while others ask for as little as two. All the managers questioned by Zhou and Mockus (2010) would be willing to commit customer-critical and mentoring tasks to developers with five years' experience, so this is chosen as the criterion for the study. By this stage of their careers developers are likely to have encountered different colleagues' styles and be familiar with tackling significant tasks involving existing code. To gain an overall picture of participants' career experience, a questionnaire will be used.

3.3.2 Elicitation

Observation or introspection

Identifying patterns in the opinions of experienced programmers calls for the research to include a breadth of programmer perspectives across a variety of types of software application. It precludes detailed case studies or observational approaches such as ethnography or participant observation (Symon & Cassell, 2004, chapters 26, 25 & 13), which are particularly suited to gathering rich, in-depth data in the context of a particular environment but not appropriate in this research. Here it is not the setting but the individual's perceptions that are important, synthesised from any and all of the environments they have encountered during their whole career. Seeking this "long view" over years of experience also has practical implications. Events happening during a workplace observation, for instance, would afford an excellent opportunity to record in great detail the nature of an issue and its consequences, but the data would be limited to those occasional incidents. Such an approach would be more feasible as part of an extended participant observation, but even then would only yield data for one location and a limited number of projects.

If the issues cannot be observed in person, an alternative is to ask programmers themselves to reflect on circumstances in which they perceive that they have been helped or hindered in their work by others' behaviour. A questionnaire would be a fast way to canvas the opinions of a large number of respondents, but structured questions cannot capture the richness of data required (Coolican, 2014, p.170). This research seeks not only to catalogue the behaviours which are perceived as having most impact (whether positive or negative), but also to understand why this is the case. A questionnaire could investigate the former with a simple Likert scale, but is not well suited to collecting the latter. Even if respondents took the time to a enter a sufficiently detailed account in a text box asking them to explain the impact of important behaviours, there would be no opportunity to seek clarification or pursue interesting aspects of their response in more detail.

Rich and detailed introspection

To fully explore the reasons why as well as cataloguing which peer behaviours have most impact requires a more personal approach than a questionnaire — a one-to-one interview in which a rich and frank account can be elicited (Gray, 2013, pp.382–388). The interviews need to remain grounded in the everyday details of participants' workplace experience; without those details, the account risks becoming generalised and over-simplified rather than explicitly explaining the symptoms and their consequences, much as good practice maxims like "seek to write clear, self-documenting code" (Goodliffe, 2006) are insufficient without the accompanying explanation of the principles behind them. It is helpful in this respect if participants can talk to a fellow specialist for whom they do not need to interpret (and therefore simplify) their very technical world, but can give detailed accounts using the vocabulary of the profession. Coolican (2014, p.179) explains the benefits of an interviewer understanding the normal language mode of the community; it prevents misunderstanding, and interviewees are "at their most comfortable and fluent" when using it.

In this research, understanding the language of the profession is also a prerequisite for the ongoing data analysis that occurs during the interview process:

> "[T]he interviewer is not just listening and recording. They are or should be attempting to establish the full meaning of the respondent's account from their point of view. For this reason the qualitative interviewer is not a passive recipient of information but needs to be active in checking what the interviewee is saying, watching for inconsistencies, encouraging fuller detail where a story is incomplete and generally keeping the research aims in mind throughout the process." (Coolican, 2014, p.177)

In interviews which seek detailed accounts of a technical specialism, a lack of subject knowledge would make such checking and follow up questions difficult. It would also affect the story that participants tell. Interviewees respond differently depending on who the interviewer is; this includes social categories such as age and gender (Miller

& Glassner, 2004) but also their role (Coolican, 2014, p.171). The researcher has the relevant experience to interview participants about their work, although this comes with the risk of her own experience influencing their responses by the form in which questions are asked (see §3.5). Interviews by the researcher herself are therefore chosen as the appropriate means to elicit rich and reflective accounts of participants' workplace experiences, with awareness of this risk reflected in their design.

Facilitating recall

The interview will use what Tashakkori and Teddlie (1998) described as a 'funnel interview': starting with open-ended questions before moving on to a more structured format. The interview will thus be designed to first capture the thoughts that spring freely to participants' minds and then move on to structured techniques to facilitate further recall.

Just as observational methods would be limited to capturing only the situations that happen to arise, interview techniques can only collect what participants actually recount. Cognitive psychology has shown that recent events are more accessible than older ones (Baddeley, Eysenck, & Anderson, 2015). Autobiographical recollection is also better for emotional events, whether positive or negative, and unusual incidents that stand out (Bower & Forgas, 2000). Cued recall — asking people to recall a memory associated with a prompt, such as a simple word or a particular period of their life — has long been understood by cognitive psychology as an effective means of probing for memories (Baddeley et al., 2015).

There is a problem, too, with implicit knowledge (Berry, 1987): things which are so "obvious" that they are not usually consciously articulated. Aids are therefore useful to help prompt participants to recall a wide range of their experiences at work. Card sorting offers a means of tapping into implicit knowledge; it has been used, for example, to explore mental constructions of programming concepts (e.g., Sanders et al., 2005). It is also a tool within the computing industry itself, where website builders use it to understand how users categorise information so that the website can be structured in a way that makes sense to them (Usability.gov, 2013). It is an engaging and informal process which also has the advantage of helping to "solidify" the experience (Chope, 2015); physical artefacts such as cards do not just carry content but also facilitate the communication process itself (Sharp, Robinson, & Petre, 2009). A card sorting task is therefore included as a part of the interview procedure to cue participants to reflect on the breadth of their experience and articulate their thoughts about it. This has similarities with a Q sort in eliciting first-person perspectives by asking participants to rank a set of items in order of personal significance (Watts & Stenner, 2012) but differs in the use that these are put to. Q methodology applies a factor analysis to reduce the list of items to a smaller number of factors and thus recognise individuals who share a similar perspective. In this research, the goal of the card topics is to prompt recall.

The process of ranking of the cards, although it will also produce quantitative data on what matters, is intended as an aid to help participants reflect deeply on their career experiences and elaborate on why the topics presented have an impact. Interpretative statistics would be inappropriate for a method designed as a means to provoke thought and discussion rather than as a quantitative measurement. Descriptive statistics, however, will be helpful to illustrate the trends in the sorting task and the topics which dominated the subsequent discussion.

3.3.3 Analysis of common themes

The methods selected to elicit programmers' perceptions about peer behaviour are chosen to produce rich data which addresses the full breadth of tasks involved in their work. The approach taken not only seeks to find out what matters to them, but to delve more deeply to understand why.

Some of their responses might be anticipated. Others might be unforeseen or express different priorities to those emphasised by the field of programming good practice. Indeed it is important that the opinions expressed should not be constrained by the researcher's own experiences or training as a programmer. This research calls for an approach to analysis which can accommodate both anticipated and new themes. Template analysis (King, 2012) is chosen for thematic analysis of the interview data because it flexibly accommodates both an open discussion, in which participants can raise any topic, and themes anticipated prior to the study. "Bottom up" approaches such as interpretative phenomenological analysis (IPA) and grounded theory avoid any prior theorising, preferring themes to emerge only from the data. Matrix analysis and other "top down" analyses, in contrast, define themes from a theoretical basis and apply these to the data (King, 2012). Template analysis uses such a priori themes more provisionally: as coding progresses they are refined and new codes are added, as needed. Interviews can be coded until the data does not give any new information or possible codes.

Template analysis

Template analysis begins with preliminary coding of a subset of interview transcripts. Material relevant to the research question is coded either to one of the a priori codes (refined as necessary) or to a new code. The end of this stage is reached when the coding no longer produces distinctly different new themes. The modified a priori codes and the new codes are then organised under a smaller number of higher order codes describing broader themes, creating the initial template.

The initial template is then developed by applying it to remaining transcripts, modifying the template when text relevant to the research question does not fit within an existing theme. Significant changes to a template require re-coding of earlier work. This development of the template continues until no further new and relevant information is emerging (i.e. saturation is reached and reading of the remaining transcripts yields no new material that could not be coded to the existing template). The template at the end of the process is the final template, a hierarchical model of the themes.

3.4 Evaluation Study: Testing practical applications

Research Question 2: Can experienced programmers' accounts of the peer behaviours that most help or hinder them in their own tasks be used to help others in their practice?

Objectives:

1. To present the common "helps and hindrances" themes collected in the first phase to groups of professional software developers for discussion.

2. To evaluate the usefulness of such discussion.

3. To identify other avenues through which the material could be useful.

The outputs from the Exploratory Study are expected to be a catalogue of peer behaviours that affect others' progress, and an account of why these behaviours are significant. "Good practice" is good for a variety of different reasons (e.g. accuracy; readability; performance), but the Exploratory Study outputs will identify practice that is "good" for the specific reason that it makes someone else's job easier. Such findings drawn from the careers of experienced programmers are not an idealised theoretical picture but a pragmatic model of the ways of behaving that affect the progress of real workplace activities in the field. Research Question 2, repeated here for convenience, asks whether this model can be used in a practical framework to help others in their practice. This section outlines the considerations taken into account in the design of a study to address the objectives of Research Question 2. The resulting design is a study in which groups of professional programmers participate in a workshop to discuss software development practices from the perspective of the impact they have. This study, referred to throughout as the Evaluation Study, is described in Chapters 6 and 7.

3.4.1 Target audience

Professional programmers range from recent graduates who have basic technical skills but have not yet achieved the "developer fluency" described by Zhou and Mockus (2010) to experts looking, as experts do ("Experts keep learning"; Petre et al., 2016), to refine their practice. If the material used in the Evaluation Study is to have a practical application in the profession, mixed groups of programmers are the audience for whom it needs to have value as a form of continuing professional development (CPD). Workplace teams with a mix of experience will therefore be the participants.

There is also a pragmatic consideration in choosing an application designed for continuing professional development. While the materials may also have practical value for recruitment or appraisal, testing these would be dependent on the timing of company processes and therefore difficult to complete within a limited timescale. Introducing the material in a discussion format creates the opportunity for participants to see it for themselves and reach their own conclusions about other potential uses. Rather than try to evaluate multiple different uses within this research, the Evaluation Study will draw on the expertise of participants to suggest how else it might help them.

3.4.2 Working environment

This target audience includes professional software developers as a whole, whereas the Exploratory Study is likely to have attracted those with a strong interest in continuing professional development in general or this research in particular. They will also be talking in a group rather than one-to-one. A format designed to encourage open and constructive reflection on professional practice is unlikely to succeed in an environment where communication is already strained or the team culture difficult. Cockburn (2004) referred to a property of Personal Safety: "being able to speak when something is bothering you without fear of reprisal". In the absence of an instrument specifically to measure Personal Safety (see Chapter 7 for later reflection on this), a proxy measurement of team environment using a well-established instrument will be needed. The selection criteria for this team environment measure are that it should not be about the individual characteristics that individuals bring to their work

Advice has been sought from a management training consultant with accreditation for MBTI and Belbin testing, asking about instruments he finds useful in getting a picture of the state of a team environment: how much they collaborate, whether they communicate, how they make decisions that could affect others. The requirements were discussed (M. Mereu, personal communication, June 24, 2016) and the Tuckman Model of Stages of Team Development (forming, storming, norming and performing; Tuckman, 1965) was identified as a suitable candidate. This model meets the criterion for a measure of the current state of a team environment and it is in practical use. The consultant reports finding that it can bring "great insights" and it is described by Human Resources at MIT as "a helpful framework for recognizing a team's behavioral patterns" (Stein, n.d.). It is also cited in a software engineering context as a theoretical framework for understanding agile teams (Morris, 2017; Rowley & Lange, 2007; Lee, 2008; Kuhrmann & Münch, 2016).

A questionnaire based on the Tuckman model will be used prior to the discussion to identify the stage of the model that a participating team is currently operating in (Clark, 2016).

3.4.3 Delivery of material

Personal experience suggests that programmers do not have time to stop and discuss in depth when they are busy firefighting the problems in front of them. Reflection may occur afterwards, but tends to focus on processes — what went well, what didn't — and not on broader patterns of behaviour that can contribute to some of the symptoms. This is addressed further in a discussion of Agile retrospectives in Chapter 7. As an experienced programmer commented during a pilot of the materials for the Exploratory Study "I'm trying to think what makes programming hard. And it's not stuff that people talk about."

The Evaluation Study seeks to help participants to reflect on "what makes programming hard" by considering impactful behaviours, as catalogued in the Exploratory Study. To develop a shared understanding it is important that participants share how and why certain behaviours have a positive or negative impact, rather than simply declaring things as a good or bad practice. There is no suggestion here, for example, that the unhelpful practices identified by the Exploratory Study necessarily equate to a lack of competence. There are multiple possible solutions to a software task, all leading to a successful outcome in the common metric of delivering working code. But collateral damage inflicted on others by a particular approach is not necessarily apparent to the perpetrator. The aim is therefore to help all participants see the consequences of actions through the eyes of their peers. This rules out delivering the findings from the Exploratory Study in the form of a presentation or written report, which would only give them the researcher's perspective. To maximise the relevance of the activity they will talk among themselves and have the choice of which topics to select for discussion from those that emerge as themes from the Exploratory Study. This will allow them to focus on sharing their thoughts about behaviours having positive or negative impact locally rather than general tenets of good or bad practice.

Creating a space for programmers to step back, reflect and share their reflections calls for Cockburn's (2004) property of Personal Safety so that the discussion can be a free and frank one. A card sort exercise such as that in the Exploratory Study has the advantages of both offering a wide variety of topics and explicitly putting those topics on the table for consideration. A topic on a pre-printed card represents something that is not just personal opinion but legitimised and depersonalised by being included in topics originating from multiple companies. Cards will therefore be used as a medium for offering the topics for discussion, with an explanation of their origin, and the introduction of new topics by participants will not be allowed.

The use of cards focuses the topics for discussion. To facilitate the actual discussion, influences will be taken from focus group techniques. "Focus groups work best for topics people could talk about to each other in their everyday lives — but don't." (Macnaghten & Myers, 2004, p.65) This makes them a promising approach for discussion whose purpose is to help participants step back from the immediate demands of problem solving and communicate their opinions on matters not typically discussed. Focus groups are also useful for "exploring the energy and time-consuming nuisances that prevent workers from doing their assigned work to the fullest capacity" (Elvins, 1985, p.481), exactly the kind of exploration the Evaluation Study seeks to facilitate by group discussion of the offered topics. Since the goal is to elicit a common understanding among the participants themselves, rather than information for a researcher, the format for the study will be a workshop rather than a focus group per se and may deviate from focus group practice accordingly. The audience for the output differs but in essence the goal of eliciting opinions is the same, and focus group techniques will inform that process.

3.4.4 Facilitation of discussion

Activities

Achieving meaningful discussion in a workshop calls for searching, reflective engagement from the participants. Making them “do” something promotes a more in-depth discussion as an activity engages them more actively with the questions, as well as being potentially more enjoyable (Colucci, 2007). In particular, activity-based discussion is recommended for quiet, reflective participants who benefit from time to reflect before speaking (Colucci, 2007). Given that the material in this research included things participants would prefer their colleagues to do differently, Colucci’s advice that activities can make sensitive topics seem less threatening is also apt:

> “[E]xercises ...accomplish their role best if the moderator ...invites participants to describe their answers more in depth, provide more detail, apply them to a real situation, and express agreement or disagreement with other participants’ answers.” (Colucci, 2007, p.1430)

These descriptions crystallise what is needed for the Evaluation Study, which will therefore use activity-oriented questions to promote discussion. Since the workshop is not an end in itself but an exercise to foster communication and behaviour change, one final feature described by Colucci (2007, p.1431) is apposite: activities are “likely to be remembered as positive experience and recommended to others”.

Since card materials are to be used to introduce topics in a standard fashion, an obvious choice is to build an activity around the cards. Alternative formats to textual cards could be considered, with photo elicitation the most likely candidate as craft or construction activities do not lend themselves either to the question or to the audience. Some participants might find such activities ”a bit weird” (Colucci, 2007, p.1432), and for this audience (see Cruz, da Silva, & Capretz, 2015, for a review of the predominant personality traits) it is prudent not to push it too far and risk losing their engagement by asking them to undertake activities that they might find uncomfortable and indirect.

Photo elicitation is considered because it offers scope to represent the topics in something resembling their normal context, e.g., an example of a poor bug report shown in the format of the participants' issue tracking system. However most of the topics would be difficult to represent clearly and unambiguously with an image. For example, how could automation or dogmatism be clearly depicted? Even concepts that might be represented using screen shots (e.g., an example of an uninformative bug report displayed in an issue-tracking product) would be more text-heavy. A unfamiliar screen layout belonging to some tool not in the locally-used toolset would be a counterproductive distraction. Images have potential to convey some ideas faster and less laboriously that reading text, but the topics to be discussed here do not lend themselves to instantly recognisable images. A text format will therefore be used.

Use of a facilitator

Because the objective is ultimately to provide a technique of practical use to software developers, the possibility of making it something which they could run for themselves must be considered. This would allow the materials to be used for group discussion at any time without the need for a specialist facilitator to conduct the session. Written instructions alone are unsuitable for a self-directed workshop due to the risk of readers only skimming through them. Video-recorded instructions, although not guaranteeing full attention, would at least dictate the pace and ensure that all the instructions through to the end are covered. But a self-directed workshop at this stage would be attempting to test two independent characteristics at once: the usefulness of the workshop format and the feasibility of companies using it independently. Stewart and Shamdasani (2015) suggested that in focus groups, a good facilitator is needed to gently and unobtrusively enable rich insights to emerge and to manage the group. In this research those insights are to be elicited for the benefit of the participants rather than the researcher, which perhaps calls all the more for a well-managed discussion. The workshop therefore will be tested with the researcher as a facilitator, investigating whether it serves its purpose in promoting discussion when conducted as designed. If successful as a format with a facilitator, the possibility of self-directed, independent use can be investigated as a future step to support wider adoption. A question about the role of the facilitator will be included in the participant feedback to gauge the interest in such an approach.

Participative safety

For the discussion to allow participants to reflect honestly and constructively, respectfully and without fear, the environment needs to be one of participative safety (Anderson & West, 1998). It is important that everyone is able to contribute and that they are listened to. As well as obvious concerns for inclusivity and courtesy, this is also an element of making the discussion a practical application of the research, creating output that is usable by participating developers because the content of the discussion is their own, joint construction.

This goal is supported in part by explaining that the topics offered for discussion come from the Exploratory Study. Choosing a card is therefore not merely personal opinion but a reference to the collective wisdom of many, unknown peers in the industry. The expectations of participant behaviour will also be explicitly set out in ground rules. In order to keep the stating of rules from being awkward or didactic, props will be used as a visual cue to explain them to participants in a lighthearted manner. Citizens Advice (2015) has proposed the use of props as an icebreaker for group discussions, which suggests that they can be effective as a means of relieving any tension. Although Stewart and Shamdasani (2015) recommended keeping any props hidden until needed, their advice does not apply to the other reason for their use in the Evaluation Study: the props will remain visible throughout as a three-dimensional reminder of the rules governing the discussion.

3.4.5 Evaluation of delivery

There are two aspects to measuring the workshop's success. Firstly, the procedure must run smoothly. If this criterion is not met it is unlikely that the workshop will fulfil the requirement expressed in RQ2 of being useful to its target audience. The researcher will observe the conduct of the workshop and make notes during and after it so that any necessary improvements to the procedure can be made. However the researcher's non-participant observations are only part of the picture, able to record such things as failure to follow the instructions but with no insight into how the procedure is experienced by the participants for whom it was designed. Participant feedback on the logistics will be collected, using a paper questionnaire immediately after the workshop questionnaire so that it can be completed before leaving while the details are still fresh in their minds. The procedure will be developed iteratively, using researcher observations and participant feedback to refine the procedure for the next workshop.

3.4.6 Evaluation of usefulness

The usefulness of the discussion to its participants is the key criterion. Evaluating this requires their feedback. Empirical measurement of longitudinal changes is not feasible because there are too many confounding variables at play to permit a controlled experiment, and no clear choice of a dependent variable. As discussed in Chapter 2, software metrics do not measure the Developer eXperience (DX).

The epistemological position of the research is that the impact of peer behaviours can be reported by programmers themselves; it is consistent with this approach to similarly trust programmers' insights into the impact of the workshop and ask them to report whether it was helpful to their work. Evaluation of the usefulness of the workshop will therefore be done by asking participants for their feedback immediately afterwards, using a paper questionnaire so that it can be completed before leaving. It will be structured so that questions about uses and benefits of the workshop can be answered with a simple yes or no which can be analysed quantitatively, but also invite further comment so that reasons for the answers can be understood.

Taking part in a workshop may be diverting and interesting at the time but it is of most value if it has some lasting influence. Participants will therefore be asked about the workshop again some time later. An interval of four working weeks will allow some opportunity for ideas from the workshop to take effect without being so long that it is difficult for participants to distinguish between specific effects of the workshop and other factors. Because the questions will be asked as a later follow-up they will be presented as an online questionnaire, again allowing for simple or more detailed answers.

The Evaluation Study will test a CPD application, but the participants may also have insights about other practical uses of the materials. The post-workshop questionnaires — both the immediate feedback and the later follow-up — will ask about other potential uses for the materials such as appraisals or recruitment.

3.5 Reflexivity

The researcher comes from the software developer community just as the participants do. This is invaluable for communicating effectively with them about their work (see §3.3.2) and participants in both the Exploratory Study and Evaluation Study will be informed of her background as a professional programmer to assure them of a knowledgeable audience for anything they choose to say about their work. But this background also comes with a risk of becoming involved as a software developer rather than a researcher. Design measures will be taken to minimise this risk but it is impossible to remove oneself from the process; reflexivity allows the influence of the researcher upon the research to be acknowledged (Coolican, 2014, pp.269–270). The reports of both studies will include the researcher's own reflections in order to make the role in the research of her own perspective as transparent as possible.

3.6 Conclusion

The research will comprise two sequential studies. The Exploratory Study will elicit experienced programmers' accounts of the peer behaviours that most help or hinder them in their own tasks. Its detailed design is set out in Chapter 4. The findings (Chapter 5) will inform the content of the Evaluation Study, which will test a method of using these accounts to help other programmers in their professional practice. Its detailed design is set out in Chapter 6.

Chapter 4

Exploratory study: Peer impact on software developers

4.1 Introduction

This chapter sets out the detailed design and conduct of the Exploratory Study introduced in Chapter 3. The study ran from April to November 2013, ethical approval having been granted in November 2012 by research ethics representatives of the School of Design, Engineering and Computing at Bournemouth University.

The purpose of the study was to address Research Question 1 (What is the perception of experienced programmers on the peer behaviours that help or hinder them in their own tasks?). It did so by collecting the perceptions of software developers on how they are affected in their work by the behaviour of their peers. Participants were asked via interview questions and a card sorting task to reflect on the impact caused by contemporaries or predecessors involved in working on the same software as them.

Industry domain	Scale of organisation
Avionics	UK SME, single location (<250 staff)
Equipment testing	UK Multinational SME (<250 staff)
Video equipment	UK enterprise, single location (<500 staff)
Finance	UK Multinational (<500 staff)
Enterprise software	Overseas Multinational (<10,000 staff)
Software development tools	Overseas Multinational (<40,000 staff)
Defence technology	UK Multinational (<100,000 staff)

Table 3: Profile of companies in Exploratory Study, in order of size

4.2 Participants

The participant profile was professional developers with at least five years' industry experience. Recruitment followed a stratified non-random sampling strategy (Tashakkori & Teddlie, 1998), drawing volunteers from known strata - different software industry domains - in the population. Seven contacts from a variety of domains were initially recruited via existing contacts and advertising with posters and lightning talks at two programmer conferences. These individuals enlisted the support of their companies and colleagues, resulting in the participation of programmers from a broad spectrum of domains and companies, as shown in Table 3. The locations of participating companies (Southern England, London and the Midlands) are excluded from the table to maintain anonymity.

All prospective participants were sent an information sheet (Appendix A) explaining what was involved, how the data would be used and the measures in place to protect their anonymity. A separate information sheet for their employers (Appendix B) set out the potential benefits of the research and addressed questions of commercial confidentiality and what was needed from the company. This proved extremely useful in securing approval. Participants were given the opportunity to ask questions about the research via the email address or phone number included on the information sheet, and also in person when they read the consent form (Appendix C).

All participants completed an online survey about their career history (Appendix D). For this they were given a unique participant number so that their name would not be attached to the data. All recordings and transcripts also used this id number. All data was treated with confidentiality and could be traced to individuals only by securely stored paper records: the consent forms they signed and a log of the assigned id numbers.

The participants were 28 professional computer programmers with between 4 and 33 years of professional experience (Mean = 16.5, standard deviation = 9.20); three who signed up fell slightly short of the 5 years' experience criterion. There was a very wide variety of programming languages which participants had used extensively; the most common are shown in Table 4. No participants had programmed in fewer than 3 different languages; most of them had used more. See Appendix E for a complete list of languages.

Programming language	Percentage of participants
C++	68
C	61
Python	61
Java	46
Javascript	36
C	32

Table 4: Languages which interview participants had used extensively

4.3 Materials

A card sorting task was designed as an aid to exploring participants' experiences, prompting them to consider not just reading and writing code but the wide range of activities involved in their job. Each card described a peer behaviour drawn from: advice on good practice in Hunt and Thomas (2000) and Henney (2010), the researcher's personal experience, and observations about contemporary events on the Twitter feeds of programming textbook authors Henney (2010) and Goodliffe (2006). Goodliffe (2012), for example, tweeted about the frustration of encountering "TO DO" notes denoting something unfinished in what should be finished code: "Suffering the tyranny of the 'TODO' comment. People: 'do' your 'todo's."

4.3.1 Criteria for card sort materials

The card topics were chosen to encompass a broad range of activities included in the day-to-day tasks of a professional software developer. To gauge whether this breadth was achieved the cards were classified according to the type of activity they addressed, based on the researcher's experience of the job. Three categories were identified:

- Indirect: activities in which the behaviour of peers is typically encountered in the form of an artefact or symptom (e.g., a module they wrote; the build breaking)

- Interactive: aspects of the work in which there is direct interaction with peers (e.g., asking or answering questions).

- Attitude: psychological characteristics of the approach to the work pervading both indirect and interactive encounters (e.g., attachment to a particular technique shapes not only artefacts but conversations).

The card topics concerning indirect encounters with peers' work can be further subdivided to show the breadth of software development tasks covered:

- Code: Use of programming language features within a unit of code.
- Structure: Distribution of code across discrete units.
- Testing: Writing and facilitating tests.
- Infrastructure: Activities associated with building the software.
- Tracking: Bug reporting and version control.

This model of software developer activities is illustrated in Figure 2. It illustrates the necessity of measures beyond metrics of the kind mentioned in §2.5 to form a holistic picture of Developer eXperience since only the 'code' and 'structure' subcategories include characteristics amenable to such metrics. Attitude is not shown as it applies throughout all elements of the model.

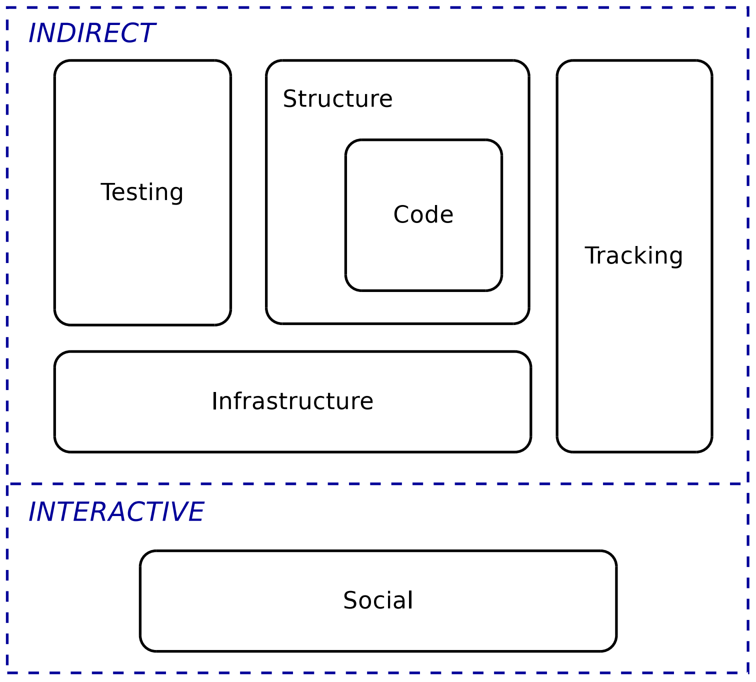


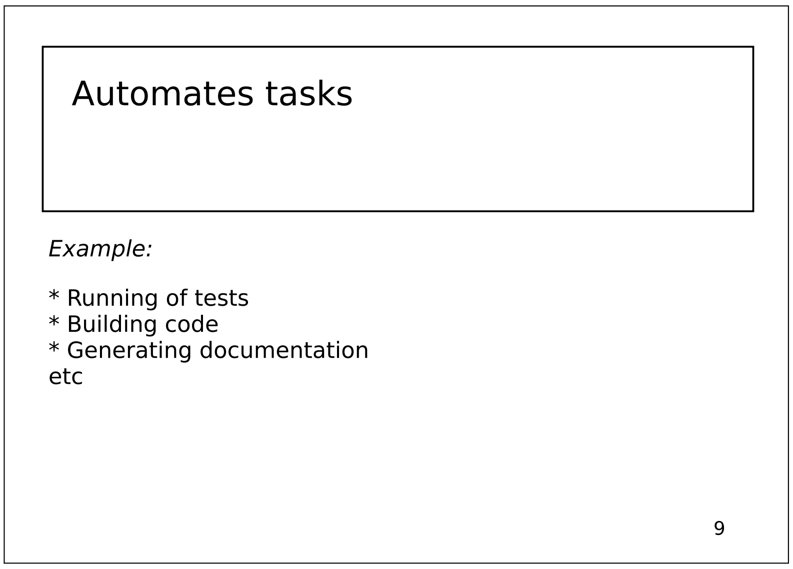
Figure 2: Aspects of software developer tasks represented on cards

4.3.2 Pilot study of card sorting task

To check the wording of the cards and the viability of the task, the card sorting was completed by two experienced programmers. As a result of researcher observations and their feedback the wording of some cards was clarified and the initial set of 88 cards was reduced by digesting groups of cards containing similar topics into a single card (e.g., a group of cards about commenting were replaced by a single card with a number of examples). This produced a final set of 54 cards (see Appendix F), which can be sorted within 20 to 30 minutes. For most topics it was uncontroversial whether the impact of the behaviour was positive or negative; the purpose was not to establish which behaviours are good or bad but to find out which of them exercise people most, and why. The wording was chosen such that there were similar numbers of "good" and "bad" (28 and 24 respectively; the pilot results suggested that the remaining 2 of the 54 topics were more a matter of personal preference).

4.3.3 Card format

The topics were printed on white A6 cards in the format shown in Figure 3. All cards included a box with a description of a possible peer behaviour. Some also contained examples (not intended to be exhaustive) of the behaviour. The id number of the card was included to facilitate recording of results. A sample card (card 0) was used for explaining this format to participants and was not included in the sort.



Automates tasks

Example:

* Running of tests
* Building code
* Generating documentation
etc

9

Figure 3: Format of cards for interview card sorting task

4.4 Procedure

All participants were interviewed one-to-one in a meeting room at their workplace. The researcher began by inviting them to think about fellow developers they had worked with, past or present, and describe how those people stood out for having made the participant's job harder or easier. Half an hour proved to be adequate for this open-ended discussion. Typically participants readily recalled a few memorable examples, recounted recent incidents or described scenarios which they find particularly vexing. The discussion continued while these accounts continued to come readily.

The next phase of the interview aimed to explore their experiences more comprehensively, cued by the card sorting task. Participants were asked to place each of the 54 cards according to the impact upon them of someone else exhibiting that behaviour. It was emphasised that the sole criterion was the impact on the participant's work and not whether the behaviour is generally considered "a good thing". The impact categories into which the cards should be sorted were printed onto colour-coded A4 cards. These were arranged in a line in order of impact from worst (on the left) to best.

- Bad. Noticeable impact. Makes my job harder/slower. (pink card)

- Bad. Slight impact. Makes my job a little harder/slower. (pink card)

- Neutral. Does not much affect my own tasks. (cream card)

- Good. Slight impact. Makes my job a little quicker/easier. (green card)

- Good. Noticeable impact. Makes my job quicker/easier. (green card)

The neutral category was explained and laid down first, followed by the "good — slight impact" and "good — noticeable impact" and finally the "bad — slight impact" and "bad — noticeable impact".

Participants were advised that examples, where included, were there only to help explain the card topic and were not intended to be an exhaustive list. They were asked to sort the cards according to "gut feeling" — to follow their immediate reaction rather than deliberate over the "it depends" criteria.

After the instructions were given, participants were asked before starting the task if there were any cards they would expect to see. This step was included so that further discussion was not limited to the a-priori topics which had been printed on the cards. If any suggestions were made at this stage, the wording of an appropriate card for the behaviour was worked out in collaboration with the participant, who then placed it according to its impact. The participant was then allowed to consider the printed cards. Some did this largely in silence, with occasional comments or requests for clarification, whilst others chose to spontaneously commentate on the cards as they went along; the approach was not prescribed.

After completing the task, participants were asked if there had been any topics they felt were missing. Again, any suggestions were added as a new card (for that interview only) which was sorted into a category. Inviting suggestions afterwards allowed scope to raise any new thoughts prompted by the exercise. By this stage participants could potentially have been constrained by the cards they had already seen, but the risk of serious omissions in the range of topics addressed by the cards was mitigated by also inviting suggestions beforehand.

Finally, the peer behaviours with most impact on the participant's work (i.e. those on the cards assigned to the two "noticeable impact" categories) were discussed to explore the nature of the impact — how and why the behaviour affects the participant.

4.5 Analysis

The interviews were transcribed and analysed following the template analysis process set out in §3.3.3. The initial coding reached an initial template after three transcripts, when the coding was no longer producing distinctly different new themes; this number suggests that there was much in common between the participants and the a priori card topics covered the breadth of a software developer's actitivies quite comprehensively. The earliest transcripts were recoded as the initial template evolved.

During the subsequent template development stage the changes were mostly refinements to existing codes (e.g., refining the description of a priori codes to reflect the scope of discussion that the associated cards inspired), with occasional new topics added. The template development process continued coding further transcripts until, after eight transcripts, no further new and relevant information was emerging. At this point the remaining transcripts were read through and yielded no new material that would not be coded with existing codes. The template at this point became the final template.

4.6 Conclusion

The interviews with experienced programmers proved effective in collecting rich accounts of their perceptions of peer behaviours. Participants spoke articulately and frankly about their experiences and indeed seemed to enjoy the opportunity to do so. As well as being invited to share the thoughts that came most readily to mind, participants also completed a card sorting task to elicit further recall. It was clear from the frequent signs of recognition that it did so. The common themes in participants' accounts of the human helps and hindrances in their working lives are represented by the final template, which is used to interpret the report of the findings in Chapter 5.

Chapter 5

Findings of the Exploratory Study

5.1 Introduction

This chapter sets out the results from the Exploratory Study described in Chapter 4, in which one-to-one interviews were conducted with experienced software developers to ascertain which of their peers' behaviours have most impact in helping or hindering their own work.

Both quantitative and qualitative data were collected. A card sorting task was used to promote recall and ensure breadth of discussion, but this also yielded figures about participants' responses and these are reported in §5.2. However the purpose of the interview was to collect rich accounts to understand how peer behaviours impact upon the participants' jobs. The themes that emerged are analysed in §5.3.

5.2 Card sort results

The cards which excited the strongest opinions were the ones which were then discussed further to elicit the nature of the impact. This section illustrates the most common concerns in those discussions, underpinning the themes which emerged (see §5.3).

Table 5 shows the distribution of all participants' card sorting decisions across the five categories. "Good. Noticeable impact", the chosen category in 27% of decisions, was the most frequently used. Much of this is accounted for by 15 card topics, just over a quarter of the entire set, which the majority of participants categorised in the same way. These are listed in Table 6 along with the percentage of participants who sorted them into this category. Overall, positive behaviours seem to have excited a more intense response than the negative behaviours, with only 6 of the cards placed in the "Bad. Noticeable impact" category by at least half of all participants (Table 7). Few topics were commonly considered "Neutral"; the only two which the majority felt did not much affect them are listed in Table 8.

Category	Decision frequency (%)
Bad. Noticeable impact	17
Bad Slight impact	21
Neutral	15
Good. Slight impact	21
Good. Noticeable impact	27

Table 5: Distribution of interview card sorting decisions across categories

%	Card text
89	Automates tasks
85	Includes accurate details of symptoms and how to reproduce the bug in their bug reports
82	Makes APIs easy to use correctly
82	Is good at helping others
78	Is willing to discuss suggestions about their code
75	Is willing to ask questions
67	Writes short, simple functions which perform a single task
67	Is always willing to consider that the bug may lie in their code
64	Uses code comments in ways that aid understanding
60	Finds out whether functionality is already available before writing their own implementation
57	Includes useful logging messages in their code
57	Keeps the flow of control easy to follow
53	Follows the DRY (Don't Repeat Yourself) principle
53	Is rigorous about deallocating allocated resources
50	Does not assume that a complex problem necessarily results in complex code

Table 6: Cards categorised "Good. Noticeable impact" by majority of participants

%	Card text
71	Tends to work in isolation
71	Fixes the symptoms without discovering the root cause of a bug
64	Espouses "one true way" of doing things
57	Is often the person who breaks the build
50	Chooses identifiers which are not succinct, meaningful and distinct
50	Tends to "own" code

Table 7: Cards categorised "Bad. Noticeable impact" by majority of participants

%	Card text
64	Follows formal methods to the letter
57	Includes brackets which are not demanded by the language's operator precedence

Table 8: Cards categorised "Neutral" by majority of participants

Contrary to the expectation inherent in the inclusion of the Code element in the model of software development covered by the cards (Figure 2), cards classified as “code” topics (i.e. dealing with fine-grained detail about the use of programming language features) rarely excited strong feelings. The few exceptions are: the positive cards about comments (a topic complicated by a wide spectrum of views on what actually does aid understanding), flow of control and freeing resources; and one negative card, poorly chosen identifiers. Instead it is social and attitudinal topics that dominate.

Several topics that were included on the cards were raised spontaneously by participants before they undertook the card sorting task. The few new cards added at the end tended to be topics at a detailed level that fell within the broader themes already on the cards, and therefore mostly contributed to a more detailed description of an existing theme rather than generating a new one. Few new cards were added at the end of the sorting task, participants often commenting that the existing set was a comprehensive one.

While its results are interesting in suggesting the relative importance of various software developer practices, the primary role of the card sort was not to catalogue practices by the degree to which they have positive or negative impact. The cards' purpose was to serve as cues to prompt participants to reflect on the whole breadth of their experiences across of their role as a developer and not focus solely on fine-grained details of code. Cards which provoked the strongest response in sorting were important because they then formed the basis for a much richer exploration of the topics which most resonated with each participant, reported in the following section.

5.3 Overview of interview themes

This section explains the themes that emerged from the interviews, whether in the initial discussion, the discussion structured by the choices made in the card sort, or when participants had the opportunity at the end to add any topics not yet covered.

Together with the card sort results, the qualitative analysis of the themes discussed by participants created a subtly different model of how programmers are affected by their peers than the one envisaged when creating the cards (Figure 2). The revised model (Figure 4) reflects not only the aspects of peer behaviour that participants focused on but also the reasons behind their significance. The elements of the revised model are used to structure this report of the interview themes represented in the final template, which is shown in full in Appendix G.

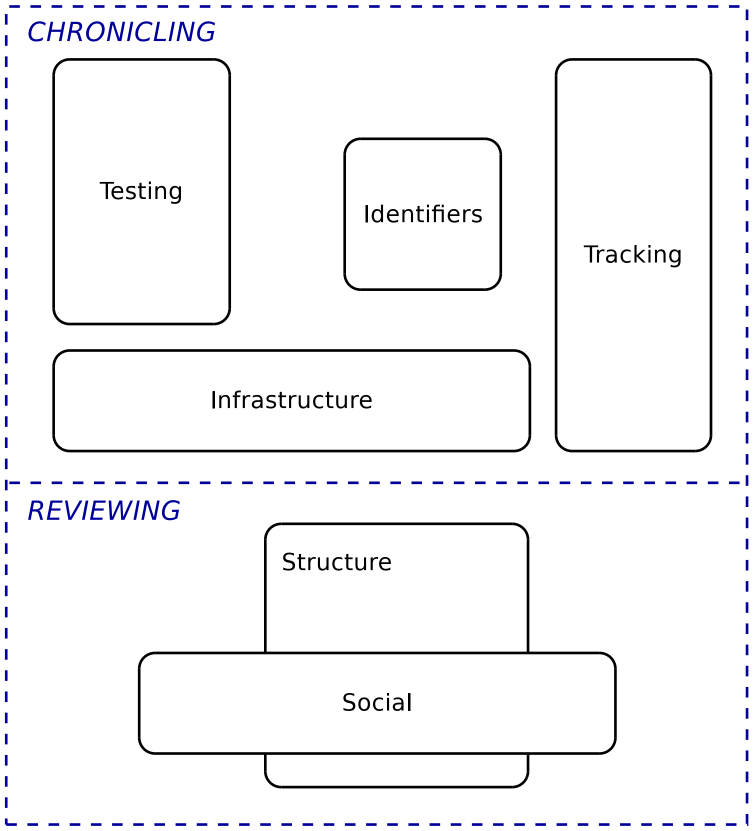


Figure 4: Model of software developer tasks emerging from discussion

The most striking change in the new model is the overlapping of structural and social aspects. The "interaction" part of the model is now characterised as "reviewing" to represent the reasons participants gave for the significance of their peers' interaction styles. In §5.4 participants' accounts show that software is less likely to become a "monster" if it is created in the daylight by an open-minded creator.

The "indirect" part of the model is now characterised instead as "chronicling", emphasising the importance that participants placed on these elements as much as sources of information as of functionality; what an artefact tells them is as valuable as what it does. As described in §5.5, good chroniclers create a detailed history in their work that helps future readers to gather valuable information from it. A commenter on a poster presentation of these results expressed it well: "Documentation is scaling for knowledge" (K. Glass, personal communication, PyCon UK, September 2015).

Attitude still applies across all aspects of the model and participants' thoughts on this are reported in §5.6. Participants also mentioned how not just they but the companies they work for are affected by the issues discussed. This and the part their working environment plays in exacerbating or mitigating problems is the subject of §5.7, as is the value they perceived for themselves and the business in having such reflective discussions of working practices.

5.4 The Reviewing element: live communication

Participants expressed very little concern about difficulties in understanding other people's lines of code. The interviews showed that for these experienced programmers the difficulties of the job lie at a higher, more structural level of granularity: finding the appropriate lines to look at in the first place. In discussing these problems participants spoke of the value of early and frequent communication between a programmer and their peers as an ongoing and informal review process. It allows team members to give timely feedback on approaches which make structure hard for others to make sense of, such as inventing a new (and therefore unfamiliar) way of doing things, using something that is unsuitable or unnecessary (e.g., a new and interesting framework) or simply writing code that is not needed.

This reflects programmer behaviour themes from the final template that were linked to consequences for others' understanding of the code (included in Figure 5). These include some technical skill to write code that is not too complex. For example writing more code than is really necessary, such as unused or unduly complicated features, means that future readers will be looking in a bigger haystack for the needle they need to work on. But social behaviours predominate. "Works in isolation" or "Asks questions", for instance, emerge as two sides of the same coin for the impact they have on others. It is not obvious that quietly getting on with the job alone can adversely affect anyone else nor that asking questions of colleagues is of particular benefit to any but the questioner, but they play a key role in creating a solution that works, fits in and makes sense. One participant (Participant A) put it in a nutshell: "To my mind development of computer systems is always a team oriented activity. There's very few people who will sit down by themselves in a dark room and produce a good, useful and maintainable system."

Talking to colleagues develops a shared understanding of the task. Solutions are less likely to be idiosyncratic or hard to navigate through. But these benefits depend on the talk being productive; a recurring dialogue theme emerged in participants' accounts of the attitude that needs to accompany the interaction.

Figure 5: Extract from template: live communication

5.4.1 Shared understanding

Asking questions is necessary to a shared understanding of what needs to be done and appropriate ways of approaching it. It allows misunderstandings to be detected and resolved in good time:

> "It's quite amazing how often people, and even yourself, can get hold of the wrong end of the stick really. Communication is very important ...But you need a team that's sitting down together, altogether, in close proximity to each other so you understand what's going on and you can ask any question. There shouldn't be anything out of bounds, you know, that's what I think. Otherwise you don't want to find out when it all comes together that somebody's totally misunderstood." (Participant B)

> "You always need to be listening to and talking to other people in the same team to make sure that you're all on the same track." (Participant A)

5.4.2 Conformity

Sometimes shared understanding is about knowing and respecting existing approaches. A programmer who communicates with peers is better than one "huddled in a corner" (Participant A) because they will benefit from a kind of continual, informal review process that lets them freely discuss their ideas with others and exposes them to team norms. Novel approaches, particularly a fancy new technology, need to be discussed. They can be a hindrance to others if they appear out of the blue:

> "The chances are it's going to be a bad maintenance experience in years to come. Because they've just not really understood the overall thing, and not tried to you know, fit in with the team ethos ...reflected in the overall commonality in the way things look and work and written under the hood." (Participant A)

> "Occasionally you seem to get new technology suddenly being put in because somebody liked it and thought it was the greatest and best and, it's always a bit disturbing to be presented with it for the first time when the project is going and you've never seen this thing before." (Participant B)

Lack of experience in a particular area is another source of puzzling, idiosyncratic code. Conventional solutions are easier to understand, and asking questions is an opportunity for learning them:

> “They can immediately understand what they’re doing ...without having to sort of wonder why you’ve come up with some other alternative. So that kind of thing, that saves time I think.” (Participant C)

> “I’m a bit more familiar with front-end sort of work ...But they perhaps are better when it comes to things to do with databases and things like that ...you start to pick up from them and when you have something you’re not sure about just ask one of your colleagues. And you know, at the same time they ask me about my knowledge and experiences with other bits.” (Participant C)

Conformity saves time for those trying to understand what unfamiliar code is doing. In contrast, ignoring existing solutions and creating something new can face readers with some complex and puzzling contraptions:

> “Then someone’s written their own code, and reinvented the wheel. And added an elephant onto it. And a spaceship.” (Participant D)

5.4.3 Navigation

The difficulties of navigating through others’ code often came up as physical metaphors: some sort of movement through a landscape, or wrestling with a monster.

> “It is navigation that’s, it can be very hard, and very hard to understand what it’s doing. ” (Participant D)

> “The more you dig deeper, the more you get exasperated by it. Because you’re thinking ‘what are, what have you done here? what, what monster have you created?”’ (Participant D)

> “You don’t feel like you’re wading if people go around and remove the weeds.” (Participant E)

The movement metaphor is particularly apt for another big concern: the size of the space that needs to be explored. Quantity mattered to participants because any unnecessary or unused code means a bigger landscape to navigate through. It also creates uncertainty about the work they need to do when they get there, as captured by this in this expression of frustration:

> "If it's in there it's got to be maintained and um, you say they're not currently needed but it could have an impact on some other existing code that is needed and then it's all tied up together and you're afraid of deleting or fixing the existing and used code because you break this other code because you're not quite sure if it's needed and it's just like 'ahh, don't add it in there if it's not needed'." (Participant E)

Irrelevant code not only obliges participants to search for a needle in a bigger haystack but also creates unnecessary work. Unless it can be identified with certainty as unneeded it has to be maintained, not removed.

> "Not leaving redundant code in. It's hard enough to track real code." (Participant B)

5.5 The Chronicling element: development as documentation

The type of communication discussed so far has been live dialogue. Although its consequences remain embedded in the shape of the code that others will later read, there are also more explicitly documentary ways for programmers to leave behind information which can later help their peers. Figure 6 shows the final template themes for programmer characteristics associated with such activities, and for the consequences they have.

The essence of the chronicling concept is that information should be preserved for the author's successors. As Participant J put it, "We're not writing in assembly language because we're talking to each other through this stuff". One very specific way in which programmers can talk to each other clearly through their code is to use meaningful names (for an overview of features of meaningful naming see Liblit, Begel, & Sweetser, 2006). As experienced programmers, participants often expressed a "seen it all before" attitude to poorly written code; by this stage in their careers they were adept at figuring it out. There is some parallel with other forms of expertise; expert chess players, for instance, can absorb the details of a board position provided that the configuration is one that could actually be reached within the rules of the game (Chase & Simon, 1973). Similarly, it seems experienced programmers are able to take in what a section of code is doing if it is syntactically correct. They recognise its shortcomings, but are not much hampered by them. As Participant L put it: “The thing is I’ve been doing this long enough that things that are potentially really annoying, I’ve learned to put up with it and I’m like, ‘Oh yeah, OK. Whatever.”’

Figure 6: Extract from template: chronicling

5.5.1 Meaningful identifiers

In the revised model the more general “code” element has become specific to “identifiers”: a serious issue which provoked frustration in a way that the annoyances of limited style and skill in a peer’s use of a programming language use did not. While programming language syntax is unambiguous, even when used in unusual, overly complex or unidiomatic ways, there is no such guarantee with the names that a programmer chooses for data and functions. Poor identifiers were the one fine-grained detail of coding practices to be frequently and spontaneously cited as a problem. In real life, code comprehension is typically part of a modification task that often demands not only understanding what is happening in a section of code but also locating that section in the first place. Meaningful vocabulary is useful for both. For example, to locate code implementing a cut-and-paste feature the programmer may search for identifiers containing the word “cut” (Rajlich & Wilde, 2002).

Chronicling is about the programmer preserving the information that they have, being able to see things from others’ perspective and understand what will help them. The “others”, in this respect, may include their own future self. Meaningful identifiers could be considered another form of preserving information: good ones document what data a variable represents, or what job a method is doing. Themes from the final template concerning the symptoms and consequences of badly-chosen identifiers are shown in Figure 7.

Figure 7: Extract from template: identifiers

Identifiers are singled out here from other sources of information because they featured so prominently in the data. Approximately 50% of the cards in the sorting task describe behaviours which relate to the many small decisions in how lines of code are written. Yet these did not feature frequently in the “noticeable impact” categories, nor were such issues often raised spontaneously in the initial stage of the interview. Identifiers are the notable exception:

> “Choosing vegetable names for variables …years ago there was somebody at another company that thought it was humorous.” (Participant B)

> “There’s a wonderful document out there on the web about how to write unreadable code. Specifically so you can keep yourself in a job by making yourself indispensable. Everything that’s in there you see people do in real life. You know, things like calling all your integer variables I, J, K, L, M, N, O regardless of what they do.” (Participant A)

In both cases the original programmer had adopted a theme for the variable names, rather than think up meaningful names to convey what the variable contained. The letter theme was the only prosaic example to come up. Playful or outré themes were more common and included Star Trek characters, characters in a novel, swearing and offensive words, and species of fish. Some lighthearted names managed to be at least somewhat meaningful in context, such as moon (a boolean) and blue (a constant) used to conditionally execute a block of code “when moon is blue”, or the use of “ickle” to mean “little”. No theme or whimsy was welcomed by participants; as its readers, they found such code readily memorable for its unreadability.

The topic of poor identifiers, whether meaningless or verbosely meaningful, also provoked vivid accounts of frustration:

> “If you ever try to read Linux kernel code where you just get these functions with data structures with two letter names that aren’t documented anywhere and makes you want to eviscerate someone.” (Participant G)

> “So you end up with ridiculously long variable names and that drives me insane because it just becomes a soup.” (Participant C)

For methods, the identifier issue is closely related to that of partitioning the program into blocks of functionality. If it is not a cohesive block it is hard to give it a meaningful name:

> “Just reading the name and you know, addVATToOrderAmount, okay, that’s a sensible method name. And if you can separate out some code that does exactly that, great. Whereas if you’re just randomly taking a chunk of lines and shoving them out into another method just because this one has reached 20 lines or whatever isn’t helpful. You’ll find out because you can’t give it a good name.” (Participant A)

In contrast to the problems posed by identifiers, choices reflecting differences in coding style are not an issue for these experienced programmers.

> "I mean it's some very different coding styles from people. And you get used to reading coding styles the more you work with people." (Participant B)

> "Say if you're writing in Java for example, it's, you know, it's Java. Java's Java." (Participant D)

This explains why identifiers emerged as the only fine detail of code to exercise participants. Programming language syntax is unambiguous, however inelegantly used, and its meaning is not changed by any layout that is not part of that syntax. But no amount of experience can help with guessing what data "Spock" represents.

5.5.2 Comments

Like identifiers, comments are intended to explain the meaning of code. In both cases the wording of them has no meaning to the computer so its sole purpose is to communicate with a human reader. The difference is that identifiers are essential in the program, hence the problems described above when their meaning is obscure. Comments, on the other hand, exist only help explain a program and it was clear that they do this with mixed success. Many participants were exercised by this topic and mentioned it spontaneously before seeing it on a card. On all other topics there was a broad consensus among participants about what they want from their peers. The lack of consensus on comments is evident in the polarisation of the themes: a number of different reasons why comments are a hindrance, and a similar number explaining how they are helpful. These quotes are illustrative of the conflicting opinions:

> "Generally I would say that if you have to comment your code, your code is unreadable. Change the code. Don't comment it." (Participant A)

> "And if you don't put comments in code and things like that it makes it even harder still, you know." (Participant D)

What does unite opinion is that a programmer must consider their work from an outside perspective, putting themself in the shoes of someone who later needs to work with their code:

> “Or they’re not as smart as they think they are, or they just don’t care what anyone else …the fact that they know anybody else is going to have to deal with it.” (Participant G)

> “Rather than just being a bit sort of ‘OK just get it in there’ it is a bit more sort of thought about ‘OK, well, if this does happen’ or ‘if this odd sort of situation does happen, what would help me actually resolve that’.” (Participant C)

Participants differ over whether comments will help, but agree on the likelihood that there will be someone reading the code in the future because the job predominantly involves working with other people’s code rather than starting with a blank slate:

> “It’s a big part of your existence. It’s part of the frustration.” (Participant A)

5.5.3 Logging

Another chronicling theme which arose spontaneously was the benefit of code which writes appropriate and informative messages to a log file as it runs. Like commenting, too much and too little logging are both cited as a hindrance, but there is consensus about it being valuable for debugging when done correctly.

> “You don’t want just to be swamped with ‘this is every variable I had in the scope’ which I’ve also seen in the past …It’s actually being able to understand what might be useful or necessary at this point to stick out into the log file and whether that includes data or not, or to just try and inform a bit more about what caused the problem as opposed to just that there was a problem and it was vaguely in that area.” (Participant C)

> “Again it’s a very fine balance, getting this right.” (Participant D)

As with comments, getting it right means imagining what would be useful from the perspective of someone faced with needing to understand what is going on. If someone is reading logs, this will typically be because they are debugging what has gone wrong. The right message can make this task easier:

> "So it just informs you a bit about what's going on and you may not have to sort of jump straight into debugging the code, so it might jump out at you immediately what the problem is." (Participant C)

This also applies to the recording of bugs, where a lack of detailed information about the symptoms and circumstances was a theme often recounted with some frustration:

> "It doesn't really sort of help you work out what's going on and it's very frustrating to be given a bug report where somebody goes 'Oh something doesn't work' and then actually you spend you know, a good chunk of your time sort of communicating back and forth and back and forth 'What were you doing?' You know, all these sorts of things 'Oh yeah I was doing that thing.' 'What did you click on?' …'When did it happen?"' (Participant C)

More often than not the source of such frustration was bug reports from non-programmers rather than from peers. Fellow programmers are better placed, in the sense of both expertise and location, to offer the kind of information that is needed to tackle a bug:

> "Oh God, we have a lot of customers in non-English speaking countries who seem to put a one line description of their problem through Google translate and then go to bed because they're twelve hours ahead and that's always a bit frustrating." (Participant G)

> "If you can replicate something it's better to tap you on your shoulder and say 'If I do this, this and this' and 'Oh yes, yes, I know what it is', fix it. Rather than someone sitting down for five minutes creating a screenshot, another five minutes to submit it and then you go through it, and press that and do this and- something that could have been demonstrated." (Participant B)

> "Yeah it is kind of irritating but again we're a small team so it's not too difficult and we just sort of shout over the desk [laughs] 'What did you do?"' (Participant C)

5.5.4 Commit messages

Peers attracted criticism for less than informative commit messages when submitting their changes to a version control system. It is evident from participants' observations on this theme that the change history tracking the evolution of the code is a potentially valuable source that they turn to for information:

> "You'll think 'OK, this code is doing something I don't quite understand, maybe I'll look through the history and see if it's changed to do that' And there won't be a test against it or a commit message that says 'This is why it's changed' or a comment or anything, and it, that doubt slows you down because you don't know if you should change this bit or not." (Participant H)

> "God yes, there we have a rash of, every time we've taken on new graduates we have to get them through the 'No really, you need to put a message in every commit'. It's a phase of training." (Participant G)

5.5.5 Automation

The behaviour that most stood out among those that help successors to the code was automation of tasks such as a build or release. This topic was sometimes raised spontaneously in the open discussion before the card sort, and among the card topics it was categorised as having noticeable impact more frequently than any other card. Being able simply to run a script instead of following a series of manual steps clearly makes things easier and faster for participants, but questioning them about why automation helps them also revealed a benefit in the form of "executable documentation". Not only has the writer of the script shared the information they have about how to perform the task, they have done so in a way that allows a greater degree of confidence. A script is tested by continual use, unlike a text document that can easily get out of date — and even if the text itself is correct, following it remains prone to manual errors.

> "In the majority of cases it's just hit a button and off it goes and does it all. It saves so much time and reduces the chance of problems occurring as well, which is mainly the big problem." (Participant C)

5.6 The Attitude element

5.6.1 Openness to ideas

Participants spoke not only about why ongoing communication is necessary but also about how: the essential ingredients to make such communication successful. An imperious attitude is not helpful:

> "'I am God, this is how you do it'. No." (Participant D)

The value of asking questions is interlinked with the theme "good at helping others"; there needs to be someone willing to answer, and do so constructively. Participants commented on feeling "lucky" and appreciative for the help they have received. It is not surprising that they find it useful when others are willing to help them, but the benefits go beyond an individual getting answers; it keeps the team aligned:

> "Some people can be harder to approach than others. And it makes it a lot easier if you can ask a question, frequent questions and small questions, then you realise you're, you're aligned and both going in the same direction, following the same path. " (Participant B)

For conversations about a particular piece of code, being open to discussing suggestions is an important theme. In a properly open conversation, it could be any or indeed all of the contributors who learn from the exchange. If discussion is shut down, an opportunity to change things for the better can be lost:

> "You might come up with an idea and ...somebody says 'there's a better way of doing this and this is not the best way because' ...' then you can say 'well that's a learning experience' you know, you can learn from that." (Participant F)

> "You say 'I don't understand this bit of code' and their response basically is 'well that's because you're an idiot'. And by the time six people have knocked on the door and said 'I don't understand this bit of code' you really think they should start getting the hint that it doesn't matter how clever they are compared to the rest of humanity, they've gotta write for their audience." (Participant G)

5.6.2 Attachment

Openness is impossible for someone too attached to an idea. The card topic from which the "owns code" theme developed was originally envisaged in terms of territorial behaviour, creating code that will be difficult for others to understand when the need arises because no-one else has been allowed to touch it. But the card was more often talked of in terms of the programmer becoming too attached to code, and thus reluctant to change or discard it:

> "They build an attachment to what they themselves have produced and a particular way of doing things." (Participant H)

The attachment may extend beyond their own code into a wider insistence on people doing things "One True Way". One participant outlined the difficulty of resolving the disagreements this can provoke:

> "Sometimes they can end up butting heads for no good reason really other than you've both learnt different ways ...If there's no statistics to say which ones are actually best then all you've got is what you've been told and what you've experienced and if both of your experiences have been good then there's no real argument to be had. All the pluses and negatives are really theoretical because you've both had success." (Participant H)

This observation hints at the importance of explaining a logical reason for the adopted approach. Sometimes the result of a question will be a change to the code that provoked the query. But often it will be the questioner's own knowledge that changes; things become easier to understand and work with when they learn about the reasoning:

> "Sometimes what you find is, people say 'This is how we do it'. 'Why?' 'Because it is is.' 'But why?' 'Because it is' ...If they can sort of quantify and say 'This is why, because of all these factors', then you go 'Oh right, OK, I understand now."' (Participant D)

5.6.3 Mature, professional conduct

For conversations to occur, those with questions or feedback must be willing to tackle the issue, even when faced with someone who is not open to discussion:

> “You have to be strong enough to question that person even, whether they’re going to take umbrage or not to that.” (Participant E)

> “If no-one sort of goes to him and says, you know, ‘Why have you been doing it that way?’ then it sort of gets forgotten about and then it just rolls on day after day, doing the same sort of thing. Whereas I think people need to go ‘Hang on a minute, why are you doing it like this?”’ (Participant D)

When they tackle it, they need to do so constructively and diplomatically; it should not come over as an attack on either the person or their code. Similarly, a person receiving feedback should not take it personally or be upset:

> “So it’s not all negative or we say ‘Have you considered this?’ rather than ‘This is rubbish’.” (Participant H)

> “If you don’t talk about things and say ‘Ooh, don’t talk to such-and-such, it’ll upset him’, well, I dunno, grow up, man up, you know. It’s, we’re all adults here and we’re all trying to do something for a customer.” (Participant D)

> “There’s someone in the current role who is very stubborn and that’s not helpful if you’re trying to come to something together. So he has this ability to make himself come across as affronted by any questioning if you can do it in a way where you can separate the person from what you’re talking about you can have a fantastic discussion and we actually promote that, that’s fantastic.” (Participant E)

The phrase “trying to come to something together” summarises the essence of participants’ talk about fruitful technical dialogue: it is not about ego or point scoring but moving forward with an appropriate technical solution and a better understanding. Programming is an exercise in problem solving.

5.6.4 Attributions

It is a human tendency to infer causality for the behaviour of others; in psychology this is termed attribution. Sometimes participants expressed thoughts about why their peers act as they do. While the focus of this study was to identify which behaviours are significant, the perceived reasons behind them might help to inform any intervention intended to change behaviour, although they should be interpreted with caution. There is psychological evidence for an Actor-Observer effect in attribution (Hogg & Vaughan, 2011), in which our own behaviours are attributed to external factors (such as those in §5.7.2) and others' behaviours are attributed to their internal disposition. This perceptual bias may mean that the perceptions are erroneous (Tashakkori & Teddlie, 1998).

Figure 8: Extract from template: attribution

Of the attributions participants made (see Figure 8 for the themes from the final template), some are simply about experience; never having seen later stages of the product lifecycle, for instance, means limited opportunities to understand the consequences of actions taken in the earlier stages. But experience too has its drawbacks if it becomes ingrained as learned habits and reluctance to change.

Conservatism also manifests itself in failure to look for alternatives; conversely, good programmers learn new things and update their toolkit so that they have a wider range of ideas to call on. It is possible, though, to embrace new ideas too enthusiastically; a healthy degree of scepticism is needed for the "flavour of the month", together with an appreciation that introducing new technologies will affect others:

> "And you have that sensation quite often, finish a project and you've just, you finally got to grips with everything that was introduced last time, only to think, woah what's this?" (Participant B)

Getting the balance right requires an interest and openness for new ideas combined with a collaborative approach to change:

> "Everything comes down to interacting with people ultimately ...I've learnt over the time that just coming in and saying 'This is the way we should do it you idiot' isn't, funnily enough doesn't get things to work out the way you'd hope they would. So gradual change or something, or convincing people you're competent and then trying to talk them around is sort of the way to go." (Participant G)

5.7 Business implications

This research is concerned with the impact on software developers of peer behaviour, but professional developers write software for an employer. Some themes emerged which illustrate the significance of the issues for business and the environmental factors that can themselves help or hinder.

5.7.1 Consequences for business

To summarise briefly, programmers want their peers to communicate open-mindedly with colleagues for timely feedback and to put themselves in the shoes of others in order to include meaningful identifiers within the code and helpful information in the framework of artefacts that surround it. The emotive words they use suggest that these things matter to them.

Figure 9: Extract from template: business consequences

What businesses want from their software is that it works, is delivered on time and makes a profit. Why should it matter to them just how it was done? The answer lies in the final template themes shown in Figure 9. Even if software were written, delivered and forgotten there would still be reason: good software developers who leave because of low morale are not easy to replace. But software is revisited, often time and again, as it is fixed, updated, extended or re-purposed. Participants describe how peer behaviour can: make tasks harder or easier; create tasks that could have been avoided; cause or prevent problems. They also mention how these issues affect the one thing that is generally the major cost of a software project: their time.

> “ Working around it all, effectively re-writing code that does the same thing somewhere else.” (Participant H)

> “Rather than doing a complete job on it, it’s a bit of a quick fix and then you get, you come around to it about three times again and it’s a bit like ‘oh maybe we should have just sorted it out in the first place’.” (Participant C)

> “In maintenance work it’s pretty much about being able to speed read somebody else’s code ...you scroll down and each method comes up,

> ‘Yep I see what that does, I see what that does, I see what does. Yes, that’s fine’. Whereas if you’ve got a big wodge of text, suddenly you’ve got to stop. You know, ‘What exactly does this do?’ It just slows you down.” (Participant A)

5.7.2 Confounding and mitigating factors

While the focus of the interviews was on peer behaviours, such actions do not occur independently of their context. Sometimes participants mentioned how an issue does not affect them as much because it is mitigated by factors in their workplace. Conversely, there were problems which can be created or exacerbated by external pressures. These themes from the final template are shown in Figure 10.

Figure 10: Extract from template: external factors

The beneficial factors included practices the team follow, such as a stringent process of code review. Similar benefits were cited for pair programming. In both cases the work is subjected to timely scrutiny, allowing a course correction for any errant software.

> “It’s an area also where pair programming helps tremendously. If one person is writing the code and somebody is looking at it and saying ‘What are you doing? What’s that? No, that can’t be right’. So pair programming or code reviews and so on, all that helps.” (Participant A)

Unit tests are also beneficial, beyond just checking the code’s correctness. The cards about trying to leave a module a bit better and not being constrained by existing code both elicited some equivocal responses; refactoring can be desirable but nonetheless inadvisable due to the risk involved. Because tests reduce that risk they make it easier for improvements to happen:

> “Tests to check existing behaviour …once you’ve got those you can change it however you want and make it much nicer.” (Participant H)

Tests were also mentioned as another kind of executable documentation, giving clues as to what the code should do. However, this must depend to a degree on how well the test is written. One participant’s anecdote is a salutory reminder that following policy rules does not guarantee that the practice is properly carried out:

> "We'd poke him to write unit tests and the unit tests would appear to work, but if you looked closely he'd written the unit test by running the function, looking at what came out of it copying and pasting out the assertion." (Participant G)

The other main theme for beneficial factors was the team's environment. Communication is of enormous importance, and live conversation the preferred medium, avoiding what one participant called "the electronic warpath". As they spoke about their interactions with colleagues, it was evident how participants' office environments (all of them open plan with no or low screens between desks) facilitated this: they are able to "push back", "lean over" or "turn round" to a colleague when they have a question. One described how they have deliberately removed high screens from between their desks and no longer have to "meerkat up" to do this.

5.7.3 Value of the reflective discussion

Participants not only responded to questions but also volunteered their thoughts on the process of taking part in the interview. Their comments suggested that they appreciated the opportunity to step back and look more deeply at what most affects them. One found it "a really interesting idea" to focus a discussion not on the usual technical details but "What makes you irrationally angry about a codebase or a team of coders? What makes you overjoyed?" (Participant J).

Reflecting at this level was evidently rather different from the content of the discussions that normally take place. Participant J spoke about the need "to circumvent all the nonsense arguments" about issues such as use of whitespace, a sentiment echoed by another in describing "pet peeves" that assume undue prominence at the expense of more substantive and troublesome issues:

> "Where you put things like the braces in C and C++, whether you have a space between the if and the bracket. They do get incredibly wound up about things like that. And yet they won't get too wound up about not having sufficient information on the bug report." (Participant K)

Citing another example, the practice of encapsulation, Participant J demonstrated how even talking about substantive principles can fail to hit the mark when the conversation does not help people to understand why and how those principles matter : “[Encapsulation] is one of those deceptively simple words …He listened to all the words and completely missed the point.”

It is evident from their feedback that the participants do not experience fruitful discussions about the impact of working practices in their normal working lives.

5.8 Discussion

The study set out to answer the research question: what is the perception of experienced programmers on the peer behaviours that help or hinder them in their own tasks? To elicit these perceptions, it encouraged programmers to draw on their own extensive experience and used card sort materials to help prompt them to consider the wide range of activities that their jobs involve.

The researcher’s own history as a programmer also played a significant role in this process. Participants spoke as to a peer, using the language of the profession. This gave the interviews the quality of a frank discussion between fellow programmers, without the baggage of colleagues at the same workplace or the commercial constraints of speaking to an outsider, affording a very intimate insight into the reality of working on a software development team.

The picture that emerged was unexpected. The anticipated themes, as represented by the topics on the cards, included considerable material drawn from guidance on how code should be written. These are not unimportant; few topics were commonly deemed “neutral” for their effect on participants’ progress. But the things that cause them most hindrance and frustration, and the things that make the job run more smoothly, are not in the fine-grained details of lines of code. Syntax does not excite much passion; it may sometimes be used in a way that is less easy to read, but it is never ambiguous. What makes most difference is how readily information is available.

At an architectural level, this means: writing only code that is necessary, not reinventing the wheel, and creating a “natural” solution rather than a “clever” one. To use participants’ metaphors, others reading the code should not have to machete through a jungle, grapple with a monster or ask “Why is this here?” These needs are consistent both with the findings of Sillito et al. (2006) about the practice of navigating through code and with the problem of violated expectations as described by Hansen et al. (2013). Meeting the needs of an imagined future reader (possibly even one's future self) calls for some empathy, but also benefits hugely from getting an outside perspective in the here and now. As some participants suggested, the practice of pair programming can help with this. Though not much mentioned in the study, regular code reviews could also offer the opportunity for dialogue; perhaps they did not feature more because this is not the manner in which they are usually conducted. When mentioned, it was as a bottleneck waiting for a reviewer's time, and it appears that some companies conduct their reviews as an electronic monologue. But interaction with colleagues is essential for the timely feedback it provides; no-one should be reluctant to have a dialogue about their work, to speak up or to listen.

No participant called for more, or better-maintained, text documents to provide the information they need. No doubt they would agree that "only the code tells the truth" (Sommerlad, 2010) or at least that, because it defines what the program does, it is guaranteed to be a reliable source of that information. But they do want better information about things that are otherwise hard to deduce. Good identifiers help them to read the code by complementing the grammar of the syntax with meaningful nouns and verbs. Unit tests provide almost as good an account of how the unit should behave as the code itself provides of what it actually does. Automated scripts can dependably show them how the software is built, tested and deployed.

Should something go wrong, participants are skilled at diagnosis if they are but given a clue about the symptoms. The messages recorded in tools supporting the development of software are worthless if they do not offer this. Bug reports and commit messages should share the information that is possessed by their writers; the study shows that like the code itself, these messages do have readers. This is true, too, for logging messages. They fail in their purpose if the writer does not put themselves in the reader's shoes and ask themselves "What would I need to know?"

Feedback from the study participants was that the interview had been enjoyable and interesting. Often this came as they were leaving, but one commented while the recording was still running: "Thank you for listening! ...I feel much better now, thank you." (Participant J). It is unusual for this community to be able to speak freely about their work to an interested, independent, non-judgemental peer listener. Conversations with colleagues can be subject to many workplace constraints (office politics, one's reputation, company culture and so on), while commercial considerations and the technical knowledge of the listener limit the extent to which work can be discussed with friends and family. Furthermore, conversation with a non-specialist is unlikely to flow. They can offer sympathy, but not empathy with the particular frustrations of a particular technical scenario.

However this also highlights that care is needed in interpreting the data. People who volunteered to spend 90 minutes talking to a stranger about their work are probably not among the most reserved and uncommunicative of software developers. The value they place on communication may not be shared by all their colleagues. Nonetheless they made a good case for its importance, talking not about personal preferences for workplace conversation but their experiences of problems avoided and solved early on, or discovered too late. Their comments on the process of these reflective discussions suggest that it would be useful to have them with their colleagues too.

Nonetheless the findings have some parallels with Curtis, Krasner, and Iscoe (1988); the software development professionals they interviewed did not identify exceptional designers by their skill at producing software. Instead, expertise was marked by knowledge of the application domain, time spent communicating with other team members and commitment to coordinating all individuals' efforts towards the overall success of the project. Peer perceptions such as these are harder to collect, but contribute to a much richer picture of successful projects than quantitative measures alone.

5.9 Personal reflections of the researcher

Women are a minority in the software industry but throughout the research I was conscious of my gender only when people asked me "are you looking at gender differences?" This question never came from participants and I never felt that they treated me differently for my gender. The fact that only two interviewees were female serves to underline the fact that I am inevitably used to being a non-stereotypical programmer. I wear this lightly and in these interviews my salient identity was simply that of programmer, but I should consider the possibility that gender stereotypes may have made it easier for participants to share their thoughts with a good listener with whom they can discuss things that have gone wrong without loss of face. If so, it was not to the detriment of the research. In a later conversation with a participant he agreed when I likened the process to "therapy for developers". If finding me easy to talk to was a factor, it was a positive one that encouraged participants to share what was on their minds. It thus enhanced the contribution of the research by faciliting a rich understanding of the issues that are important to developers.

My age did not appear to be a factor in the interviews either. Given their considerable experience in the industry many of the participants were of a similar age to me, but there was little discernible difference between the age groups except the greater degree of reminiscing from the older participants. However it is possible that my age (and thus experience) interacted beneficially with gender. Women in technology do sometimes experience credibility barriers. For example: "I had a client where I had to bring a guy to every meeting, with detailed notes on what he should say. The client ignored everything that came out of my mouth, so I got a myself a dude as a frontman" (Aas, 2019). While I have never resorted to such a drastic solution, the problem is an all too familiar one.

A personal characteristic that I do believe was very important is that I am an experienced programmer myself. Participants in the Exploratory Study were aware of this; it was mentioned on the information sheet they received before taking part, I talked about it at the start of the interview when explaining the motivation for the research, and in some cases they had first met me as a programmer. They all spoke to me as a peer, using our shared technical vocabulary in ways that would not have been possible had I not been an experienced programmer. They opened up as to a discreet peer who shares their profession but not their workplace. It struck me as a kind of conversation they could rarely have so freely: at work, what they can say is constrained by the necessity of maintaining a good working relationship or reputation; elsewhere, it is limited by commercial confidentiality and the extent to which friends and family understand and are interested in the subject. The participants appeared relaxed about speaking to me, talking freely about companies, products and colleagues, including people they knew I was interviewing.

I was concerned that my own experience might influence the discussion too much. In the event it was more of a benefit than an intrusion. Rapley (2004) argued that an interview is a cooperative work of both parties, and that sharing one's own stories can facilitate the conversation. The benefit was evident in the technical nature of stories I heard, told in specialist language (e.g., acronymns such as IDE; words with context-specific meanings such as kernel, commit and front end) which would have required clarification if I had not understood them. This would have broken the flow if done at the time or risked failure to accurately identify the concept if attempted later from the recordings. Occasionally sharing anecdotes with participants was helpful in establishing my credentials as a fellow traveller and often elicited recognition or further anecdotes from them.

My own programming knowledge certainly informed many of my requests for clarification or elaboration in the form "oh, do you mean this?" questions that only a programmer could have asked. Participants did not passively go along with my interpretation. They frequently either elaborated on my version, or corrected it and elaborated. My own experience accords with Miller and Glassner (2004, p.130): “interviewees will tell us, if given the chance, which of our interests and formulations make sense and non-sense to them”.

The subject of the discussions was close to my heart. I think my efforts to prevent this from detracting from the quality of the research were successful. I was vigilant from the start and adjusted course when I caught myself talking too much in some of the early interviews. I learned from this - an important realisation was that people will say some really interesting things if I shut up and let them think. For most of the interviews the balance between my roles as a peer and a research seemed right. The peer role was, I think, as important as the researcher role; it is hard to imagine such rich, detailed and reflective conversations taking place between a software developer and someone who does not share their knowledge. Being an experienced programmer myself was instrumental in getting thorough and meaningful answers to my research question RQ1: “What is the perception of experienced programmers on the peer behaviours that help or hinder them in their own tasks?”

5.10 Conclusion

None of the findings of the Exploratory Study contradict the material in textbooks, academic curricula and conference programs. But perhaps such sources focus on practices that are less difficult to prescribe and measure, and on the hard technological problems — programming is, after all, not a simple task. There may also be some truth in the assertion that “the idea of the programmer as a human being is not going to appeal to certain types of people” (Weinberg, 1998, p.279).

Walz, Elam, and Curtis (1993, p.74), observing that “We have historically valued both technical and communication skills in software designers”, suggested there is a need for companies to take measures to develop both in more depth. While tools and training exist for the technical skills, there is still no obvious equivalent for the communication skills, leaving plenty of scope for work that addresses this need.

Chapter 6

Evaluation study: Peer impact workshops for software developers

6.1 Introduction

Research Question 2: Can experienced programmers' accounts of the peer behaviours that most help or hinder them in their own tasks be used to help others in their practice?

This chapter describes how the detailed design of the Evaluation Study introduced in Chapter 3 was developed in order to address Research Question 2, which is repeated above for convenience. The researcher's observations and the participants' feedback on the conduct of the Exploratory Study influenced the design. These ideas (§6.2) were evaluated in three pilot studies (§6.3 to §6.5) which tested and refined aspects of the workshop to inform its final design (§6.6).

The Evaluation Study tested the final workshop design for its ability to help professional software developers stop and consider a selection of peer behaviours, focus on those that most resonated with them, and explain the positive or negative impact locally. Based on the results of the pilot studies the workshop format used direct quotations from the interviews, representative of themes collected in the Exploratory Study, as its central material. The usefulness of the workshop to its target audience in helping them to constructively review their own practice was evaluated by obtaining feedback — both immediate impressions and later reflections — from the participants themselves. The results are reported in Chapter 7.

6.2 Influences from the Exploratory Study

The outcomes of the Exploratory Study (see Chapter 5) demonstrated the continued importance of communication skills in software development. Walz et al. (1993, p.74) recommended that companies should adopt a policy of developing both technical and communication skills in their programmers, but it is unclear what resources an employer should turn to for a communication skills equivalent of technical training that is tailored to their technical environment. This is the role the workshop design seeks to fill in an engaging and practical way.

The Exploratory Study asked interviewees to consider this question in assessing each card: "How does it affect me if someone does this?" This was an invitation to contemplate the consequences of a variety of actions from the perspective of a person experiencing those consequences rather than a person focused on the technical demands of a task. Such consequences are a non-functional aspect of the work (i.e.

not directly involved in providing a working feature within a program). Technical training can demonstrate tangible examples of how to achieve functional aspects, but the impact of non-functional aspects is less easy to convey. An opportunity to see the impact from the viewpoint of another person may help programmers to imagine and empathise with that position, and perhaps become more cognisant of the peer perspective when engaged in tasks. This premise guided the use of Exploratory Study methods and findings to design the workshop for safe group discussions on themes that had emerged from the interviews.

The card sort exercise used in the Exploratory Study proved helpful in enabling participants to reflect on their experience and consider the impact of a wide range of activities. They also reported enjoying this exercise as a fun and thought-provoking way to think deeply about their work. It was evident from the reactions that sometimes occurred as they read a card — laughs, sighs, expostulations — that the contents chimed with their own experiences. The success of this method in engaging participants suggested it could also have potential for facilitating group communication about such experiences.

A positive experience of a Continuing Professional Development (CPD) exercise improves the chances of a team choosing to use such an exercise again in future. There was evidence of this from the Exploratory Study, where a company reported that the participants had engaged in a great deal of discussion about the material in the weeks after the interviews. Having taken on further staff the company requested a copy of the interview cards to facilitate a similar discussion with the newly expanded team.

6.3 Pilot study 1: Card materials in group discussions

The idea of creating a framework to help programmers step back, reflect and share their reflections was influenced by the reactions of participants in the Exploratory Study, who evidently enjoyed the opportunity to do so. Sharing with a wider audience rather than one-to-one was explored in pilot study 1.

6.3.1 Participants

A 90-minute session conference workshop session was delivered to an audience of professional software developers at the ACCU 2014 conference (Ollis, 2014), repeated for ACCU regional groups in Oxford and Bristol. These sessions were also an opportunity to publicly disseminate the findings of the Exploratory Study to the target audience for this research. Programmers participating in these workshops were brought together regardless of experience level by a common interest in professionalism in programming.

6.3.2 Materials

The materials used were those from the card sorting task in the Exploratory Study: the five colour-coded A4 category cards and a set of A6 cards setting out peer behaviours. For practical reasons this was a subset of 21 of the cards used during the one-to-one interviews, making it possible for a group to discuss them all and reach a joint decision on where to place them within a reasonable time.

6.3.3 Procedure

After a brief presentation about the research, the large part of each session was an activity in which people carried out a version of the Exploratory Study card sorting task in small groups (between three and five depending on the size of the audience and the venue), discussing their reasons for their decisions. This disseminated information about the topics discussed in the interviews in a more engaging way than a slide presentation and gave them an idea of what had been involved for the participants in the research. Importantly, it also acted as a trial for group discussion of the material.

6.3.4 Discussion

The material resonated with this audience of professional programmers and created a lot of animated discussion. Presenting the findings prompted exclamations of recognition and sharing of anecdotes about the same topics from audience members' own experience, but it was the activity in particular that generated the most interest. Many people asked about its potential for use in staff appraisal or development and as a means for facilitating discussion in teams; some requested a copy of the cards to use in their workplace. This reaction encouraged the decision to follow up these informal group workshops more formally, to test whether a similar format could deliver a practical application of the research as a catalyst for team discussions.

There was already some evidence for the cards' potential as a facilitator of group discussion from a senior software engineer from one of the companies who had participated in the Exploratory Study. He tried out an informal group discussion based on the interview cards with six of his team and found it worked well: "I suggested we lay out the cards, and each pick one or two that made us think 'oh yeah, that reminds me of …' and tell the story …I found that really helped allow everyone to contribute, as well, including the less experienced and quieter people — it's less intimidating to tell a story of your experience." (Senior software engineer, personal communication, June 20, 2014)

6.3.5 Conclusion

Together, the small workplace trial, the feedback from public engagement at ACCU and the positive response of the Exploratory Study interviewees to the card sort inspired the way forward for practical application of the research. The findings from the Exploratory Study include an inventory of team-friendly practices, which of itself has potential use for managers as a prompt to explore these in recruitment and appraisals. But in addition, the methods used to collect and disseminate this information showed potential for facilitating communication. The design of the Evaluation Study therefore focused on the possibility of using a card activity in a group discussion workshop to encourage understanding of the newly inventoried team-friendly practices.

6.4 Pilot study 2: Testing the logistics

Having established the capacity of the cards to engage animated discussion in an informal setting, a test was conducted to work out how to structure the discussion to be suitable for a single group within a company setting. In such a setting, the goal is learning for the particular team undertaking the workshop rather than the general opportunity for discussion with a broad spectrum of industry peers as in pilot study 1.

6.4.1 Participants

A discussion based around cards bearing quotations from the Exploratory Study (“quotation cards”) was piloted with the four research supervisors to test the logistics of the procedure. Four participants proved to be a viable number in terms of time taken and opportunity to contribute. Based on the amount of material that was covered in an hour with this number, an absolute maximum group size of eight — but preferably fewer — was agreed.

6.4.2 Materials

The quotations used were a small set selected simply for being a pithy phrase; the exact choice of content was less important in this pilot than the process for structuring the discussion. They were printed single-sided on white A4 sheets.

6.4.3 Procedure

Participants were seated around a large table with the quotation cards spread out on it and instructed to look through the cards and pick one that resonated with them. Props were displayed to remind them of ground rules given in the instructions. Once everyone had chosen, each person was invited in turn to explain their choice.

6.4.4 Discussion

The format worked quite well as means of creating discussion. Observations by the researcher and participants led to a number of small logistical refinements in the procedure:

- The instructions were subsequently made clearer and simpler and the possibility of a self-directed workshop rejected. A facilitator was deemed necessary to explain and coordinate the process and manage the time spent on each card.

- A toy car prop representing “your mileage may vary” (a reminder to respect the validity of others’ personal experience) was problematic because the phrase was not familiar to all participants. It was subsequently replaced by a toy steamroller representing a request not to “steamroller” (crush) anyone else’s personal account.

- It became evident that participants should be instructed to read their selected card aloud before explaining why they picked it. Cards should be printed on both sides so that the text is also visible to others when it is read out.

The structure in which everyone chose a card meant that everyone got an opportunity to have a say. Without knowing participants it would not be evident whether a workshop had helped them to contribute to the discussion, so it was agreed that the post-workshop feedback questionnaire should ask participants: whether they got a chance to contribute, and whether anyone who rarely speaks up contributed more than usual.

6.4.5 Conclusion

The results for the premise of the procedure were encouraging. The use of participants’ own words on the cards worked well, provided that the wording was simple. The quotation cards successfully expressed programmer opinions familiar to the computing specialists on the supervision team, and did so more naturally than the corresponding bland descriptions of the interview cards. For example, “Identifiers are very, very important” epitomises the reactions of interview participants when discussing the “Uses clear identifiers” card. The tested format, with the logistical refinements outlined above, was therefore deemed a promising one with which to proceed.

It is of no significance to this study which cards are chosen for discussion since the purpose is to facilitate discussion of whichever topics resonate most for that group. Nonetheless the choice of cards is interesting and should be recorded.

6.5 Pilot study 3: Testing with target audience

Finally, the workshop was tested with its intended audience using carefully selected quotations from participants in the Exploratory Study to prompt discussion. The purpose was to establish that the discussion procedure worked with its target audience of programmers: did the quotation cards evoke familiar scenarios and promote discussion?

6.5.1 Participants

A revised workshop format incorporating the procedural refinements from pilot study 2 was tested at the PyCon UK 2016 conference in Cardiff.

6.5.2 Materials: choosing quotation cards

Quotations were chosen for the quotation cards by reviewing the interview transcripts for resonant phrases to illustrate the common themes about peer behaviour that had emerged. Since the interview discussions had been guided by participants' decisions in the interview card sorting task, themes were initially shortlisted for inclusion by examining the results of that task and applying the following criteria:

- Select the majority consensus set: cards which were sorted into a "noticeable impact" category by at least 50% of interviewees. (21 cards, of which 6 had bad impact and 15 good)

- Exclude contentious topics. Although 18 interviewees placed card 53 (Uses code comments in ways that aid understanding) in the "Good — Noticeable impact" category there was a lack of consensus in how they talked about it.

Although at a high level it qualifies as a common theme, drilling down into the reasons given for the impact showed considerable disagreement about what is good and bad. It is inconsistent with the aims of the workshop to present material on which there is such disagreement; the focus should be on the impact of a behaviour rather than discussion of what constitutes good or bad behaviour. (Candidate set reduced to 20 cards)

- Make the materials language-neutral. Participants had noted that card 18 (Is rigorous about deallocating allocated resources) is language specific. A possible future avenue is to tailor a selection of cards (perhaps with a core common set) to help address issues of particular local concern, such as language-specific ones. The workshop process is independent of the card topics. (Candidate set reduced to 19 cards)

Consideration was then given to potentially interesting features of cards not in the initial shortlist. Three criteria were used:

- Examine any cards which the majority placed in the "neutral" category. Participants' commentaries sometimes indicated an "it depends" response to cards placed in this category. However only card 16 (Follows formal methods to the letter) met this criterion. It was ill-worded, intending to convey a dogmatic approach to software methodologies but often taken to refer to mathematical techniques for software verification. These techniques are not widely used throughout the industry and so were irrelevant to most participants' daily experiences. The card was not included. The kind of dogmatism it intended to convey is captured more generally by card 43 (Espouses "one true way" of doing things), one of the majority consensus set. (No change to candidate set)

- Examine any cards which show indications of dissent. Most have consensus on positive or negative valence, or at least neutrality. Card 26 (Logs it in the issue-tracking system when knowingly making a sub-optimal change) is the only one with more than one or two participants dissenting (4 "Bad — Slight impact"; 5 neutral; 15 "Good — Slight impact"; 4 "Good — Noticeable impact"). Some participants spoke in the interviews about the right approach being to not do a sub-optimal change in the first place. This is quite a context dependent card; what may be appropriate in an out-of-hours support call to provide a short-term solution for a customer's urgent problem, for example, might not be a desirable approach in other circumstances. Since the dissent can be accounted for and few participants reported a noticeable impact of any kind the card was not included. (No change to candidate set)

- Consider any other common themes not captured by the preceding criteria. Card 22 (Includes code features that are not currently needed) was placed in the "Bad — Noticeable impact" category by 8 interviewees and in only the "Slight impact" category by 15, but this topic was a common theme. It was frequently expressed in terms of the XP (eXtreme Programming) principle of YAGNI: "You Ain't Gonna Need It". "New technologies without good reason" was an additional theme not directly addressed by the interview cards, although in some ways it is the flip side of Card 40 (Finds out whether functionality is already available before writing their own implementation). Both themes were included. (Final size of set: 21 cards.)

Quotations were chosen from among those coded to the selected themes for their ability to clearly represent the theme, a characteristic checked by asking an experienced software developer to indicate whether they recognised the behaviour being talked about in each quotation. In a few cases this criterion was not met and with the help of the developer an alternative was selected from among the candidates. See Appendix H for the final set of quotations used. These were printed on A4 cards, referred to hereafter as "quotation cards". Both sides of each card showed a speech bubble containing the same quotation from an interview participant (Figure 11).

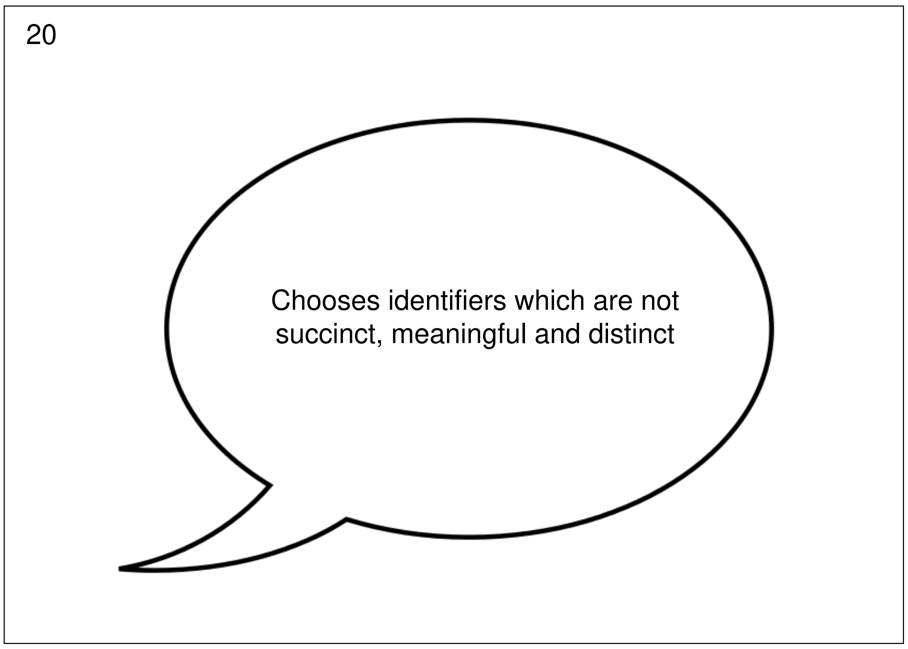

20
Chooses identifiers which are not succinct, meaningful and distinct

Figure 11: Format of quotation cards for Evaluation Study workshops

6.5.3 Procedure

The pilot took place within the context of a conference session so a short presentation about the research was given beforehand. Delegates were then organised into three groups of three or four and instructions were given to the room as a whole. Each group was given an identical set of cards to look through, with instructions for everyone to choose one that resonated with them for its impact on their work and then explain why to the group. Since this session involved multiple groups it was not possible to facilitate the process of each group individually. Instead the researcher circulated among the groups, answering questions about the process, listening to the conversation and prompting people to talk about the personal impact when the tenor of discussion had become more about good practice (what people "should" do) than the impact of such actions. At the end participants were invited to complete an online questionnaire to give their feedback. Three complete responses were received (seven online sessions were started; one answered only the first four questions and three contained no answers).

6.5.4 Discussion

There were practical difficulties with conducting this pilot exercise because it was scheduled across two conference sessions, meaning that some people left half way through to attend a presentation in another room, while others joined half way through. Nonetheless it was an instructive exercise that informed subsequent refinements to the workshop.

It was evident, from both the number of questions participants asked about what to do and from feedback in the post-workshop questionnaire, that the instructions given needed to be clearer. They were refined to say less about the research behind the workshop. These revisions continued throughout the full study until the final version of the instructions (Appendix I) said as little about the previous work as possible while still conveying that the quotes came from a wide range of experienced programmers — the participants' peers. Based on parting comments, it appeared that including too much of the research background in the instructions sometimes encouraged the mistaken impression that the goal was to learn participants' views for research purposes, rather than the actual goal of attempting to create a useful discussion.

One change was made to the content of the quotation cards as a result of this pilot because a participant found "Code is a shared responsibility for everyone. If you don't like others to work on code you wrote, well, get over yourself!" somewhat offensive. Other quotations might also be considered quite forthright in their language, but the way they are phrased reflects the feelings and vocabulary of real developers in a way that underlines their authenticity. No attempt was made to sanitise them to pre-empt possible future offence, but the one which had actually been deemed offensive was replaced with a less truculent alternative, “Anyone who thinks it’s ‘their’ code is missing the point.” The full set of quotations used in all subsequent workshops is listed in Appendix H.

6.5.5 Conclusion

The goal of the workshop to facilitate discussion by use of the quotation cards worked well. Observing the overall ”buzz” in the room and listening in to the conversations showed that animated discussions were taking place. Questionnaire feedback was also positive. One participant responded to the question “Would you do the workshop again?” with an encouraging ”Yes, I’d love to discuss these issues with my team and the cards would be a good starting point.” All three respondents reported that they would recommend others to try the workshop, one believing that would help to open up discussions that would be difficult to have otherwise. Together, the observations and the questionnaire responses encouraged pursuing the plan to deliver the workshop in a company setting.

6.6 Final design for the Evaluation Study

The Evaluation Study ran from November 2016 to September 2017, ethical approval having been granted in November 2015 by research ethics representatives of the Faculty of Science and Technology at Bournemouth University[1].

6.6.1 Participants

The Evaluation Study commenced with companies (but not necessarily the same staff) who had participated in the Exploratory Study. This served as a means of sharing with them, in an engaging and practical format, how their contribution to the research had been applied; a workshop building on the findings of the Exploratory Study offered participants a more interactive, direct and practical summary of the common topics than a report or presentation would.

[1]The former school of Design, Engineering and Computing which approved the Exploratory Study became part of this new faculty in a reorganisation.

Industry domain	Scale of organisation
Avionics	UK SME, single location (<250 staff)
Simulation and training systems	UK Multinational SME (<250 staff)
Equipment testing	Overseas Multinational (<40,000 staff)
Energy trading	UK Multinational (<100,000 staff)

Table 9: Profile of companies participating in workshops

Working with familiar companies also meant that the first tests of the formalised workshop format took place in an environment known to be welcoming and cooperative with the research effort. However in order to test potential industry applications of the material in its own right it was also taken to companies which had not taken part in the Exploratory Study.

The workshop was evaluated first with two companies who had participated in the Exploratory Study. These companies were recruited via the same contacts as before and are referred to as companies A and B. Participants were a mix of previous participants and newcomers to the research. Two companies new to the research (companies C and D) were also recruited, one via a contact from the Exploratory Study who had since changed employer and one via introduction from a university colleague. Broad characteristics of the organisations visited are shown in Table 9. Their locations (southern England and London) are excluded from the table to maintain anonymity.

The number of participants in each workshop is shown in Table 10. In total, five workshops were run and 24 participants took part.

Company id	Number in workshop	Of whom Exploratory Study participants
A	7	3
B	5	1
C	5	1
	3	0
D	4	0

Table 10: Participants per workshop

6.6.2 Materials

Preparatory and follow-up materials

An information sheet (Appendix J) was sent to all contacts with a covering email and with their cooperation arrangements were made to conduct workshops on the companies' premises. Researcher planning for each site visit followed the steps shown in Appendix K.

In the week before the workshop, participants were asked to complete an online questionnaire about their career experience (Appendix L) and work environment (the Stages of Team Development questions described in §3.4.2 and listed in Appendix M).

Feedback was collected using a paper questionnaire (Appendix N) immediately after each workshop and an online questionnaire (Appendix O) approximately one month later.

Workshop materials

Props (see Figure 12) were used as a visual cue to introduce participants to the ground rules for discussion in a fun and gentle way; a terrible pun in particular served as something of an ice-breaker. The props remained displayed on the table throughout the discussion as a visible reminder and, if needed, a gentle vehicle for enforcing the rules. The props were:

- A toy donkey, representing the idea that no card should be deemed obvious. The impact should always be explained, rather than making any "ASSumptions".

- A toy steamroller, representing the idea that everyone's account of their own experience is valid. In the event of others having experienced a different impact from the behaviour described on the card, the reasons for this could be discussed but no-one's experience should be dismissed, or "steamrollered".

- A toy cow, serving as a reminder that the question was not about common tenets of good practice but about the impact of the behaviours listed on the cards upon the participants. The participants were reminded to avoid the dogma of good practice "sacred cows".

The quotation cards used are listed in Appendix H. Apart from the one change documented in §6.5, the quotations were the same as those used in pilot study 3. The target demographic was demonstrably comfortable with the use of cards to facilitate discussion, having engaged with them both individually (in the Exploratory Study) and in groups (during pilot studies 1 and 3).

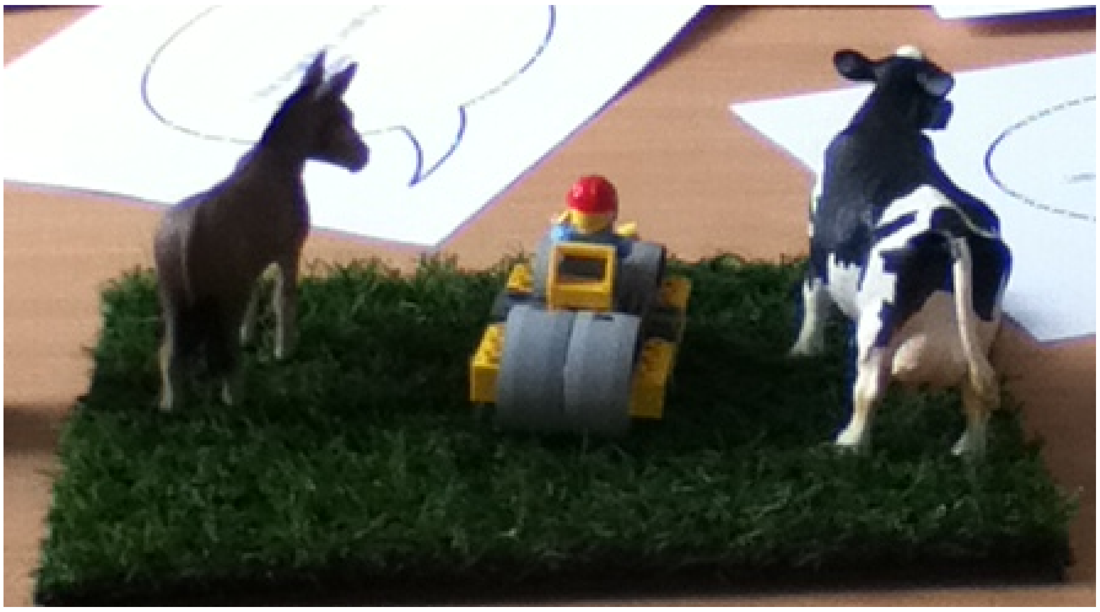


Figure 12: Props ready for a workshop

6.6.3 Procedure

Each workshop involved: pre-workshop questionnaires, the workshop itself, and subsequent feedback questionnaires.

In the week prior to the day of the workshop the local contact at each company invited participants to complete an online questionnaire (Appendices L and M). The workshop procedure was intended to facilitate discussion among willing contributors and its success in this was to be evaluated. While the procedure might potentially help teams in conflict to communicate constructively, it might equally be futile in such an environment. In such circumstances it would be impossible to evaluate its utility for its intended purpose, so some understanding of the prevailing environment was required.

All workshops were conducted in meeting rooms at the companies' premises. Before participants arrived, props intended to gently and lightheartedly remind participants of the "ground rules" — a toy cow, donkey and steamroller — were displayed on a field of artificial turf (Figure 12) and the quotation cards were spread out on the table in the fashion shown in Figure 13. A folded A4 tent card with a reminder of the question participants needed to ask themselves was available, but as all rooms were furnished with a whiteboard the question was written on the board instead (Figure 14).

When participants arrived they were thanked for coming and asked to complete the consent form (Appendix P). Recording devices (a digital voice recorder as the

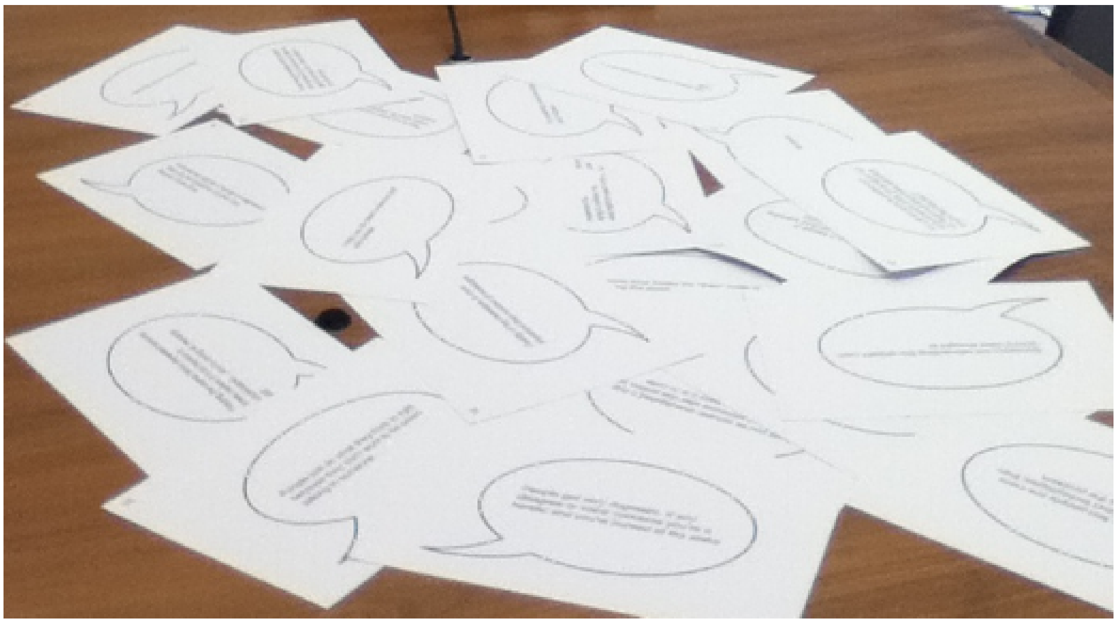


Figure 13: Cards laid out ready for a workshop

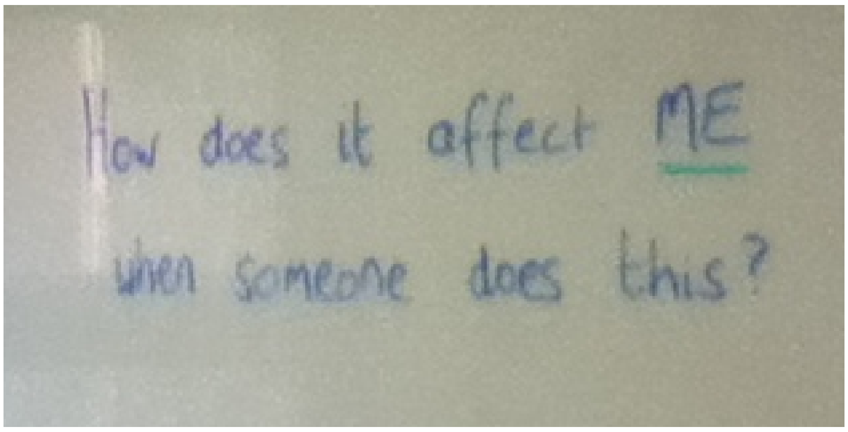


Figure 14: Question reminder for a workshop

main device and an iPod Touch as a backup) were started only after confirming that all participants agreed to audio recording. Contacts organising the visit at each company had been asked to distribute the information sheet to potential participants but printed copies were also available at the workshop and offered to participants along with the consent forms.

The procedure for the workshop discussion was modeled on pilot study 2 (§6.4) and the workplace trial reported in §6.3. It was explained to participants using an instruction sheet as a prompt to the researcher. The instructions evolved as the study progressed, with changes after workshops at companies A and B to reduce the references to the Exploratory Study; conversations with participants suggested that these references had created some ambiguity about the purpose of the workshop. Appendix I shows the final version of the instruction sheet as used at companies C and D, which emphasises the expertise represented by the quotations rather than the method by which it was collected.

In summary, the instructions asked participants to spend some time looking through the cards and choose one that resonated with them in response to the question "How does it affect ME when someone does this?" They were told that they would each be invited to explain their choice once everyone had chosen; the ground rules for subsequent discussion were laid out. Each of the props was explained at this point to illustrate the rules and serve as a visual reminder throughout.

Participants were then allowed time to look through the cards. No time limit was set and choosing took five to ten minutes. Once everyone had chosen, each participant was invited to read out their card and explain why they had chosen it. In the subsequent discussion, everyone was allowed to share their own experience on the subject. Should they agree, a consensus could be established. Should they disagree, the activity offered a basis for discussion about why they differ: perhaps contextual factors such as timing within a project lifecycle or some characteristic of the software subsystem. Using quotations offered the opportunity to openly express agreement with words spoken by some other, unknown software developer and relate the quotation to the local situation. The topic was thus by definition not merely an idiosyncratic pet issue but one commonly recognised by a broad variety of peers from other companies. This approach drew on the participants' fund of experience and war stories, knowledge which was described by a participant in the Exploratory Study as "The Lore".

In order to keep track of who had presented their card a map of the seating layout was drawn up and each seat checked off when a card had been discussed. In later workshops it was also used to monitor who had contributed to the discussions in order to know whom to encourage to speak. Timing was monitored to ensure that there was time for everyone to present a card. If time allowed within the allotted hour participants were invited to select a "bonus" card, offering them some opportunity to include a topic which it had been a hard choice to omit.

At the end of the discussion participants were thanked and given the opportunity to ask questions. Before leaving they were invited to complete written feedback forms (Appendix N). Section one included the debriefing information, recapping the purpose of the workshop and reiterating the right to withdraw from the research. Section two concerned the logistics of the workshop just conducted, asking how well the practicalities had worked: the clarity of instructions and materials, the extent to which people had been able to contribute, and any good or bad features of the procedure itself. Answers to these questions were used to inform the conduct of subsequent workshops. Section three addressed the second objective of Research Question 2 — evaluating the usefulness of the workshop — by asking participants for their immediate impressions about the usefulness of the discussion and how it might influence them or others.

Finally, participants were reminded that they would be invited to complete an online follow-up questionnaire (Appendix O). It was sent four working weeks later and asked about outcomes from the workshop they had attended: any actions taken or behaviour changes observed in themselves or other participants. Looking forward, the questionnaire assessed the usefulness of the workshop by asking about the circumstances in which participants would use it again. To address the final objective of Research Question 2 — identifying other possible uses of the material — the questionnaire called on participants' own insights to comment on other scenarios (e.g. recruitment) in which the material they had experienced could be useful. There was also space for any comments they wished to make.

6.6.4 Company B variant of the procedure

The design of the workshop was iterative, reflecting on the researcher's observations and participant feedback after each company visit and refining the process as needed. Apart from the revision of the instructions already mentioned, this approach also led to experimenting with a different card size.

The table space at company A was limited, making it necessary for participants to lay the cards out very carefully so that they could all be read. A participant suggested on their feedback form that the procedure should use one smaller deck of cards per person rather than a single set of A4 cards on the table. This was trialled at company B using A6 cards, but was much less dynamic and interactive as everyone sat quietly reading their own set.

Individual card desks also created the risk of multiple people picking the same card. Participants were therefore allowed to select a "shortlist" of cards instead of a single one, and when presenting choices they were asked to select from these one card that had not yet been discussed. However this made the instructions more complicated and the participants more indecisive in their eventual single choice. The procedure was therefore reverted to the original A4 cards thereafter, but care was taken to request a room with a suitable table when booking the remaining workshops.

6.7 Conclusion

The workshops proved effective not only at facilitating discussion of topics in the abstract but also in prompting recognition, analysis and sometimes proposed solutions to concrete local examples. As with the card sorting task in the Exploratory Study, it was clear from the frequent signs of recognition that the quotation cards resonated with the participants' own experiences. The usefulness of the exercise was evaluated by the participants themselves and the results are reported in Chapter 7.

Chapter 7

Findings of the Evaluation Study

7.1 Introduction

This chapter sets out the results from the Evaluation Study described in Chapter 6, in which experimental workshop sessions were run with groups of software developers to investigate whether this format, using material derived from the Exploratory Study, could serve as a fruitful basis for honest and constructive group discussion of local practices.

Data was collected from workshop participants at three stages and the chapter explores each of these in turn. §7.2 describes the working environment in which the workshops took place: both the stage of development of the team taking part and the career demographics of the team members, as collected using a questionnaire prior to the workshop. These results are discussed in §7.5.1.

Relevant to the research question were the opinions of participants about the value of the workshop. The quantitative aspects of these are summarised using descriptive statistics and the associated comments complement these by illustrating participants' reasons and suggestions. §7.3 addresses the practical questions: what worked well in the workshop procedure and what needed refining. The answers are discussed in §7.5.2. §7.4 reports participants' reactions to their own workshop experience and their thoughts on the usefulness of the format for themselves. The findings are discussed in §7.5.3.

As with the card sorting task in the Exploratory Study's interviews (Chapter 4), a participant's choice of quotation card was not an outcome in itself but an invitation to talk about the topic that they chose. The study was intended to test how well the process encouraged discussion of locally relevant topics from the cards, regardless of which topics those were. Card choices were, however, logged as an interesting record of which topics resonated with participants; this can be found in Appendix Q.

7.2 Pre-workshop questionnaire findings

Prior to the workshop all participants were asked to complete an online survey comprising the Team Development questionnaire described in section 3.4.2, which gives an impression of the context within which a respondent works, and a questionnaire about their career experience. The results are reported in §7.2.1 and §7.2.2.

7.2.1 Stage of team development (Tuckman questionnaire)

All participants (24 developers in 5 workshops across 4 companies) completed the questionnaire. Scores out of 40 representing each stage of team development were calculated as shown in Appendix M as a measure of how participants perceived the developmental stage of their team. A score of 32 or higher indicates a clear perception by the respondent that the team is in that stage, and a score of 16 or less it is not in that stage.

No scores indicated the first stage, Forming, characterising the initial orientation of a new group. This would be an interesting environment in which to test the workshop. As a means of helping team members get to know each other and start to understand how they will approach the task ahead, it may be useful. But this stage is also one in which team members are still orienting themselves, and may be anxious (Mind Tools, n.d.) or cautiously polite.

The second stage, Storming, was the only one to have scores indicating that the team was not in that stage (one respondent at Company B and one at C). In the Storming phase there is "conflict and polarization around interpersonal issues, with concomitant emotional responding in the task sphere. These behaviors serve as resistance to group influence and task requirements." (Tuckman, 1965, p.396). One respondent's responses moderately indicated this perception at company A, scoring 24 for Storming. Had there been a consensus on this stage, any failure of the workshop would have to be evaluated in light of this finding. Similarly, a successfully constructive discussion in more cohesive environments cannot be ascribed to the workshop alone. Evaluating the contribution of the workshop format to a Performing team depends on participants' perceptions of its value.

Three participants' scores indicated the third stage, Norming, in which "ingroup feeling and cohesiveness develop, new standards evolve, and new roles are adopted. In the task realm, intimate, personal opinions are expressed." (Tuckman, 1965, p.396). This is a suitable environment in which to evaluate the workshop as participants in this stage are willing to express their opinions and the material could be helpful to the evolution of team standards.

Most participants believed their team was in the Performing stage — "the fourth and final stage ...Roles become flexible and functional, and group energy is channeled into the task. Structural issues have been resolved, and structure can now become supportive of task performance" (Tuckman, 1965, p.396). This maturely functioning environment is one in which the workshop can be evaluated as a tool to support the team in reflecting on their practice and focusing their discussions.

The results are summarised in Table 11; numbers of respondents showing a clear perception of a stage (i.e. score greater than 32) are shown in bold. Guidance for interpreting the questionnaire scores (Clark, 2016) is vague: having "only a small difference" between scores for each of the four stages can occur during the volatility of Storming but Clark does not define "small". Apart from one participant at Company D (forming 27, storming 26, norming 27, performing 29), there was a minimum difference of at least 6 between any one respondent's highest and lowest stage scores. Given the strength and consistency of the Performing responses, results are reported here on the assumption that 6 does not constitute a "small difference".

Company	Forming	Storming	Norming	Performing
A	0	1	0	3 + 3
B	0	0	2	3 + 0
C	0	0	0	4 + 4
D	0	0	1	2 + 1

Table 11: Number of respondents with clear or predominant perception of each team stage

7.2.2 Participant demographics

All 24 participants completed the profile questions (Appendix L). Descriptive statistics for their work profile are shown in Table 12. Figure 15 shows that a range of experience was represented, from software developers just starting out to those with a long career behind them. One participant commented that team sizes at company B are dynamic, with people being moved between teams depending on the needs of the projects.

Profile	Mean	SD	Median	Minimum	Maximum
Experience (years)	11.58	8.90	10.0	<1	30
Previous companies	2.5	2.57	2.0	0	10
Current team size	7.62	2.24	8.0	2	10

Table 12: Work profile statistics of workshop participants

The questionnaire used a free-text question when asking participants to describe their role. The majority, 83%, identified with a technical role directly associated with software development (developer, software developer, software engineer or quantitative analyst). The roles that participants reported are shown in Table 13. These are matched by the activities that account for most time in participants' current jobs, with 92% of participants citing some aspect of software development as their main activity (Table 14).

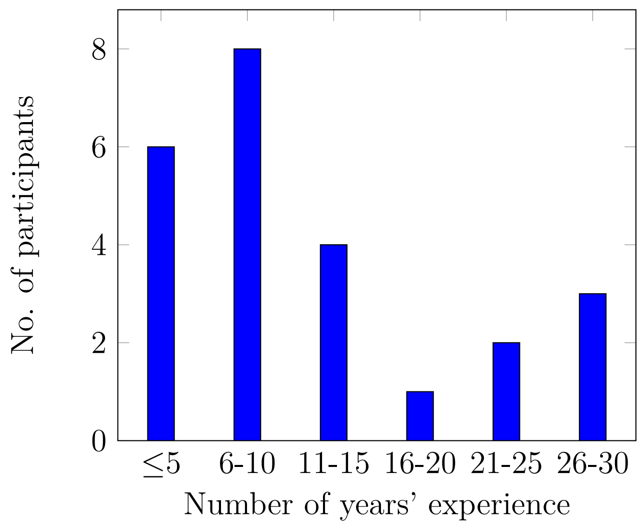


Figure 15: Workshop participants' experience in software development

Two identified in leadership roles but it was evident from the discussions that all participants were closely familiar with details of the team's software; "manager" did not signify a participant at any remove from the day-to-day technical minutiae.

Role	Qualifiers	Subtotal	Total
Developer	Senior	1	1
Software developer	<5 years experience	1	8
	Senior	3	
	—	4	
Software engineer	and scrum master	1	7
	Graduate	3	
	Junior	1	
	Principal	1	
	—	1	
Quantitative analyst	—	4	4
Web developer	—	1	1
Consultant	Senior	1	1
Software team leader	—	1	1
Manager	—	1	1

Table 13: Work roles of workshop participants

These statistics show that the workshop was delivered to the audience of software development professionals for whom it was intended, and that maintenance and adapting existing software play a substantial role in their jobs.

Main activity	No. of participants
Software development:	
System analysis/design	1
Writing new software	9
Adapting existing software	11
Maintenance	1
Other:	
b2b technical support	1
researching/learning a technical specification or piece of equipment	1

Table 14: Main work activity of workshop participants

7.3 Feedback and observations on workshop procedure

An important objective for the workshop procedure was that it should be a practical tool which software developers can use. This was assessed mainly by feedback from the participants. Their feedback on the logistics was collected in a questionnaire immediately after the workshop and is reported in this section, supplemented by the researcher's observations.

First impressions about the usefulness of the workshop were collected in the same questionnaire, followed up a month later with a final online questionnaire. The feedback showing how useful participants found the workshop is reported in §7.4.

7.3.1 Participant feedback on procedure

The feedback form completed by participants immediately after the workshops (Appendix N) asked about how the procedure had gone so that their impressions could be used to evaluate the design. These were reviewed after each workshop and used to refine the process for the next one. The quantitative results are listed in full in Appendix R. The comments that accompanied the quantitative answers to each logistics question are summarised here and for ease of reference the corresponding question text is included (each labelled to indicate whether it came from the Logistics section of the form or from the later Reflection in the follow-up).

Instructions

Logistics Q1. Was it clear what you were supposed to do?

On the whole participants found the instructions understandable, with 87.5%

deeming them "reasonably clear" or "very clear" (Table 15). Three participants wrote comments to elaborate on their answer, explaining what they had not understood at first: that they were to discuss the cards (company A); "that you can pick quotes that resonate with you negatively" (company B); that it wasn't clear they could have a "shortlist" (a problem specific to the variation of the procedure at company B). Apart from the experiment with a different card format at company B (see §6.6.4), refinements to the instructions were the only changes made to the procedure. It is unclear from the data in table 15 (in chronological order from first to last workshop) whether this made much difference to instructions which were at least reasonably clear to most people from the start, although it did help with the researcher's confidence in delivering them.

Company	Very unclear	A little unclear	Reasonably clear	Very clear	% clear
A	0	0	3	4	100
B	0	2	1	2	60
C	0	0	2	3	100
	0	0	1	2	100
D	1	0	0	3	75
Total	1	2	7	14	87.5

Table 15: Workshop participant feedback on the clarity of the instructions

Conduct of discussion

Logistics	Q2.	Did you get a chance to contribute?
Logistics	Q3.	Did anyone who rarely speaks up share their thoughts?

Asked about the opportunity to contribute to the discussion, all 24 participants reported that they had had their say. Those who considered the question applicable to their group reported that people who rarely speak up had said more than usual (42%) or even been equal contributors (33%). The 25% of participants who answered this question "not applicable" were distributed across the companies: one or two people at four of the five workshops. One participant expanded on their answer, suggesting that the facilitator promote contribution to the discussion from people who have been quiet. This was apposite because 2 of the 7 participants had spoken little, if at all, except when explaining their card. In subsequent workshops the facilitator not only made sure everyone explained a card choice but also brought quieter participants into the conversation, although at other companies most seemed to chip in at least a little without prompting. The results suggest that, although some were noticeably quieter than others (see §7.3.2), the workshop had a degree of success in its goal of facilitating discussion among all participants.

All participants reported that the discussion either stayed on track most of the time

Logistics	Q4.	Did the discussion remain focused on what has most impact on people?
Logistics	Q5.	How was it to have an external facilitator rather than someone from your organisation?

(58%) or quickly got back on track if it had strayed off-topic (42%). An external facilitator was considered useful: 54% found this very helpful and 38% somewhat helpful. A participant comment explained one aspect of this: "We're not thinking in these ways!". While none found it actually unhelpful, 8% said it made no difference. A participant who reported facilitation "somewhat helpful" observed that someone from within the company would "have the ability to make improvements". This is discussed in §7.5.

Materials

Logistics	Q6.	Did all the cards make sense? If you can recall them, please list any that didn't.

The cards that were chosen for discussion are listed in Appendix Q. The feedback shows that on the whole the quotations were clear to participants, with 63% explicitly reporting that yes, all the cards made sense and 17% leaving the answer blank. The remaining 20% listed some cards as confusing (Table 16), mostly citing only one or two cards but one person citing several (4, 6, 13, 19 and 20), unfortunately without elaboration. Of the cards which some considered unclear, several were chosen for discussion by other participants; the frequencies are shown in Appendix Q.

With some exceptions concerning specific phrases, the use of quotes from experienced developers proved a viable means of offering topics for discussion, often a particularly salient and enlivening one (see §7.3.2).

Procedure

Asked which aspects of the procedure they would keep, 29% of participants responded that they would keep procedure just as it is. There were positive comments about the breadth and relevance of the card content. For example:

> "A lot of the cards I could relate to but never really thought about. So bringing them up when I feel like that again would make for interesting convos with colleagues."

People liked the open and inclusive discussion in which everyone got to explain a card of their choice. The procedure was also praised because the cards prompted overdue discussion:

> "Sometimes we all know about the issues but we don't discuss/address them."

These findings echo the feedback about the conduct of the discussion: the workshop achieved its goal of helping participants to identify and talk about important issues, including some that they related to but had not previously discussed.

Asked which aspects of the procedure they would change, 58% explicitly answered "none" or left the answer blank. The changes suggested by the others mainly concerned the constraints imposed. There were proposals to:

- Include additional card topics. ("management/ project schedule/ scope")
- Select just one topic in order to delve deeper. ("the hour was over quickly")
- Allow people to suggest ideas instead of depending completely on the cards.
- Vary the quotes used from time to time

Evidently the participants saw future potential in the process (also see §7.4) as these ideas suggest a desire to refine it to address local needs. This was further evident in the comment of a participant who proposed a more concrete outcome to take forward afterwards:

> "Take notes for us. Would there be benefit in structuring discussion more? Questions/ Answers."

Practical issues were raised only at company A, where the comment concerned making sure everyone sees all the cards. All participants working with a communal set of cards spontaneously shared them around, but this was not very easy at company A due to lack of space.

Negativity was a concern for 2 participants. One observed that it was hard not to say something negative about an individual's behaviour. Another suggested that negatives should be balanced out by asking people to pick two quotes, one positive and one negative. Although it was not often mentioned, the issue of valence is a potential threat to the goal of facilitating an open but relaxed discussion. Possible solutions are discussed in §7.5.2.

7.3.2 Researcher observations on procedure

Behaviour was observed and notes taken (at the time or shortly after the end of the workshop) about how participants went about deliberating over the cards and discussing the ones chosen. Two aspects were highlighted: the care with which participants made their card choices, and the manner in which they quickly settled into peer-to-peer discussion.

Choosing

The observations suggested that participants were engaged with the process and taking their decision seriously.

At company A, where table space was limited, participants spontaneously arranged the cards linearly so that all the text was visible. Half way through their deliberations they carefully swapped cards around from one end of the table to the other so that everyone got the opportunity to see them all. At subsequent workshops there was, at the researcher's request, plenty of table space so the careful alignment was not necessary. There was still shuffling around of the cards as people engaged with looking through them all and ensuring others got to see them all.

Across all the different workshop sessions there were common responses to seeing the cards. As participants began to read there would be an exclamation as someone spotted a familiar and personally resonant theme, and perhaps mentioned aloud the name of a local component that they recognised in the description. More than once there was very animated recognition as a participant caught sight of a resonant card and pounced on it.

Some participants had a clear moment of decision where they settled back decisively with their chosen card, others wrestled with the final choice between a small number of candidates, but all appeared to make their choice thoughtfully. The objective to facilitate thoughtful reflection appears to have been achieved, at least in choosing topics; as will be seen in §7.4, this also extended to the discussion.

Peer to peer discussion

One of the goals of the workshop design was to create a technique of practical use to software developers. Though knowing the workshop was part of a research project, participants quickly settled into the kind of peer to peer discussion it was intended to facilitate.

All groups started by directing their explanation of card choices to the researcher, but it rapidly became a conversation with each other as they engaged in discussion of local issues; participants frequently applied the card to a real-life, current situation, using names with which the researcher would not be familiar — rather than being self-conscious research participants they were talking naturally among themselves. This is an extract from one such natural conversation:

> "I think [name]'s point of sometimes yes, you have to write something shit first …You've got to write the thing first and then you can actually understand what you were trying to do, you can be like oh, I don't actually need half of this maybe …"
>
> "And you haven't got the opportunity I think, at work, to do that so often and that's why you get a lot of shit in …"
>
> "This code looks like a five year old drew it."

Groups also explored what they could do about their specific local issues, sometimes when asked this by the researcher but often spontaneously raising possible solutions. In some cases issues appeared to be long-standing bugbears with no simple solution within the scope of what the team themselves could do. In others, teams reached conclusions about realistic practical actions they could take. This extract is one such example from a workshop in which several participants analysed an issue that causes them ongoing problems and agreed how they could solve it:

> "There are ways of avoiding this [problem] …we don't currently really do that at all …there are things that can be done to avoid that …we should put things in place."

Engagement

Some participants were quite garrulous. Others spoke infrequently, but contributed when drawn in to the conversation by the researcher. None appeared disengaged or resistant to speaking. The goal of everyone having a say was at least partially achieved, though how much this was due to having a facilitator and how much to the format is impossible to say.

Card	Text	Comments (where given)	Chosen
4	Until you've actually investigated a bug report that someone else has written you don't realise how easy it is to write bad ones	1 participant	✓
6	If someone makes it harder to approach then you're forced to choose less frequent questions	Missing clear context. Does it mean verbal? If someone makes it hard to approach — is it that someone is unapproachable? +1 without comment	
13	Someone not reinventing the wheel? Cor! Who'd have thought it!	2 participants	✓
15	People get very dogmatic. If you disagree or voice concerns you're a heretic and you're burned at the stake	Only because I don't know what dogmatic means	
17	Finding the same block repeated several times makes it a lot harder to understand ...you're trying to mentally diff	I didn't understand	
18	I think you do need to discuss stuff with people	Missing clear context. "Stuff"?	✓
19	If you have an interface that is natural and obvious then it leaves you freer to get on with your job. It does what it says on the tin.	1 participant	✓
20	Identifiers are very, very important	1 participant	
?		One or two were a little confusing	

Table 16: Workshop participant feedback on the clarity of the cards

Logistics	Q7.	What aspects of the procedure would you keep?
Logistics	Q8.	What aspects of the procedure would you change?
Reflection	Q6.	Is there anything you would change about the workshop format?

7.4 Feedback on usefulness of the workshop

All 24 participants completed the written feedback form (Appendix N) presented immediately after the workshop. The online follow-up survey (Appendix O) four weeks later had a 63% response rate.

Responses to both questionnaires provided constructive feedback about the workshops, including explanations of what it was people had particularly found useful and the minor modifications they would suggest. Participants took time over completing the paper forms (up to ten minutes), frequently expanding helpfully on the answer rather than simply answering yes or no. The detail of the responses to the online questionnaire suggests a similar diligence. Perhaps some of this diligence is due to a desire to be as helpful as possible with the research, but the constructive nature of the comments also suggests that they considered the workshop worthwhile.

This section summarises their evaluation of the workshop across both the immediate post-workshop questionnaire and the later follow-up. For ease of reference the corresponding question text is included (each labelled to indicate whether it came from the Usefulness section of the immediate feedback form or from the later Reflection in the follow-up).

7.4.1 Prospective reuse

Usefulness	Q1.	Would you do the workshop again (e.g. to continue the discussion or talk with a different group of people)?
Reflection	Q4.	Would you consider using the workshop format again in future (e.g. to continue the discussion, or with a different group of people)?

Both questionnaires asked whether participants would consider using the workshop again themselves. Immediately after the workshop, 88% said yes, with one “maybe”, one “no” and one blank answer. A month later the response remained positive, with 93% of respondents (58% of all participants) saying yes and one answering “potentially”.

Some specific reasons were given for reusing the workshop. Conducting it with a different mix of people was a recurring theme. Some people reported that they already have quite open discussion within their teams, but others felt that the process could be useful to help “get everyone on the same page”, whether at the introduction of new team members or as an occasional review. For example:

"It could be useful to run the workshop again in conjunction with another team to improve communication and consensus building."

The tenor of the comments was that they were keen to have discussion such as had occurred in the workshop but perhaps lacked a structure to help them do so effectively. For example:

"As a team we have no rituals which attempt to improve consistency."

Finally, two participants felt that the breadth of the discussion within the time meant it had to be relatively superficial and that it would be beneficial to use the workshop as a platform to explore one selected topic in depth.

These very positive results suggest that the objective of creating a useful process to facilitate communication about software development was achieved. The ways in which they would use it are specific to local needs such as on-boarding, inter-team communication or in-depth team review, but for whatever application the great majority of respondents saw a future use for it.

7.4.2 Broaching discussion

Usefulness	Q3.	Did the discussion cover topics which are rarely talked about in depth or can be difficult to raise?
Reflection	Q3.	Have there been informal discussions prompted by the workshop?

With the contents of the discussion still fresh in participants' minds, the immediate post-workshop questionnaire asked whether the discussion had covered topics that are rarely talked about or can be difficult to raise. A common response was that the topics are not new or controversial; they are known but not necessarily addressed. The tenor of the comments was not so much that the issues are difficult to raise — many reported that the team talks quite openly — but that the opportunity does not normally arise.

"Mostly issues are known but rarely discussed openly."

"We rarely have time to talk about such issues."

> "Often talk is focused on just the work to be done and not how to go about it."

The workshop appears to have provided a space for discussion but not engendered much subsequent discussion. Asked in the later follow-up questionnaire whether there had been informal discussion prompted by the workshop, 47% of respondents (29% of all participants) said no. A similar number (40% of respondents, 25% of all participants) reported that informal discussions, for example at lunch or around the coffee machine, had happened in the days immediately after the workshop. 13% of respondents (8% of all participants) reported efforts to address work practices as a result of the workshop:

> "The topics discussed in the session have been raised a number of times since, and action is now being taken."

> "Yes — a number of discussions on how we work."

So although some have taken things forward, these numbers correspond with comments reported in §7.3.1 about making a more concrete outcome a part of the procedure. This is discussed in §7.5.

7.4.3 Behaviour change in self and others

Usefulness	Q4.	Are you considering doing something differently in your work after the discussion today?
Usefulness	Q5.	Do you think any other participants might now consider doing something differently?
Reflection	Q1.	Have you noticed yourself think or do anything differently since the workshop?
Reflection	Q2.	Have you noticed other participants do anything differently since the workshop?

In the immediate post-workshop questionnaire participants were asked if they were considering doing something differently as a result of the discussion. At this stage there were similar numbers of yes (42%) and no (46%) answers, with the remainder responding "possibly" or "not sure". Half of the "yes" respondents gave more information, all citing an intention to communicate more. For example:

> "Ask more questions/interact more with team members."

> "Talk more, track decisions."

The later follow-up questionnaire asked whether participants had actually noticed themselves thinking or acting differently in the weeks since the workshop. Several said they had (40% of respondents, 25% of all participants), reporting greater consciousness of issues, taking others' perspective into account and being more communicative. For example:

> "Changed my perspective …in a way that better allows me to engage in healthy debate."

> "More communicative with team members about complex changes that may affect them."

The remainder felt they had not made any changes (60% of respondents, 38% of all participants), sometimes sounding somewhat rueful:

> "I don't think I've reflected enough on the comments made during the session."

Fewer participants in the immediate post-workshop questionnaire thought that other participants might now be considering doing something differently, only 25% responding yes and 33% no. The remainder responded "possibly" or "not sure" (17%) or gave no answer (25%).

When it came to the later follow-up questionnaire, several participants reported changes they had noticed in themselves but few reported noticing any difference in others. Nearly all (80% of respondents, 50% of all participants) answered no to this question, sometimes qualifying their answer by saying that they had not paid attention to noticing. 13% of respondents (8% of all participants) reported that it was difficult to say. For example:

> "We have had more discussions around testing recently, how much of that is related to the workshop and how much to other changes in the team is hard to tell."

One participant did report noticing a change in others' behaviour:

"Better communication and awareness of each other's needs."

This supports the conclusion of §7.4.2 that some concrete follow-up is needed to avoid the outcome being "well, that happened" and moving on. The participating teams were already largely open and communicative — for example, "We feel everyone can contribute to the ongoing conversation already" — so workshops were not necessarily doing anything new in enabling discussion. But it is clear from the responses reported in §7.4.2 they did open up topics in need of discussion and follow-up.

7.4.4 Prospective recommendations

Usefulness	Q2.	Would you recommend others to try it? For what reason?
Reflection	Q5.	Would you recommend others to try it?

Both questionnaires asked whether participants would recommend the workshop to others. Immediately after the workshop, 96% said yes. The one participant not responding with an unequivocal positive replied "tentatively", explaining that they were not sure what the benefit had been for them personally. A month later the response remained positive, with 93% of respondents (58% of all participants) saying yes and one answering no without further explanation.

The reasons given for recommending the workshop fell into two main themes: the catalyst it provides for useful reflection that does not usually occur and the insight it gives into others' perspectives, both within the team and (given the source of the topics) within the industry. For example:

"Forces you to step back and evaluate processes/how you work."

"Stimulates interesting discussions within the team, not usually/traditionally talked about."

"It helped to see issues from both sides."

"It was beneficial to see the concerns and ideas raised by others in the industry."

There were also suggestions that it would be good for problem teams to have this kind of discussion, while acknowledging that the discussion is most likely to work in "open and trusting" teams. Even with the performing teams who took part, one participant explained their recommendation in these terms:

> "Discussion is healthy and the facilitator helped it remain healthy."

Participants were also asked if there was anything they would change about the workshop format. It appears that the recommendations described above were largely for the workshop 'as-is' since few proposed any change to the format. Those who proposed changes (13% of respondents, 8% of all participants) suggested a greater clarity of instructions (understandably from company B; see §6.6.4) or a "focus on one or two areas", which may again reflect the desire for a more concrete outcome.

Of those who proposed no change (87% of respondents, 54% of all participants) three expressly commented on positive features: the length ("about right, given the number of attendees"); their enjoyment of the workshop; and the role of the facilitator:

> "The key for me was having someone facilitate a discussion. As developers we don't do enough personal development (ironically ...)."

7.4.5 Suggestions for other uses

Usefulness	Q6.	Could the material be helpful in recruitment (e.g., to learn more about an interviewee)?
Usefulness	Q7.	Could the material be helpful in appraisals (e.g., 360-degree feedback or self-assessment)?
Reflection	Q7.	Have you had any thoughts about other potential uses of the workshop materials (e.g. recruitment, appraisals)?

The workshop was presented as an opportunity for peers to discuss issues in their workplace, but the material could also have potential to be used to explore individuals' understanding of team-friendly practices. Participants were therefore asked, in both the immediate post-workshop questionnaire and the later follow-up, whether the material could be helpful in recruitment, appraisals or other uses.

Immediately after the workshop, participants gave positive responses to the idea of use as a recruitment aid, with 38% giving a clear yes (e.g., "almost certainly"; "yes definitely") and 54% answering with some variant of "possibly" and an explanation of why it would be useful or the caveats that might apply. There was just one outright "no" and one participant entered no response.

Suggested benefits included assessing a candidate's compatibility with the team ethos. For example:

> "Could help get insights into how interviewee would fit with team working."

> "Some interviews concentrate on language knowledge and not enough on software development good practice."

Those who saw potential recruitment use but expressed concerns cited practicalities (length of interview; dependency on an appropriate company culture) or the difficulty of getting a meaningful response from candidates due to nervousness or a deliberate attempt to give desirable answers. One participant observed "they are generally being positive to the point of lying!".

Support for the idea of use in appraisals was still more positive, with 58% of participants saying that they saw potential, in one case responding "yes — please create a form!". Where reasons were given, participants mentioned the value of the process in fleshing out issues, "getting you to think outside the box" and being a thought-provoking prompt for self-evaluation. 21% were unsure, one person explaining their uncertainty about how the group exercise would transfer to an individual context. Only one answered with an outright "no".

The balance of responses was slightly less positive in the later follow-up questionnaire, with a greater number of respondents (20%, 13% of all participants) giving an outright "no" to the question about other potential uses. One further participant was unsure, citing the difficulty of getting a meaningful response in either recruitment or appraisals:

> "Concerned you would get the safe answer rather than what a person really thinks?"

Nonetheless the overall tone was still positive. Some respondents (20%, 13% of all participants) reported that they are not personally involved in running processes such as recruitment and appraisal, but speculated that some of the material could be beneficial. The responses which straightforwardly suggested further potential (40%

of respondents, 25% of all participants) included uses for recruitment, appraisal or both, although in one case appraisals were excluded because the process would not fit the company's existing structure. Several of these participants extended the appraisal idea to encompass more of a group retrospective for reflecting on practices within the team as a whole:

> "Almost like an appraisal, but a group session yearly might be a good idea."

> "Similar ideas for coding retrospectives."

During informal conversations after the workshops participants at two companies also proposed potential use of the workshop for inter-team communication, seeing it as an appropriate way of facilitating dialogue in what appeared to be a somewhat vexing lack of mutual understanding.

7.5 Discussion

This section reviews the characteristics of participants in the workshops to evaluate how well it reached its target audience (§7.5.1). The logistics and conduct of the workshop procedure are then discussed to consider how the process worked in practical terms (§7.5.2). Finally the workplace outcomes of the workshop as evaluated by participants are considered (§7.5.3). The conclusions show some very effective elements which encourage using this approach further to facilitate team discussions, and also exploring its application to different scenarios.

7.5.1 Participant characteristics

Demographics

No specification was given to organisers regarding the composition of the workshop groups, but in the event all groups comprised people who worked together. This was evidently a factor in the nature of the discussion since they often linked the card topics to specific local issues familiar to the team (see §7.5.2). At some companies there was a suggestion that the workshop process could also be useful for inter-team communication. This is considered further in §7.5.3.

The terms participants used to describe their roles, especially when also specifying a level of seniority, often had the air of official job titles. It would have been interesting to ask for both the job title and how they would describe themselves since the list of main activities shows a greater commonality than company job titles might suggest.

Describing the work they do, fully half of the participants listed an activity explicitly involving work on existing code ("adapting existing software" or "maintenance") as their primary activity. All but one (a graduate software engineer with less than a year's experience) listed "adapting existing software" as either their main activity or one of the activities undertaken during their career. These results concur with personal experience, the comments of interview participants and sources such as Verhoef (2000) and Glass (2003) to illustrate that that the nature of software development is, as discussed in Chapter 2, at least as much about evolution as it is about creation.

Team stage

Spontaneously, five participants volunteered information about the team stage questionnaire they completed prior to the workshop (see §3.4.2). They cited compound questions and other ill-phrased questions that were awkward to answer. Some questions conflated two independent items, making it difficult to answer when the respondent agrees with one and disagrees with the other. For example: "There are many abstract discussions of the concepts and issues; some members are impatient with these discussions".

The questions about processes and procedures were also mentioned, since the import of these questions for a software team depends on the level of detail. A participant reported that their team has agreed broad principles to ensure that "our work can be combined to produce a coherent whole". But the team is not prescriptive about the fine-grained details of how to do it. As the participant observed: "after all it takes a thinking mind to produce working software."

Another participant reported that for a few (unspecified) questions, none of the available answers really applied. They also found some questions difficult to answer in respect of the team as a whole. For example, "We express criticism of each other constructively" does not have a clear answer when there are individual differences between individual team members.

With hindsight, a better instrument might have been found. Nonetheless, on the whole the questions were appropriate to the goal. The questionnaire was intended not as a precise measurement but as a litmus test of the context in which each workshop took place: a precautionary measure to ascertain whether the environment was healthy, lest a troubled environment confound the results by being unpromising ground for constructive discussion in any form. The results were consistent at this coarse-grained level with the researcher's own impressions at each company and the questionnaire served its function. Clark (2016) commented that it is intended as a learning tool for use in training programs rather than being a validated research instrument, and has been developed using feedback from just such applications. It is evident from a web search for "Tuckman survey" that numerous consultancies and organisations find it useful and its use is widespread; it was even mentioned in passing by a participant in the Exploratory Study.

Nonetheless a validated measure of team climate might, in light of the feedback, have been a better choice. Developer eXperience (DX) concerns the quality of working life as a software developer. Easton and Van Laar's (2018) Work-Related Quality of Life (WRQoL) scale, an evidenced scale of workforce experience that is applicable across organisational settings (e.g, Van Laar, Edwards, & Easton, 2007; Edwards, Van Laar, Easton, & Kinman, 2009) would now be the preferred instrument. At 23 items it is less demanding than Tuckman, and rather than giving a single measure of an overall state it identifies 6 factors affecting quality of working life: General Well-Being (GWB), Home-Work Interface (HWI), Job and Career Satisfaction (JCS), Control at Work (CAW), Working Conditions (WCS) and Stress at Work (SAW). This gives a more nuanced picture in which aspects particularly influenced by individual circumstances (GWB, HWI) could be considered separately from workplace circumstances (JCS, CAW, WCS, SAW) which can affect the whole team.

Another possibility is the Team Climate Inventory (TCI, Anderson & West, 1998), which "attempts to assess the extent to which team members share information, listen to one another, give each other feedback, and strive for high performance." At 61 questions it may be too much for busy volunteers to stomach, but it could perhaps be condensed by focusing only on the most pertinent of the five climate factors it measures. "Participative safety" stands out as a crucial factor for the climate in which a workshop takes place. The applicability of the other factors ("interaction frequency"; "support for innovation"; "task orientation"; "vision") is less clear-cut.

As assessed by the team stage questionnaire used, the teams taking part were, in the terms coined by Tuckman (1965), "performing". The characteristics of teams in this stage of development include “constructive self-change” and “ability to prevent or work through group problems” (Clark, 2015), features which were also suggested by participants’ feedback (see §7.4.2). This observation raises the issue of self-selecting participants, which was a concern going into the research. The initial motivation to conduct it came in part from experiencing the behaviour of teams in less advanced states of development, but self-selection meant that the research was likely to attract healthy teams rather than those in the throes of “storming”.

The findings do show that already well-functioning teams found the material useful; they appear to have used the workshop discussion as part of their “constructive self-change” (Clark, 2015). It is welcome to see that the work has found practical application in this context of making good better. Classic works for software teams include “Death March” (Yourdon, 2004), “Peopleware” (DeMarco & Lister, 1999) — an altogether less gloomy sounding title, but its first chapter is “Somewhere today, a project is failing” — and “Beautiful Teams” (Stellman & Greene, 2009), a book of “inspiring and cautionary tales”. While there is no doubting the need for cautionary tales and advice on rescuing software projects from the mire, there is limited material available to inspire performing teams seeking continuous improvement in their processes, though plenty (e.g., McConnell, 2004; Goodliffe, 2006; Oram & Wilson, 2007; Henney, 2010) to help them craft better code.

7.5.2 Workshop procedure

Instructions

One of the reasons for including the researcher as a facilitator for each workshop rather than making it a self-directed activity was to ensure that the instructions were followed. The introduction was refined as the study progressed to address observations in the field notes and feedback forms (e.g., ”not entirely clear to discuss the cards”) but the differences in reported clarity of the instructions were within rather than between groups. The workshop at company B understandably got the lowest score for clarity due to an insufficiently thought out change to use individual packs of A6 cards rather than a communal A4 set (see §6.6.4; this approach also reduced the interactivity of the procedure and was not used again). But across all four other workshops just one participant considered the instructions unclear. However there is evidently room for improvement to deliver instructions that are not just “reasonably clear” but “very clear” to everyone in the room. Pace may have been a factor here, with the researcher conscious of the generosity of companies allowing time for the workshop and the evident curiosity of the participants to see what the cards on the table were all about. A written summary could also be considered so that participants have a reference if their attention drifts and to cater for people who may prefer written instructions to verbal ones.

The lack of clarity of purpose in the early versions of the instructions may have contributed to some participants' thoughts about using the workshop again. A respondent from Company A reported in the online follow-up questionnaire that "it wasn't 100% clear what the purpose was. My understanding was that it has been for research but the follow up questions imply it was for a benefit to the workplace". The question about reuse makes sense only for the latter purpose, which may perhaps have contributed to the participant saying they would not recommend the workshop to others. Yet they reported that they would themselves consider using it again in future and described it as "good for inspiring discussions". The relationship between these somewhat contradictory answers can only be speculated at, but nonetheless prompts reflection that the point of a process needs to be clear before asking participants about their intentions to repeat the process.

Emotional valence of quotation cards

Whether because the original participants more often expressed their opinions in terms of negatives, or simply because it produced the more pithy, eye-catching quotations when they did so, there were more broadly negative statements (e.g. "Anyone who thinks it's 'their' code is missing the point") on the cards than positive/neutral ones (e.g., "Automating's beautiful"; "Identifiers are very, very important"). Workshop participants' choices included both types about equally, but a few participants commented on negativity. One suggested that people could be invited to select two cards: one that they find negative and another that they find positive. Another participant reported that they chose "Automating's beautiful" particularly because it was a positive one among many negatives.

Whether or not there are quotations to represent them, in theory most of the topics could be expressed in either positive or negative terms — although it is not clear whether the two sides are actually symmetrical, e.g., are bad behaviours generally more annoying than good behaviours are welcomed? The results of the card sorting task in the Exploratory Study's interviews (§5.2) may suggest otherwise. From a set of cards balanced for positive and negative valence, more than twice as many positive ones were singled out as important than negative ones, even in a confidential one-to-one discussion where no one would infer personal criticism. An interviewee in that study also spoke about the benefit of noting positives as well as negatives.

The perceptions of negativity invite consideration of balancing the quotation cards for emotional valence just as the interview cards were (§4.3), or even selecting only positively phrased quotations. The intention is, after all, to encourage a constructive discussion, and positive wording might help to defuse the tension felt by one participant that it is "Quite hard sometimes to not say something negative at someone when trying to be anonymous!" As there is a negative flip side inherent in each positive card perhaps the implied criticism is still present in the implication that someone is not doing the positive thing on the card, but the wording is more constructive. Conversely a negatively phrased set, while also a possibility, seems likely to threaten the constructive nature of a discussion that is expressly designed to minimise feelings of personal criticism.

Resonance of quotation cards

The phrases on the quotation cards were intended to convey styles of behaviour that were familiar to participants. The presence of the topic on a card was in itself an invitation to talk about it, and representing it in the form of a quotation from peers in the industry was intended to offer an element of social proof (Cialdini, 1993), reassuring participants that the issue also matters to others and thus persuading them to contribute their own experience. Precautions were taken (involving a developer in finalising the choice of quotations and piloting the workshop; see §6.6.2) but there was an element of risk in taking more colourful natural speech rather than a more formal description to represent the topics. Would they convey enough information?

Together, the variety of cards selected during the workshops and the information given on feedback forms showed that with a few exceptions, the statements on the cards did indeed make sense. Only 6 of the 21 cards went unchosen throughout any workshop (see Appendix Q), of which 4 were identified as unclear in the feedback. But how well the cards conveyed familiar scenarios was best demonstrated by the response of participants upon being invited to look at the cards. It was not long before someone would catch sight of a card and swoop on it with some kind of exclamation, often identifying it with the name of a local issue. This reaction became so familiar that the researcher mentally named it "the Pounce"'.

Group engagement

The relating of card topics to local issues that began with comments made while choosing cards continued within the discussion that ensued after everyone had chosen. Often participants began by addressing themselves to the researcher, but usually this turned quickly and naturally into a discussion among the participants as they talked about a familiar local issue. Having facilitated the discussion by providing the materials and starting the session, the researcher's role was frequently that of an observer at a team meeting. Occasionally there was an intervention to invite a contribution from a quieter participant, a request for clarification, or a question about how an issue might be resolved but in essence it was a team, talking.

As discussed in §7.5.1, the teams taking part were "performing" teams. As such it would seem unlikely that they need much facilitation in order to communicate openly and readily. Yet more than half of participants deemed the presence of an external facilitator "very helpful". A comment from one participant may partly explain this: "We're not thinking in these ways!". The willingness to communicate openly and constructively was already there, but the workshop may have served as a catalyst to help them step back from their daily routine of solving the problem that stands in the way of the next step and instead pause and reflect more broadly and deeply.

7.5.3 Usefulness of workshop

As discussed in §7.5.1, workshop participants were people who worked together and this was reflected in their discussions by their shared knowledge of the issues affecting them. Several participants' feedback suggested that the workshop format could also prove an agreeable vehicle for broaching inter-team communication, airing and reconciling views of the same system from different perspectives. As a measure to prevent isolation of teams, Coplien and Harrison (2004) suggest a "Watercooler" pattern — providing a common space where people from different teams can naturally congregate. A workshop is a less informal space, but it is still a space away from the demands of specific features in which wider conversation can unfold. Even within a team, it is both a physical and mental space in which to sit back and reflect.

7.6 Personal reflections of the researcher

Neither my age (similar to the most senior developers and some three decades older than the most junior ones) nor my gender (women are a minority in the software industry) appeared to be a factor in the Evaluation Study. There was a woman in the group on two occasions but I did not feel participants treated me differently for my gender at any of the five workshops. As for myself, I was not even conscious of my gender in these scenarios until prompted to consider it by writing my reflection. I have worked in the industry for a long time and it is not at all unusual to be the only woman present.

Participants in the Evaluation Study knew, whether from previous involvement in the research or from the information sheet, about my programming background. I did not mention it during the workshops since it was less relevant in this situation, with the conversation intended to be among the participants themselves rather than with me. In my role as facilitator, my own experience in the field was useful in knowing when to ask "is there something you could do about that?" when participants were discussing a local problem.

In the final workshop this strayed into making specific suggestions, albeit phrased as questions — "would it help if you ...?". Once a programmer, always a programmer and thus by definition a problem solver. A simple, obvious idea for a potential solution is almost impossible to suppress, and in this instance I had reasons for particularly wanting my visit to be helpful to the company. Fortunately contributing these ideas did not change the character of the discussion; my suggestion fitted into the discussion as if it had come from a member of the team and was considered in the same way.

It felt at that point like I was an adopted member of the team — an equal contributor drawing on my expertise to contribute to the discussion. While inviting participants to suggest their own solutions is more appropriate to the process of facilitating their discussion, becoming a part of it temporarily suggested to me that the workshop does function as a catalyst for a genuine conversation and not just as an artificial exercise; during those moments I was briefly just another software developer, engaged in a real discussion of a problem.

The obsession that led me to the research in the first place did, however, affect my perspective in deciding how to evaluate the workshops. My goal was to make the results of the exploratory study of practical use to developers; this elicitation of developer perceptions of peer practices has considerable potential to help them in their work by identifying the behaviours that most help or hinder them. My personal impression of the workshops was that they helped bring such behaviours into focus for teams in an accessible and locally relevant way, but my formal evaluation of the workshops prioritised participant feedback over all else. Whilst this was needed to answer my question of ”does this help developers?” my interests as a developer somewhat overwhelmed my interests as a researcher, leading me to look at the outcome while neglecting the details of the discourse by which it was achieved.

7.7 Practical application to professional practice

Research Question 2 asks if the approach tested in the Evaluation Study can be used to help software developers in their practice. This section addresses that question from the perspective of the target audience, a professional software developer considering using the workshop themselves. There are a number of questions that they might ask:

- Was the workshop an enjoyable experience? (§7.7.1)
- What could I use it for? (§7.7.2)
- Could I learn from using it? (§7.7.3)
- How would I go about using it? (§7.7.4)

7.7.1 Was the workshop an enjoyable experience?

People who participated in a workshop would recommend it to others (§7.4.4). Most would also consider reusing the workshop format themselves; the numbers for this were slightly lower than the recommendations but the comments show no lack of enthusiasm for the process. For example, participants reported that it “was good fun”, “stimulates interesting discussions”, “forces you to step back and evaluate” and “would be interesting to develop further”. Some simply indicated a sense of not needing to repeat the process having done it, or at least not soon. There are also some potential refinements to the process which could make it more overtly useful, and not simply an interesting diversion, by producing visible practical outcomes for specific local needs. These are explained in §7.7.3.

A notable feature of participants' responses was that the workshop encouraged discussion of topics which were rarely discussed in depth (see §7.4.2). The impression given from their questionnaire comments and the nature of their discussions was that these included well-known bugbears, local examples of the topic on a card, that were perhaps frequently moaned about but never tackled. People would like to see these addressed but could not tackle them single-handed. The workshop is an opportunity to finally sit down and discuss such issues constructively, jointly identifying the most painful of them, understanding reasons for the situation, assessing potential solutions and agreeing on action to remove or reduce the pain.

7.7.2 What could I use it for?

Participants found the workshop a welcome opportunity to stop and reflect about local issues and practices in a way that does not seem to happen within the normal work routine (§7.3.1). Those taking part were already in quite open and communicative teams but although their workshop discussions were not necessarily raising anything entirely unknown to the team they were nonetheless relating and addressing systemic issues that had not received attention when competing with the immediate demands of specific tasks. For some participants it was an opportunity to reflect and articulate ideas for the first time.

The workshops were not merely a theoretical discussion of abstract examples of the possible impact software development behaviours. Participants rapidly recognised their own concrete local issues in the generic quotation cards. These were issues which appeared to be well-known to all present but had not been addressed. The workshop prompted teams to address together the reasons, consequences and solutions for them in a way that had not happened in their day-to-day conversation. This discussion included what was, and what was not, realistic to solve the problem. In some cases this simply led to everyone understanding why an issue was intractable and just something to be lived with. In others, examining it together revealed viable solutions.

Turner and Boehm (2003) noted that Agile methods "cultivate the development and use of tacit knowledge, depending on the understanding and experience of the people doing the work and their willingness to share it." This reliance is also illustrated by the "Social" element in the model of software developer tasks that emerged from the Exploratory Study (Figure 4). This research suggests that sharing of understanding and experience is an aspect of software development that needs improvement. The workshop could therefore be useful as a framework for retrospectives that facilitate a broader overview than might otherwise be the case. As Highsmith and Cockburn (2001) observed, face to face communication is faster than writing and reading documents. And indeed no participant called for more traditional documents, preferring the documentation inherent in the artefacts of software development that is characterised in Chapter 5 as ”chronicling”. But having fewer written documents means that it is all the more crucial to talk face to face.

Derby and Larsen (2006, p. xv) described Agile retrospectives as “a special meeting where the team gathers after completing an increment of work to adapt and inspect their methods and teamwork”, and presented practical and creative ways to elicit information using tangible tools such as sticky notes and coloured dots. The retrospective approach focuses, though, on events during the iteration (increment of work, also known as a sprint) — which is its valuable purpose, but one that does not necessarily prompt a wider holistic perspective on the team’s practices. A wider perspective seems more in keeping with one of the principles underpinning the Agile Manifesto: “At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly” (Agile Alliance, 2001b). This principle is listed independently of “Deliver working software frequently.” and the Agile Manifesto does not contain any suggestion that retrospectives should be tied to iterations.

Pikkarainen’s (2008) case study of communication within agile projects suggests that practices centred around iterations, whether prior planning or subsequent retrospectives, promote useful communication about the features in that iteration but can have negative effects on the longer-term overall goals. The study examined just two teams, but methods under the Agile umbrella vary widely and are not always clearly specified (Jalali & Wohlin, 2012); it would be difficult to generalise even from a much larger sample when there are so many variables in their practices. It is perhaps more helpful to consider individual practices regardless of the local interpretation of the Agile Manifesto: Vallon, da Silva Estácio, Prikladnicki, and Grechenig (2018) show that retrospectives come fifth in the frequency of agile practices employed in the many case studies they reviewed (44 cases compared to 70 for the most widely adopted practice, standup meetings).

Both Jalali and Wohlin (2012) and Vallon et al. (2018) concern geographically distributed teams, unlike the teams in the workshops. Enquiries about participating companies’ processes revealed that retrospectives are not universally or consistently practiced as described by Derby and Larsen (2006). One admitted “We do ‘retrospectives’ but only at the end of projects”, which would appear to have more in common with a project wash-up meeting than an agile retrospective, and too late to help the project. Another said they had "tried a retrospective — but we don't really have very well defined sprints at the moment." (note the singular, "retrospective"). Both would describe their process as agile. Exploring the breadth of practices followed under the "Agile" label is beyond the scope of this research but presumably these companies have not seen sufficient value, whether through experience or expectation, to warrant regular retrospectives.

The workshop seems to have given them something that their retrospectives have not. It might, with the refinements discussed below, be a viable alternative that allows them to discuss and address their practices at intervals; another participant evidently also saw room for improvement in retrospectives and explicitly suggested a future use of the format to take a similar approach in coding retrospectives (see §7.4.5). It also has potential as an occasional supplement for teams which do conduct regular retrospectives but want to step back to take a broader view; one participant suggested doing this annually or when new staff join the team. Their feedback about the workshop makes it clear that participants found the workshop experience useful.

Participants also had suggestions for using the workshop that went beyond discussions with their existing team (see §7.4.5), as a potential means of facilitating dialogue between teams. There is scope too to use the materials as a tool for assessment. The cards illustrate points that are of significance to others working in a team, so some participants thought the materials would be useful as a point of focus for appraisals, where they could be used to discuss how well the developer measures up on desired team-friendly behaviours. Similarly they could be used as a discussion point in interviews, this time not to ask someone how they fare on any given criterion but to form an impression of their attitude as a developer by asking them to select cards that resonate with them and explain why. This could be done either in an interview panel using just the cards, or in a workshop scenario where they can have a discussion with the team they would be joining so that both sides can assess whether they would "fit in". The latter could also be useful as an onboarding exercise to help introduce the norms of the team and ongoing issues they are facing.

An approach to using the cards in interviews has already been discussed in detail with an independent experienced developer who did not participate in the research; he has received the materials but at the time of writing had not yet had the opportunity to try them. His analysis (reproduced in full in Appendix S) is that the cards offer a conversation starter to get candidates to talk about the issues they run into and how they tackle them. Giving them choice from a selection of issues raised by experienced people could help to structure the interview around the candidate's own experience rather than the interviewer's and thus avoid wrong-footing them.

7.7.3 Could I learn from using it?

The workshop in its current format was successful at encouraging discussion. Sometimes the topic was a well-known one that had never been properly addressed. Upon addressing such topics participants were sometimes able to articulate a feasible solution, and in some cases they deemed it an intractable issue too difficult to solve within the constraints of the business. These were all ideas that teams could usefully take forward, whether to action, work around or simply move on with a common understanding.

What the workshop lacks is any procedure to support this ongoing change. Without some concrete follow-up structure the workshop risks being just a mild diversion, creating a little more mutual understanding but otherwise in danger of being perceived as unproductive. Therefore to really learn from using it, it could be useful to tailor it to local needs and add follow-up to the procedure.

The selection of quotations for use on the cards followed a strategy (see §6.6.2) to create a set suitable for a general test of the workshop format at a variety of companies. Depending on the local reason for holding a future workshop, a number of variations in procedure may be useful. Some participants mentioned varying the content of the cards or delving more deeply into a single topic as possible changes to the procedure. A subset of cards could be pre-selected by the organiser, perhaps from a larger set containing more of the original interviewees' quotes so that topics not of universal concern can be included. For example, not all companies practice continuous integration or mandatory code review and as such these topics do not appear among the cards that represent common themes. If a company has concerns in such an area they can still benefit from the topic being recognised as shared by others in the industry. One way to begin the process might be to conduct a workshop with the common cards to introduce the process and identify salient topics from those, but also ask participants beforehand to privately identify any broad categories they hope or expect to see, and then ask at the end which of these expectations were not met. Care is needed, though, not to undermine the benefit of the topics being common ones in the industry. Cards might be selected to be locally relevant, and a larger range could be offered from which to select a manageable subset (see §6.6.2 for how the workshop set was selected), but a free-for-all in which wording originates locally risks making the discussion more personal and highly charged.

Delving more deeply would require a consensus on a single topic. For this purpose it could again be useful to begin with the existing procedure and a brief discussion of why each card was chosen, but then agree on a single one of particular concern in order to discuss potential remediation in more depth and draw up a plan of action.

If the workshop is held as a means of facilitating a general review of practices, as in this research, then the current procedure is an appropriate start. But to maximise the learning from it there should be some ongoing engagement. This could involve concluding the meeting by choosing and recording a card to revisit in more detail in a follow-up action planning session. Sometimes participants have "close second" cards where they struggled to choose between options; these could be earmarked for future discussion. In order to act as an "information radiator" (Cockburn, 2007), cards selected in a team's most recent meeting could be physically displayed to offer regular effortless and unobtrusive reminders of the topics that mattered most to people. The discussion process could also be repeated using only cards chosen in previous sessions, with participants choosing a card not for its relevance but how they feel about its progress: whether there have been positive changes or a lack of them; what enabled the change or stood in the way of progress; how the change has helped; what the next step should be.

7.7.4 How would I go about using it?

Participants found facilitation useful (§7.3.1). In theory the workshop could be conducted following the same procedure without the need for a third party, but this has not yet been tried. There is a risk that without one, prompts to get quieter people to contribute might be missed and the discussion get off track. A third party is desirable not only to manage the workshop but to create a tangible outcome for follow-up as the research participants suggested. The proposals in section §7.3.1, for example, require someone to record the decisions about follow-up actions. It appears from a participant's comment about having someone with "the ability to make improvements" that they considered some improvements possible, but not within their own capability to make. This does not mean, though, that the facilitator needs to be a person with authority to make certain changes. Indeed such a person may have too much of a stake in the discussion to be an effective facilitator. The important thing is perhaps not so much who facilitates as that conclusions about viable actions are captured and taken forward with appropriate support for acting upon them.

There is a disheartening degree of passivity in the responses wishing for a workshop outcome that is beyond the powers of an external facilitator to fulfil. While features can be incorporated into the procedure to prompt and facilitate it, there is nothing to prevent participants from making notes and resolving to exercise their own “ability to make improvements”. At one company there seemed to be an air of resignation about changing a characteristic of the work which appeared to be beyond their control, but this was unusual. Having owned the discussion to address local issues in detail, it is not clear why this did not necessarily extend to owning potential solutions. Perhaps it was simply an issue of having a written record as a reminder (as suggested by the comment “take notes for us”), but there is something troubling about the phrase concerning someone “with ability to make improvements”. The appropriate ways to address this will be local according to the make-up of the group and the way that the business is run, but a clear message about expectations, responsibilities and support for participants to enact change would be a useful addition to the workshop. The material concerns programmers’ decisions and actions, so unless constrained by the weight of past decisions it should be within the power of programmers themselves to change these.

The procedure itself (§7.3.1) is straightforward and requires only an uninterrupted meeting room with a large table, simple materials and an hour of people’s time. In the research workshops, participants spent the hour focused entirely on this activity with no interruptions or use of laptops or mobile phones. This is considered essential; listening to others’ explanations is a fundamental part of the workshop objectives and engaging fully only when speaking would undermine the purpose of holding the workshop.

In the research workshops the agreed time was one hour and the procedure was for everyone to choose one card each. When there was time to spare one or two additional cards were allowed, but this was a mistake; the workshop should focus on each person’s top priority. Sometimes a further card enabled people who had found it a hard choice to include the one that came a close second, but a more appropriate step would be to take that forward for future action (such as a another workshop session in a series of regular reviews).

The desired outcome should be considered when setting up a workshop, and the follow up planned accordingly. As suggested in §7.7.3 the materials and discussion procedure might be adapted according to local needs. Similarly any follow-up procedure adopted should reflect these needs; whatever form the follow-up may take, there should be one if the workshop is to be more than a one-off, “so what?” talking shop.

The focus here has been on the workshop format, designed as it is to use the findings of the Exploratory Study by helping developers to discuss aspects of practice which have been shown to exercise their peers. However the materials of the Exploratory Study have also proved useful. One company involved in both the Exploratory and Evaluation Studies reported that staff participation in the former had generated a lot of discussion for some time afterwards. They requested a copy of the interview cards and envisage continuing to use these to reflect on their practice, perhaps annually or to initiate similar discussion with programmers who join the company. Taking time to reflect, as in agile retrospectives (Derby & Larsen, 2006), can be a valuable process for a team to solve its problems and build on its strengths. The materials used in this research can act as a catalyst to focus attention more widely than the confines of the current most urgent problem or the conduct of the most recent sprint, encouraging a holistic review of the whole software development environment created by the team.

7.8 Conclusion

The response to the workshops was resoundingly positive, with 96% of participants happy to recommend the workshop to others and many of them suggesting ways they would like to use it themselves in future. Their suggestions invite further developments of the workshop itself and also raise the possibility of doing more with this kind of socio-technical approach to helping software developers. These are discussed in Chapter 8.

Chapter 8

Summary

8.1 Introduction

This research looked at good practice from a novel perspective, focusing solely on the impact software developers' practices have on their peers' day to day working life. Which practices commonly present a particular help, making them "team-friendly", and which are a hindrance? The reasoning for this perspective was set out in Chapter 1 and placed in the context of other research into the practices of software development in Chapter 2.

The research sprang from personal experience and the desire to create a practical outcome of real use to software developers. This background underpins the pragmatic approach to the research set out in Chapter 3 and the emphasis on designing a technique that would meet the need of the target audience.

To ground the research on a broader foundation than personal experience alone, an Exploratory Study (Chapter 4) was conducted to identify common themes in experienced developers' accounts of what is, and is not, team-friendly practice. The findings (Chapter 5) represent a distillation of over 400 years' experience in the industry from 28 developers working in different software domains. It is unlikely to overlap perfectly with the picture that would be produced by talking to novices; the practices identified are those which continue to affect developers for good or ill long after their novice days are behind them.

Practices were not included unless they had peer impact. For example, practices affecting such factors as the speed or reliability of software were not appraised on these criteria so made an appearance only if reported by software developers as something that affects their Developer eXperience (DX). This is not to say that the scope was narrow. In focusing on the developer's perspective, the research took care to include not just coding but the full breadth of tasks including software architecture, builds, testing, bug reports, version control and direct interactions and experiences with peers. This phase of the work answered Research Question 1, as summarised in §8.2 below.

Identifying peer behaviours that are team-friendly or otherwise was not the end in itself but the prelude to designing a practical application to help teams reflect on their own practices and identify changes they would like to make. This took the form of a workshop design in which team members selected their own choice of topics from themes which interviews with experienced software developers had shown to be important (Chapter 6). This approach not only used an engaging format but was calculated to minimise any sense of personal criticism by presenting only topics validated by people from outside the company.

The teams involved in the testing of this innovative workshop evidently enjoyed the experience and found it useful (Chapter 7); after taking part, 96% said they would recommend it to others. This gives a very positive answer to Research Question 2, as summarised in §8.3.

The success of the research in its stated objectives suggests potential avenues for further investigation, as well as wider dissemination of the approach so that it can be of use to other teams. This is discussed in §8.4.

8.2 RQ1: Perceptions of experienced programmers on peer behaviours

Research Question 1: What is the perception of experienced programmers on the peer behaviours that help or hinder them in their own tasks?

The objectives for Research Question 1 were:

1. To collect opinions from experienced programmers working in a variety of application domains about the helps and hindrances that affect them in the breadth of their everyday work.

2. To identify common themes in their responses.

These objectives were successfully met by the Exploratory Study described in Chapter 4 which collected opinions from experienced programmers. The study took an original approach to exploring concepts of good practice, not measuring code statistics such as complexity or defect count or testing programmers on particular tasks but instead asking them to recount the impact that others' practices have on them.

The picture that emerged (Chapter 5) was not about what is or is not good practice; this was not the question and there was no great disagreement about which category a practice falls into. Instead the findings show those which have most impact, from the perspective of people trying to make progress with their software development tasks, and it is clear that not all practices are created equal.

The findings are a holistic impression of programmers' opinions, spanning much more than the writing of code. Other work has looked individually at many different aspects of a software developer's working life, studying anything from the fine details of code layout to the methods used to coordinate teams. This research covers the entire gamut of processes involved in delivering a software product: not just the raw code but also its supporting infrastructure of tests, version control, builds, bug reports and team communication. These are all features which are largely invisible to end users and compilers and neither of these "audiences" is considered by this research. It considers all these aspects, but it does so from just one perspective: the perceptions of the software developer. This is important not only because it has implications for DX and morale, but also because software development is a labour-intensive task in which inefficient use of a skilled developer's time in dealing with unnecessary obstacles can ill be afforded. Since software tends to evolve rather than be completely replaced, these costs can be repeated again and again throughout its lifetime, growing progressively worse as the software grows.

The data on impactful helps and hindrances was produced by asking software developers to contribute any material they chose as well as providing prompts to help them consider the full breadth of their workplace tasks. The analysis to seek out widespread themes in this material was inevitably influenced by the researcher's perspective (see Chapter 3; peer behaviours are treated as an independent reality) but although the themes represent the researcher's interpretation, nonetheless there were some unforeseen results. The research began with much more focus on code detail, the kind of irritations and joys that are easy to identify and call to mind: grim memories of wading through swathes of copied and pasted code; recalling the unexpected ease of doing an update to crystal clear code with no dead wood; the mysteries of one colleague's apparent passion for using state machines. But the interview procedure proved very effective in enabling people to pause, step back and reflect more deeply than this, something that does not usually happen in the normal course of their working lives. As a participant put it, "It forces you to step back and evaluate processes/how you work."

Topics that were expected be considered hindrances were indeed hindrances, and likewise for the helpful practices, but asking about the relative impact of both types across the whole range of developer tasks opened up a new window onto the importance of how one's peers behave. As explained in Chapter 5, participants carefully took stock and identified the aspects which exercise them most as being ones at a different level to the details of lines of code. Identifiers matter; the rest at that level, governed as it is by strict syntax, they have the skill to cope with, merely slowed down or speeded up a little by the qualities of other people's code.

The things that do matter are all about sharing a bigger picture.

This interpretation is, of course, the researcher's. But it informed how material was selected for use in the Evaluation Study (Chapter 6) and it was clear from their engagement and their feedback that workshop participants identified with the material and readily linked it to troublesome local issues. The themes resonate with the target audience.

8.3 RQ2: Using these perceptions to help others

Research Question 2: Can experienced programmers' accounts of the peer behaviours that most help or hinder them in their own tasks be used to help others in their practice?

The objectives for Research Question 2 were:

1. To present the common "helps and hindrances" themes collected in the first phase to groups of professional software developers for discussion.

2. To evaluate the usefulness of such discussion.

3. To identify other avenues through which the material could be useful.

The workshop format tested in the Evaluation Study (Chapter 6) sought to put the understanding of programmers' working experiences garnered in the Exploratory Study to practical use.

Common themes from interviewees in the Exploratory Study (Chapter 4) were presented to groups of professional software developers in the form of representative quotes, each topic printed on a separate A4 card. While not all of the selected quotes conveyed the theme clearly, most of them served well as a pithy and familiar prompt to participants in the workshops where these cards were discussed.

The value of the discussions was defined solely by the usefulness to the audience they were designed to help with communication about these important topics: the developers themselves (Chapter 7). Evaluation came from their feedback immediately after a workshop, in which all participants responded, and a month later after they had had time to contemplate it while at work (62.5% response rate).

Their answer was clear: the workshop was useful. 96% of all the workshop participants said they would recommend it to others. Of those who gave feedback a month later, 93% said the same. Many also took the time to share their reasons.

As well as being very willing to recommend it, participants would consider using the workshop again themselves. Immediately after the workshop, 88% said they would do so. Among the others the comments showed a feeling that, having done it once, they did not need to repeat it. Of those who followed up a month later, 100% would use the workshop again — often citing specific purposes they had in mind or the adjustments that they would make to fit their purpose. Potential adjustments to the procedure were discussed in Chapter 7.

The objective of the workshop was to be useful to professional software developers. The degree to which this audience found it useful and would recommend it to others shows that this was met, although the research focus solely on the question of usefulness means that little can be said about the process through which its value emerged – information which might help to refine and adapt the workshop procedure. Their compelling enthusiasm in recommending it to others (96%) and proposing future ways in which they would like to use it themselves demonstrates the contribution of an approach that is clearly something new to the participants. Even with healthy teams who, according to both observation and their own comments, were already good at talking to each other, no team was already doing anything like this and all teams found it useful. Perhaps their usual discussions focus on the immediate demands of their iterations: what has been done; what is needed right now; what are we doing next. Rather like standing too close to an elephant, they needed to take a step back to achieve an overview. Taking that step into reflective deliberation does not easily happen in a busy workplace if it is not built into the process.

Perhaps because they were already openly communicative teams the effect of the workshop did not seem to be about seeing behaviour through the eyes of peers and realising its impact. Instead it provided a vehicle through which they were able to recognise their consensus on problems that they all knew. They had clearly not tackled the problems discussed effectively, yet some feasible solutions emerged. In the workshop sessions participants stepped back, recognised the frequency of the day-to-day firefighting and identified longer term fixes. Did the workshop format with its card procedure actually contribute to this, or could they have achieved the same benefits simply by setting aside time and space for a meeting to discuss issues? Perhaps, but it seems likely that those issues would be the pressing ones of the moment. The cards provide a prompt to consider more deeply and identify persistent problems that are a continual irritant but are always worked around, never becoming the current priority. The role of an external facilitator could also be an important factor. To make workshops easier to arrange and therefore increase the likelihood of teams using them, it would be helpful to see if and how they can be run by participants themselves or an independent party within the company.

Given that these excellent results were achieved with “performing” teams who were all willing and able to communicate openly and constructively, it is crucial to consider for whom the workshop can be useful. As they stand the findings in this research do not give any direct indication of whether the workshop can deliver benefits to a team in a less mature state. In part this reflects a missed opportunity that appears with hindsight. The researcher observations reported in Chapter 7 illustrate that the discussions typically progressed rapidly from talking in abstract terms to the researcher, switching instead to concrete discussions within the team. However the field observations did not collect any more detailed data than this about the progression of a workshop as it unfolded. The research question RQ2 to some extent reflects the researcher’s background as a software developer. As discussed in the personal reflections in Section §7.6, it asks ”Can this material help software developers?” without also asking why. The practical upshot of this perspective was that while the workshops were usefully shown to be valued by are their target audience, information about the mechanism through which the value emerged in them is limited.

Such information could be useful in comparing workshops done by performing teams with future workshops for less open and communicative teams, and perhaps adjusting the design accordingly. The feedback has shown that there is an appetite for using the materials not just for communication within teams but between them, from which it might be inferred that there are important discussions to be had, that the communication between teams is not as effective as it might be, and that participants themselves saw something in the workshop that could facilitate communication. It is not clear whether that something was the “safe” nature of the materials, the way in which topics for discussion were selected, the role of the external facilitator or some combination of these, but the desire to use a workshop in such a way points to some potential to help struggling or dysfunctional teams that should be explored.

The team stage questionnaire completed prior to each workshop was used precisely because of the possibility that the approach might not be useful in the context of of a “storming” team. It might instead be hamstrung by the arguing, defensiveness and side-taking (Clark, 2015) characteristic of such teams, or even add fuel to these behaviours. But in some respects these are the teams in most need of help, so further research could be valuable in exploring how the workshop might be applied to help them. This is, however, a difficult proposition. Teams in a turbulent state are unlikely to make the space for such an activity and recruitment would be a hard sell given that, even when more tactfully worded, the invitation is essentially "dysfunctional teams wanted for research". It is not surprising that the teams in the study were well-functioning ones; it is hard to imagine troubled teams signing up to talk to each other in front of a researcher. But it would be interesting to to see whether the respectful, inclusive and unusual format or the use of opinions of experienced professionals from outside the company could aid communication in these environments. Since teams who took part in the workshops expressed an interest in using the same procedure to facilitate discussion between teams, that could be a modest first step to begin exploring how the workshop performs when the existing relationship between the participants is more challenging. This would allow the procedure to be tested with relationships that exhibit some communication problems but are not, judging by the tone of the participants who mentioned the idea in person, difficult or strained.

There are other future adaptations of the workshop technique to explore in light of feedback from those who took part. Useful suggestions were made about the selection of materials, the depth of discussion and a process of follow-up, as discussed in Chapter 7. But there are also new applications to try. Participants saw potential use for the workshop not only for future discussions of a similar nature within or between teams, but also for recruitment, onboarding and appraisals. Appraisals were also mentioned in the Exploratory Study (Chapter 4) by an interviewee who said:

> "Every year we have our appraisals and we talk about other people and all this stuff is too petty to raise then because being mean about somebody just because ...you find dealing with their code a bit of a pain seems a bit unfair so you don't mention it and hope they return the favour, prisoner's dilemma style."

In addition to reviewing one's own performance, then, the materials which open these issues for discussion might be useful for giving feedback for others without it seeming personal.

8.4 Future work

8.4.1 Further development of the workshop framework

Despite the confounding influence of ongoing changes in work environment and practices it would be interesting to see how participants might reflect on the workshop further (perhaps one year on) into the future. No formal follow-up was done on the Exploratory Study, designed as it was to elicit individual opinions, but evidently the process had provoked discussion. Some of the suggestions about adaptations to the format also apply here. Having established the usefulness of the basic format, there is scope to investigate what is needed for an effective ongoing process of fruitful discussions and actions.

The companies who so generously participated in this research, while diverse, represent a tiny sample of the industry. It is clear from their feedback that they are not already having this kind of discussion, so it would be fascinating to do participant observations in a wide range of businesses to observe the planned discussions (retrospectives, for example, and other scheduled team meetings) that take do take place, to what effect, and investigate whether it would be helpful to design a different approach to these. It could be beneficial to accommodate effective ongoing reflection on team practices as an integral part of the process, rather than a separate meeting as was done with the workshops.

8.4.2 Applications of the workshop framework to other domains

Adaptations to the content could be taken further, using the same basic format to focus on other concerns. Bringing together new card topics and the ideas for inter-team discussion, one aspect which could usefully be addressed is security in software design. Security has become increasingly an issue for software but has not until recently been emphasised in software education. “Knowledge and understanding of information security issues in relation to the design, development and use of information systems” was a core requirement for BCS accredited honours programmes in 2012 (BCS, 2012, p.9) but its prominence has increased. Cybersecurity, not mentioned until 2015 (BCS, 2015), now merits its own section of the guidelines (BCS, 2018, pp.17–18). The Internet of Things (IoT) in particular could be an area to benefit from sharing understandings between teams in different disciplines. The enthusiasm for the affordances of IoT has run ahead of an understanding of the dark side (De Cremer, Nguyen, & Simkin, 2017) so constructive communication between those responsible for the marketing vision, developers (often with limited security knowledge) tasked with implementing it, and those who have security expertise could be helpful in avoiding incidents such as customer credentials and voice messages between parents and children being leaked by an insecure teddy bear (Franceschi-Bicchierai, 2017).

Since the approach of this research has been so successful in facilitating software developers' discussion of their practices and the impact on their work, it invites a whole range of further research into socio-technical aspects of software development. An obvious topic is tools; software developers routinely use a whole suite of them to do the job so the scope of tools to make a difference, for good or ill, is considerable. As an interviewee in the Exploratory Study put it:

> "When you have to fight the tools to just do basic things, its like, you wouldn't make a Formula 1 driver wear a cast on his leg. The tools need to get out of the way and let you focus on the job. If you struggle with a tool that doesn't work right then it can be very awkward."

The "struggle" is pertinent; like the well-known but unresolved problems the workshop participants discussed and addressed, tool frustrations are another irritant that is commonly worked around rather than resolved. Sometimes particular tools are mandated for reasons beyond the developers' control, but just as with software problems there could be value in teams taking the time out to reflect on tool use, create transparency about practices adopted to work around problems and perhaps come up with an agreed solution.

Looking even more broadly, a similar approach based on making local selections from the experiences of others in the industry might be applied in any industry where communication about the team's practices needs to be facilitated. The researcher's own experience is of the software industry, where the skills of communication are orthogonal to those for doing the technical apects of the job. The particular characteristics of software development make the process particularly apposite here, but the non-personal way of allowing topics to be introduced, "legitimised" by peers who are not colleagues, may have merit for teams of other kinds.

8.4.3 Target audience

While expertise was a requirement for the Exploratory Study, the aim of the Evaluation Study to share that expertise with others invites consideration of an audience other than professional programmers. The intended application of the research was to this community, but it also has potential application to undergraduate students.

Undergraduate degree courses typically do not expose students to the extent of the dependencies that professional programmers have on others' approach (see §2.7.1). There may therefore be scope for using material from this research to introduce undergraduates to the idea of team-friendly practices and begin to educate them about the impact their actions as a programmer can have on their future colleagues. But in the early stages of their careers some of the difficulties they encounter when working with others' software will occur simply because they are novices — and unless they have had an industry placement they may not have experienced any such software at all. The concepts of team-friendly practices may be rather too abstract without the concrete foundation of practical experience to trigger a sense of recognition, and therefore of limited benefit to them at an undergraduate stage, but the material may be of value to final year undergraduates after an industry placement and to graduate developers. Even when some of their difficulties are due to lack of knowledge or experience, it is useful to remind them that in solving those there is more to consider in software development than getting the computer to do something.

8.5 Conclusion

In the "any additional comments" section of the pre-workshop questionnaire a team leader spontaneously captured the essence of the shared understanding among software developers that this research sought to achieve:

> "The team is somewhat overworked at present and in need of a bit more of a structured guidance with a good understanding of priorities. When very busy it is easy to fall into specialists performing jobs they're most suited to and communication and sharing of knowledge to diminish. This in turn causes friction, especially when you are 'fixing' problems caused by others. [I need] to maintain the inter-team communication and 'team spirit' to avoid any virtual walls being built."

This illustrates the real world relevance of a technique to facilitate understanding among professional software developers. The results from the novel workshop format developed and tested in this research show that it is effective in aiding communication, sharing knowledge and (with a little refinement to ensure that decisions are carried forward) setting priorities to reduce the friction of the job.

References

Aas, P. (2019, July). I had a client where I had to bring a guy to every meeting, with detailed notes on what he should say. The client ignored everything that came out of my mouth, so I got a myself a dude as a frontman [microblog]. Retrieved 2019-07-28, from https://twitter.com/pati_gallardo/status/1154456543019933697

Abbate, J. (2012). Recoding Gender : Women's Changing Participation in Computing. Cambridge, Mass: MIT Press.

Abbes, M., Khomh, F., Guéhéneuc, Y.-G., & Antoniol, G. (2011). An Empirical Study of the Impact of Two Antipatterns, Blob and Spaghetti Code, on Program Comprehension. In 2011 15th European Conference on Software Maintenance and Reengineering (CSMR) (pp. 181–190). doi: 10.1109/CSMR.2011.24

Abran, A. (2004). Software Maintenance. In A. Abran & J. W. Moore (Eds.), Guide to the software engineering body of knowledge. Los Alamitos, Calif.: IEEE Computer Society Press.

ACM and IEEE Computer Society. (2013). Computer Science Curricula 2013 - Curriculum Guidelines for Undergraduate Degree Programs in Computer Science. Retrieved 2015-02-22, from http://www.acm.org/education/CS2013-final-report.pdf

ACM and IEEE Computer Society. (2015). Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering (Tech. Rep.). Retrieved 2017-08-14, from http://www.acm.org/education/CS2013-final-report.pdf

Agile Alliance. (2001a). The Agile Manifesto. Retrieved 2013-10-09, from http://www.agilealliance.org/the-alliance/the-agile-manifesto/

Agile Alliance. (2001b). Principles behind the Agile Manifesto. Retrieved 2017-08-15, from http://agilemanifesto.org/principles.html

Ahmed, F., Capretz, L. F., Bouktif, S., & Campbell, P. (2013, May). Soft Skills and Software Development: A Reflection from the Software Industry.

International Journal of Information Processing and Management(IJIPM), 4(3). doi: 10.4156/ijipm.vol4.issue3.17

Anderson, N. R., & West, M. A. (1998). Measuring Climate for Work Group Innovation: Development and Validation of the Team Climate Inventory. Journal of Organizational Behavior(3), 235–258.

Arthur, C. (2012, June). How NatWest's IT meltdown developed. Retrieved 2013-10-08, from http://www.theguardian.com/technology/2012/jun/25/how-natwest-it-meltdown

Avison, D. E., & Fitzgerald, G. (2006). Information systems development : methodologies, techniques & tools. London : McGraw-Hill Education, c2006.

Baddeley, A. D., Eysenck, M. W., & Anderson, M. C. (2015). Memory. Hove, East Sussex; New York: Psychology Press.

Baker, J. (2016, May). Developer Experience (DX) — Devs Are People Too. Retrieved 2017-07-26, from https://dzone.com/articles/developer-experience-dx-devs-are-people-too

BBC. (2002, January). Air traffic centre opens six years late. Retrieved 2013-10-08, from http://news.bbc.co.uk/1/hi/england/1781915.stm

BBC. (2018, July). IT fiasco pushes TSB into a loss. BBC News. Retrieved 2018-08-03, from https://www.bbc.com/news/business-44978503

BCS. (2012). Guidelines on course accreditation: Information for universities and colleges. Retrieved 2015-02-22, from http://www.bcs.org/upload/pdf/hea-guidelinesfull-2012_1.pdf

BCS. (2015). Guidelines on course accreditation: Information for universities and colleges. Retrieved 2015-02-22, from http://www.bcs.org/upload/pdf/hea-guidelinesfull-2012_1.pdf

BCS. (2018). Guidelines on course accreditation: Information for universities and colleges. Retrieved 2018-07-28, from https://www.bcs.org/upload/pdf/2018-guidelines.pdf

Berry, D. C. (1987). The problem of implicit knowledge. Expert Systems, 4(3), 144–151. doi: 10.1111/j.1468-0394.1987.tb00138.x

Boehm, B. (2006). A view of 20th and 21st century software engineering. In Proceedings of the 28th international conference on Software engineering (pp. 12–29). Shanghai, China: ACM.

Boehm, B., Clark, B. K., Horowitz, E., Brown, A. W., Reifer, D. J., Chulani, S., … Steece, B. (2000). Software Cost Estimation with Cocomo II. Upper Saddle River, NJ, USA: Prentice Hall PTR.

Bossavit, L. (2012). The Leprechauns of Software Engineering. Vancouver, BC: Leanpub.

Bower, G. H., & Forgas, J. P. (2000). Affect, memory, and social cognition. In

E. Eich, J. F. Kihlstrom, G. H. Bower, J. P. Forgas, & P. M. Niedenthal (Eds.), Cognition and emotion. Oxford; New York: Oxford University Press.

Brechner, E. (2003). Things They Would Not Teach Me of in College: What Microsoft Developers Learn Later. In Companion of the 18th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (pp. 134–136). New York: ACM. doi: 10.1145/949344.949387

Brooks, F. (1987). No Silver Bullet: Essence and Accidents of Software Engineering. Computer, 20(4), 10–19.

Brooks, R. (1983). Towards a theory of the comprehension of computer programs. International Journal of Man-Machine Studies, 18(6), 543–554.

Bryant, S. (2004, September). Double Trouble: Mixing Qualitative and Quantitative Methods in the Study of eXtreme Programmers. In 2004 IEEE Symposium on Visual Languages - Human Centric Computing (pp. 55–61). doi: 10.1109/VLHCC.2004.20

Buse, R., & Weimer, W. (2010). Learning a metric for code readability. IEEE Transactions on Software Engineering, 36(4), 546–558. doi: 10.1109/TSE.2009.70

Case, P., & Piñeiro, E. (2006). Aesthetics, performativity and resistance in the narratives of a computer programming community. Human Relations, 59(6), 753–782. doi: 10.1177/0018726706066853

Ceruzzi, P. E. (2003). A History of Modern Computing. Cambridge, Mass: MIT Press.

Charette, R. (2005). Why software fails. IEEE Spectrum, 42(9), 42–49. doi: 10.1109/MSPEC.2005.1502528

Chase, W. G., & Simon, H. A. (1973). The mind's eye in chess. In W. G. Chase (Ed.), Visual Information Processing (pp. 215–281). Oxford, England: Academic Press.

Chidamber, S. R., & Kemerer, C. F. (1994). Metrics suite for object oriented design. IEEE Transactions on Software Engineering, 20(6), 476–493. doi: 10.1109/32.295895

Chope, R. C. (2015). Card sorts, sentence completions, and other qualitative assessments. In P. J. Hartung, M. L. Savickas, & W. B. Walsh (Eds.), APA handbook of career intervention, Volume 2: Applications (pp. 71–84). Washington, DC, US: American Psychological Association.

Cialdini, R. B. (1993). Influence : the psychology of persuasion. New York: Collins.

Citizens Advice. (2015). How to run focus groups. Retrieved 2018-09-09, from https://www.citizensadvice.org.uk/Global/CitizensAdvice/Equalities/Howtorunfocusgroupsguide.pdf

Clark, D. (2015). Leadership. Retrieved 2017-08-24, from http://www.nwlink.com/~donclark/leader/leadtem2.html

Clark, D. (2016). Teamwork survey. Retrieved 2017-08-09, from http://www.nwlink.com/~donclark/leader/teamsuv.html

Cockburn, A. (2004). Crystal Clear: A Human-Powered Methodology for Small Teams. Boston: Addison-Wesley Professional.

Cockburn, A. (2007). Agile software development : the cooperative game. Upper Saddle River, N.J.: Addison-Wesley.

Colucci, E. (2007, December). Focus groups can be fun: the use of activity-oriented questions in focus group discussions. Qualitative Health Research, 17(10), 1422–1433.

Conboy, K., Coyle, S., Wang, X., & Pikkarainen, M. (2011). People over Process: Key Challenges in Agile Development. IEEE Software, Software, IEEE, IEEE Softw.(4), 48. doi: 10.1109/MS.2010.132

Coolican, H. (2014). Research Methods and Statistics in Psychology (6th ed.). East Sussex: Psychology Press.

Coplien, J. O., & Harrison, N. B. (2004). Organizational Patterns of Agile Software Development. Prentice-Hall, Inc.

Creative Commons. (n.d.). Attribution-NonCommercial 2.0 Generic — CC BY-NC 2.0. Retrieved 2017-08-09, from https://creativecommons.org/licenses/by-nc/2.0/deed.en

Creswell, J. W. (2013). Research Design: Qualitative, Quantitative, and Mixed Methods Approaches (3rd ed.). Thousand Oaks, CA, US: SAGE Publications.

Cruz, S., da Silva, F. Q. B., & Capretz, L. F. (2015, May). Forty years of research on personality in software engineering: A mapping study. Computers in Human Behavior, 46, 94–113. Retrieved 2015-03-23, from http://www.sciencedirect.com/science/article/pii/S0747563214007237 doi: 10.1016/j.chb.2014.12.008

Curtis, B., Krasner, H., & Iscoe, N. (1988). A field study of the software design process for large systems. Commun. ACM, 31(11), 1268–1287.

Curtis, B., Sheppard, S., Milliman, P., Borst, M. A., & Love, T. (1979). Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics. IEEE Transactions on Software Engineering, SE-5(2), 96–104. doi: 10.1109/TSE.1979.234165

Curtis, B., & Walz, D. (1991). The Psychology of Programming in the Large: Team and Organizational Behaviour. In Psychology of Programming (pp. 253–270).

De Cremer, D., Nguyen, B., & Simkin, L. (2017). The integrity challenge of the Internet-of-Things (IoT): on understanding its dark side. Journal of Marketing Management, 33(1/2), 145–158. doi: 10.1080/0267257X.2016.1247517

DeMarco, T., & Lister, T. R. (1999). Peopleware : productive projects and teams.

New York: Dorset House.

Department for Education. (2013). National curriculum in England: computing programmes of study. Retrieved 2018-07-17, from https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study/national-curriculum-in-england-computing-programmes-of-study

Derby, E., & Larsen, D. (2006). Agile retrospectives : making good teams great. Raleigh, NC: Pragmatic Bookshelf.

Detienne, F. (2001). Software Design - Cognitive Aspects. London ; New York: Springer.

Di Penta, M., & Tamburri, D. A. (2017). Combining Quantitative and Qualitative Studies in Empirical Software Engineering Research. In Proceedings of the 39th International Conference on Software Engineering Companion (pp. 499–500). Piscataway, NJ, USA: IEEE Press. doi: 10.1109/ICSE-C.2017.163

Easton, S., & Van Laar, D. (2018). User Manual for the Work-Related Quality of Life (WRQoL) Scale: A Measure of Quality of Working Life. University of Portsmouth. Retrieved 2018-10-04, from https://researchportal.port.ac.uk/portal/en/publications/user-manual-for-the-workrelated-quality-of-life-wrqol-scale(38db9f62-d8c3-4f80-8559-5b74de164383).html doi: 10.17029/EASTON2018

Edwards, J. A., Van Laar, D., Easton, S., & Kinman, G. (2009). The Work-Related Quality of Life Scale for Higher Education Employees. Quality in Higher Education, 15(3), 207–219.

Elvins, J. P. (1985, December). Communication in Quality Circles: Members' Perceptions of Their Participation and Its Effects on Related Organizational Communication Variables. Group & Organization Management, 10(4), 479.

Feathers, M. C. (2005). Working effectively with legacy code. Upper Saddle River, NJ: Prentice Hall Professional Technical Reference.

Ford, D., & Parnin, C. (2015, May). Exploring Causes of Frustration for Software Developers. In 2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering (pp. 115–116). doi: 10.1109/CHASE.2015.19

Fowler, M. (1999). Refactoring : improving the design of existing code. Reading, Mass.; Harlow: Addison-Wesley.

Franceschi-Bicchierai, L. (2017). Internet of Things Teddy Bear Leaked 2 Million Parent and Kids Message Recordings. Retrieved 2018-07-28, from https://motherboard.vice.com/en_us/article/pgwean/internet-of-things-teddy-bear-leaked-2-million-parent-and-kids-message-recordings

Ghobadi, S., & Mathiassen, L. (2016, March). Perceived barriers to effective knowledge sharing in agile software teams. Information Systems Journal, 26(2), 95–125. doi: 10.1111/isj.12053

Glass, R. L. (2003). Facts and fallacies of software engineering. Boston, MA : Addison-Wesley.

Glass, R. L. (2005). IT failure rates - 70% or 10-15%? IEEE Software, 22(3), 112+110–111. doi: 10.1109/MS.2005.66

Goodliffe, P. (2006). Code craft : the practice of writing excellent code. San Francisco, Calif. : No Starch Press.

Goodliffe, P. (2012, June). Suffering the tyranny of the "TODO" comment. People: "do" your "todo"s. [microblog]. Retrieved 2015-01-17, from https://twitter.com/petegoodliffe/status/217622701605535744

Gray, D. E. (2013). Doing Research in the Real World (3rd ed.). London: SAGE Publications Ltd.

Graziotin, D., Fagerholm, F., Wang, X., & Abrahamsson, P. (2017). On the Unhappiness of Software Developers. arXiv:1703.04993 [cs], 324–333. Retrieved from http://arxiv.org/abs/1703.04993 (arXiv: 1703.04993) doi: 10.1145/3084226.3084242

Green, T. R. G., & Petre, M. (1996). Usability analysis of visual programming environments: A 'cognitive dimensions' framework. Journal of Visual Languages and Computing, 7(2), 131–174.

Grubb, P. (2003). Software maintenance: concepts and practice. River Edge, N.J.: World Scientific.

Grudin, J. (2012). A Moving Target: The Evolution of Human-Computer Interaction. In J. A. Jacko (Ed.), Human Computer Interaction Handbook: Fundamentals, Evolving Technologies, and Emerging Applications (3rd ed.). Boca Raton: CRC Press.

Guzdial, M., & Guo, P. (2014). The Difficulty of Teaching Programming Languages, and the Benefits of Hands-on Learning. Communications of the ACM, 57(7), 10–38.

Haiduc, S., Aponte, J., & Marcus, A. (2010). Supporting program comprehension with source code summarization. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2 (pp. 223–226). Cape Town, South Africa: ACM.

Hall, T., Zhang, M., Bowes, D., & Sun, Y. (2014). Some Code Smells Have a Significant but Small Effect on Faults. ACM Transactions on Software Engineering & Methodology, 23(4), 33.

Halstead, M. H. (1977). Elements of software science. New York: Elsevier.

Hansen, M., Goldstone, R. L., & Lumsdaine, A. (2013, April). What Makes Code Hard to Understand? arXiv:1304.5257 [cs]. Retrieved 2014-09-04, from http://arxiv.org/abs/1304.5257

Henney, K. (Ed.). (2010). 97 things every programmer should know : collective wisdom from the experts. Sebastopol, Calif. : O'Reilly Media.

Henry, S., & Kafura, D. (1981). Software Structure Metrics Based on Information Flow. IEEE Transactions on Software Engineering, 7(5), 510.

Highsmith, J., & Cockburn, A. (2001). Agile software development: the business of innovation. Computer(9), 120. doi: 10.1109/2.947100

Hogg, M. A., & Vaughan, G. M. (2011). Social psychology. Harlow: Prentice Hall.

Hunt, A., & Thomas, D. (2000). The pragmatic programmer : from journeyman to master. Reading, Mass: Addison-Wesley.

Imperial College. (2017). Software Engineering for Industry. Retrieved 2017-08-15, from https://www.doc.ic.ac.uk/~rbc/475/

Jalali, S., & Wohlin, C. (2012). Global software engineering and agile practices: a systematic review. Journal of Software Maintenance and Evolution(6). doi: 10.1002/smr.561

Johnson, M., & Senges, M. (2010). Learning to be a programmer in a complex organization. Journal of Workplace Learning, 22(3), 180–194. doi: 10.1108/13665621011028620

Kaner, C., & Bond, W. P. (2004). Software Engineering Metrics: What Do They Measure and How Do We Know? In METRICS 2004. IEEE CS. Press.

Kaplan, D. (2014). Coder vs Programmer vs Software Engineer vs Architect vs… – Sleep Easy Software. Retrieved 2018-08-18, from http://www.sleepeasysoftware.com/coder-vs-programmer-vs-software-engineer-vs-architect-vs/

Katzmarski, B., & Koschke, R. (2012). Program complexity metrics and programmer opinions. 20th IEEE International Conference on Program Comprehension (ICPC), 17.

King, N. (2012). Doing template analysis. In G. Symon & C. Cassell (Eds.), Qualitative organizational research : core methods and current challenges. London: SAGE.

Knuth, D. E. (1974). Computer Programming as an Art. Communications of the ACM, 17(12), 667–673. doi: 10.1145/361604.361612

Knuth, D. E. (1984). Literate Programming. The Computer Journal, 27(2), 97–111. doi: 10.1093/comjnl/27.2.97

Knuth, D. E. (2009). The art of computer programming. Boston, Mass. ; London : Addison-Wesley.

Ko, A. J., Myers, B. A., Coblenz, M. J., & Aung, H. H. (2006). An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. Software Engineering, IEEE Transactions on, 32(12), 971–987.

Kuhrmann, M., & Münch, J. (2016). When Teams Go Crazy: An Environment to Experience Group Dynamics in Software Project Management Courses. 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), Software Engineering Companion (ICSE-C), IEEE/ACM International Conference on, ICSE-C, 412.

LaToza, T. D., Garlan, D., Herbsleb, J. D., & Myers, B. A. (2007). Program comprehension as fact finding. In Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering (pp. 361–370). Dubrovnik, Croatia: ACM.

LaToza, T. D., & Myers, B. A. (2010). Hard-to-answer questions about code. In Evaluation and Usability of Programming Languages and Tools (pp. 1–6). Reno, Nevada: ACM.

Lee, E. (2008). Forming to Performing: Transitioning Large-Scale Project Into Agile. Agile 2008 Conference, Agile, 2008. AGILE '08. Conference, 106. doi: 10.1109/Agile.2008.75

Lenberg, P., Feldt, R., & Wallgren, L. G. (2015, September). Behavioral software engineering: A definition and systematic literature review. Journal of Systems and Software, 107, 15–37. doi: 10.1016/j.jss.2015.04.084

Liblit, B., Begel, A., & Sweetser, E. (2006, September). Cognitive Perspectives on the Role of Naming in Computer Programs. University of Sussex. Retrieved from http://www.ppig.org/sites/default/files/2006-PPIG-18th-liblit.pdf

Linberg, K. R. (1999). Software developer perceptions about software project failure: a case study. Journal of Systems and Software, 49(2), 177–192. doi: 10.1016/S0164-1212(99)00094-1

Lind, R., & Vairavan, K. (1989). An experimental investigation of software metrics and their relationship to software development effort. IEEE Transactions on Software Engineering, 15(5), 649–653. doi: 10.1109/32.24715

Lingel, J., & Regan, T. (2014). "It's in Your Spinal Cord, It's in Your Fingertips": Practices of Tools and Craft in Building Software. In Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing (pp. 295–304). New York: ACM. doi: 10.1145/2531602.2531614

Macnaghten, P., & Myers, G. (2004). Focus groups. In C. Seale, G. Gobo, J. F. Gubrium, & D. Silverman (Eds.), Qualitative Research Practice (pp. 65–79). London: SAGE.

Maguire, S. (1993). Writing solid code : Microsoft's techniques for developing bug-free C programs. Redmond, Wash: Microsoft Press.

McCabe, T. (1976, December). A Complexity Measure. IEEE Transactions on

Software Engineering, SE-2(4), 308–320. doi: 10.1109/TSE.1976.233837

McConnell, S. C. (2004). Code complete. Redmond, Wash. : Microsoft.

Menzies, T., Greenwald, J., & Frank, A. (2007, January). Data Mining Static Code Attributes to Learn Defect Predictors. IEEE Transactions on Software Engineering, 33(1), 2–13.

Miller, J., & Glassner, B. (2004). The 'inside' and the 'outside': finding realities in research interviews. In D. Silverman (Ed.), Qualitative Research: Theory, Method and Practice. London: SAGE.

Mind Tools. (n.d.). Forming, Storming, Norming, and Performing: Understanding the Stages of Team Formation. Retrieved 2018-07-19, from http://www.mindtools.com/pages/article/newLDR_86.htm

Mintel. (2018). Cloud Computing - UK - August 2018 (Tech. Rep.). Retrieved 2019-07-22, from http://academic.mintel.com/display/863097/

Morris, P. D. (2017, June). John D. Rockefeller and Alexander Hamilton The Founding Fathers of Agile. CrossTalk: The Journal of Defense Software Engineering, 30(3), 15–22.

Moser, R., Pedrycz, W., & Succi, G. (2008). A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In ACM/IEEE 30th International Conference on Software Engineering, 2008. ICSE '08 (pp. 181–190). doi: 10.1145/1368088.1368114

National Geographic Society. (2011, January). Y2k bug. Retrieved 2018-07-18, from http://www.nationalgeographic.org/encyclopedia/Y2K-bug/

O'Brien, M. P., & Buckley, J. (2001). Inference-based and expectation-based processing in program comprehension. In Proceedings of IWPC 2001: 9th International Workshop on Program Comprehension (pp. 71–78).

Ollis, G. (2014). What programmers want. In ACCU 2014: The Conference for Developers. Bristol. Retrieved from http://accu.org/index.php/conferences/accu_conference_2014/accu2014_sessions#what_programmer_s_want

Oram, A., & Wilson, G. (Eds.). (2007). Beautiful code. North Sebastapol, Calif. ; Farnham : O'Reilly.

Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., ... Paterson, J. (2007). A Survey of Literature on the Teaching of Introductory Programming. In Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education (pp. 204–223). New York: ACM. doi: 10.1145/1345443.1345441

Petre, M. (1991). What experts want from programming languages. Ergonomics, 34(8), 1113–1127. doi: 10.1080/00140139108964850

Petre, M., & Blackwell, A. F. (1997). A Glimpse of Expert Programmers' Mental Imagery. In Papers presented at the seventh workshop on Empirical studies of programmers (pp. 109–123). ACM.

Petre, M., Hoek, A. V. D., & Quach, Y. (2016). Software Design Decoded: 66 Ways Experts Think. Cambridge, MA: MIT Press.

Pikkarainen, M. (2008). The impact of agile practices on communication in software development. EMPIRICAL SOFTWARE ENGINEERING, 13(3), 303–337.

Radermacher, A., Walia, G., & Knudson, D. (2014). Investigating the Skill Gap Between Graduating Students and Industry Expectations. In Companion Proceedings of the 36th International Conference on Software Engineering (pp. 291–300). New York: ACM. doi: 10.1145/2591062.2591159

Rajlich, V., & Wilde, N. (2002). The role of concepts in program comprehension. In Proceedings 10th International Workshop on Program Comprehension (pp. 271–278). doi: 10.1109/WPC.2002.1021348

Rangel, F. (2015, December). APIs for humans: The rise of developer experience (DX). Retrieved 2017-07-26, from https://thenextweb.com/dd/2015/12/20/apis-for-humans-the-rise-of-developer-experience-dx/

Rapley, T. (2004, January). Interviews. In C. Seale, G. Gobo, J. F. Gubrium, & D. Silverman (Eds.), Qualitative Research Practice. SAGE.

Robillard, M. P., Coelho, W., & Murphy, G. C. (2004). How effective developers investigate source code: an exploratory study. IEEE Transactions on Software Engineering,, 30(12), 889–903.

Roehm, T., Tiarks, R., Koschke, R., & Maalej, W. (2012). How do professional developers comprehend software? In Proceedings of the 2012 International Conference on Software Engineering (pp. 255–265). Zurich, Switzerland: IEEE Press.

Rowley, D., & Lange, M. (2007, January). Forming to Performing: The Evolution of an Agile Team. AGILE 2007 (AGILE 2007), 408.

Royce, W. (1970). Managing the Development of Large Software Systems: Concepts and Techniques. In Technical Papers of Western Electronic Show and Convention (WesCon). Los Angeles, USA.

Sanders, K., Fincher, S., Bouvier, D., Lewandowski, G., Morrison, B., Murphy, L., … Scott, T. (2005). A multi-institutional, multinational study of programming concepts using card sort data. Expert Systems, 22(3), 121–128.

Scott, M. L. (2009). Programming Language Pragmatics. San Francisco, Calif.: Morgan Kaufmann.

Sharp, H., Robinson, H., & Petre, M. (2009). The role of physical artefacts in agile software development: Two complementary perspectives. Interacting with Computers, 21(1–2), 108–116.

Shepperd, M., Bowes, D., & Hall, T. (2014). Researcher bias: The use of machine learning in software defect prediction. IEEE Transactions on Software

Engineering, 40(6), 603–616. Retrieved 2017-08-30, from http://bura.brunel.ac.uk/handle/2438/8784 doi: 10.1109/TSE.2014.2322358

Sillito, J., Murphy, G. C., & De Volder, K. (2006). Questions programmers ask during software evolution tasks. In Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering (pp. 23–34). Portland, Oregon, USA: ACM.

Skorkin, A. (2010). The Difference Between A Developer, A Programmer And A Computer Scientist. Retrieved 2018-08-18, from https://www.skorks.com/2010/03/the-difference-between-a-developer-a-programmer-and-a-computer-scientist/

Soloway, E., Adelson, B., & Ehrlich, K. (1988). Knowledge and processes in the comprehension of computer programs. In M. T. H. Chi, R. Glaser, & M. J. Farr (Eds.), The nature of expertise (pp. 129–152).

Soloway, E., & Ehrlich, K. (1984). Empirical Studies of Programming Knowledge. IEEE Transactions on Software Engineering, SE-10(5), 595–609. doi: 10.1109/TSE.1984.5010283

Sommerlad, P. (2010). Only the code tells the truth. In K. Henney (Ed.), 97 things every programmer should know : collective wisdom from the experts. Sebastopol, Calif. : O'Reilly Media.

Spraul, V. A. (2012). Think Like a Programmer : An Introduction to Creative Problem Solving. San Francisco: No Starch Press.

Stein, J. (n.d.). HR at MIT | Learning & Development | Using the Stages of Team Development. Retrieved 2017-07-23, from http://hrweb.mit.edu/learning-development/learning-topics/teams/articles/stages-development

Stellman, A., & Greene, J. (Eds.). (2009). Beautiful teams. Sebastopol, Calif.: O'Reilly.

Stewart, D. W., & Shamdasani, P. N. (2015). Focus groups : theory and practice / David W. Stewart, Prem N. Shamdasani. Thousand Oaks, California : SAGE, [2015].

Storey, M. (2006, September). Theories, tools and research methods in program comprehension: past, present and future. Software Quality Journal, 14(3), 187–208. doi: 10.1007/s11219-006-9216-4

Symon, G., & Cassell, C. (2004). Essential guide to qualitative methods in organizational research. London: SAGE.

Tashakkori, A., & Teddlie, C. (1998). Mixed methodology: Combining qualitative and quantitative approaches. Thousand Oaks, CA, US: Sage Publications, Inc.

Teague, D., Corney, M., Fidge, C., Roggenkamp, M., Ahadi, A., & Lister, R. (2012). Using neo-piagetian theory, formative in-class tests and think alouds to better understand student thinking: A preliminary report on computer programming. In Proceedings of 2012 Australasian Association for Engineering Education (AAEE) Annual Conference. (p. 772). Engineers Australia.

Tekir, S. (2012). Reading CS classics. Communications of the ACM, 55(4), 32–34.

Tripathy, P., Naik, S., Tripathy, P., & Naik, K. (2014). Software Evolution and Maintenance. New York: John Wiley & Sons, Incorporated.

TSB. (2018, April). We''re putting things right. Retrieved 2018-08-03, from https://www.tsb.co.uk/news-releases/2018-q1-results/

Tuckman, B. W. (1965). Developmental sequence in small groups. Psychological Bulletin, 63(6), 384–399. doi: 10.1037/h0022100

Turner, R., & Boehm, B. (2003, December). People Factors in Software Management: Lessons From Comparing Agile and Plan-Driven Methods. CrossTalk: The Journal of Defense Software Engineering, 16(12), 4–8.

Usability.gov. (2013, October). Card Sorting. Retrieved 2018-08-03, from https://www.usability.gov/how-to-and-tools/methods/card-sorting.html

Vallon, R., da Silva Estácio, B. J., Prikladnicki, R., & Grechenig, T. (2018, April). Systematic literature review on agile practices in global software development. Information & Software Technology, 96, 161–180. doi: 10.1016/j.infsof.2017.12.004

Van Laar, D., Edwards, J. A., & Easton, S. (2007). The Work-Related Quality of Life scale for healthcare workers. Journal Of Advanced Nursing, 60(3), 325–333.

Verhoef, C. (2000). How to implement the future? In Proceedings of the 26th Euromicro Conference, 2000 (Vol. 1, pp. 32–47). IEEE.

Vlatko, N. (2015). Coder, developer, programmer, software engineer: What's the difference? Retrieved 2018-08-18, from https://jaxenter.com/coder-developer-programmer-software-engineer-whats-the-difference-117922.html

Walz, D. B., Elam, J. J., & Curtis, B. (1993). Inside a Software Design Team: Knowledge Acquisition, Sharing, and Integration. Communications of the ACM, 36(10), 63–77. doi: 10.1145/163430.163447

Watkin, J. (2017). 17 Experts Weigh in on the Term "Soft Skills". Retrieved 2018-07-18, from https://www.icmi.com/Resources/Learning-and-Development/2017/06/17-Experts-Weigh-in-on-the-Term-Soft-Skills

Watts, S., & Stenner, P. (2012). Doing Q methodological research: theory, method and interpretation. London: SAGE.

Weinberg, G. M. (1998). The Psychology of Computer Programming. New York: Dorset House Pub.

Wilde, N., & Casey, C. (1996, November). Early field experience with the Software Reconnaissance technique for program comprehension. In International

Conference on Software Maintenance 1996, Proceedings (pp. 312–318). doi: 10.1109/ICSM.1996.565034

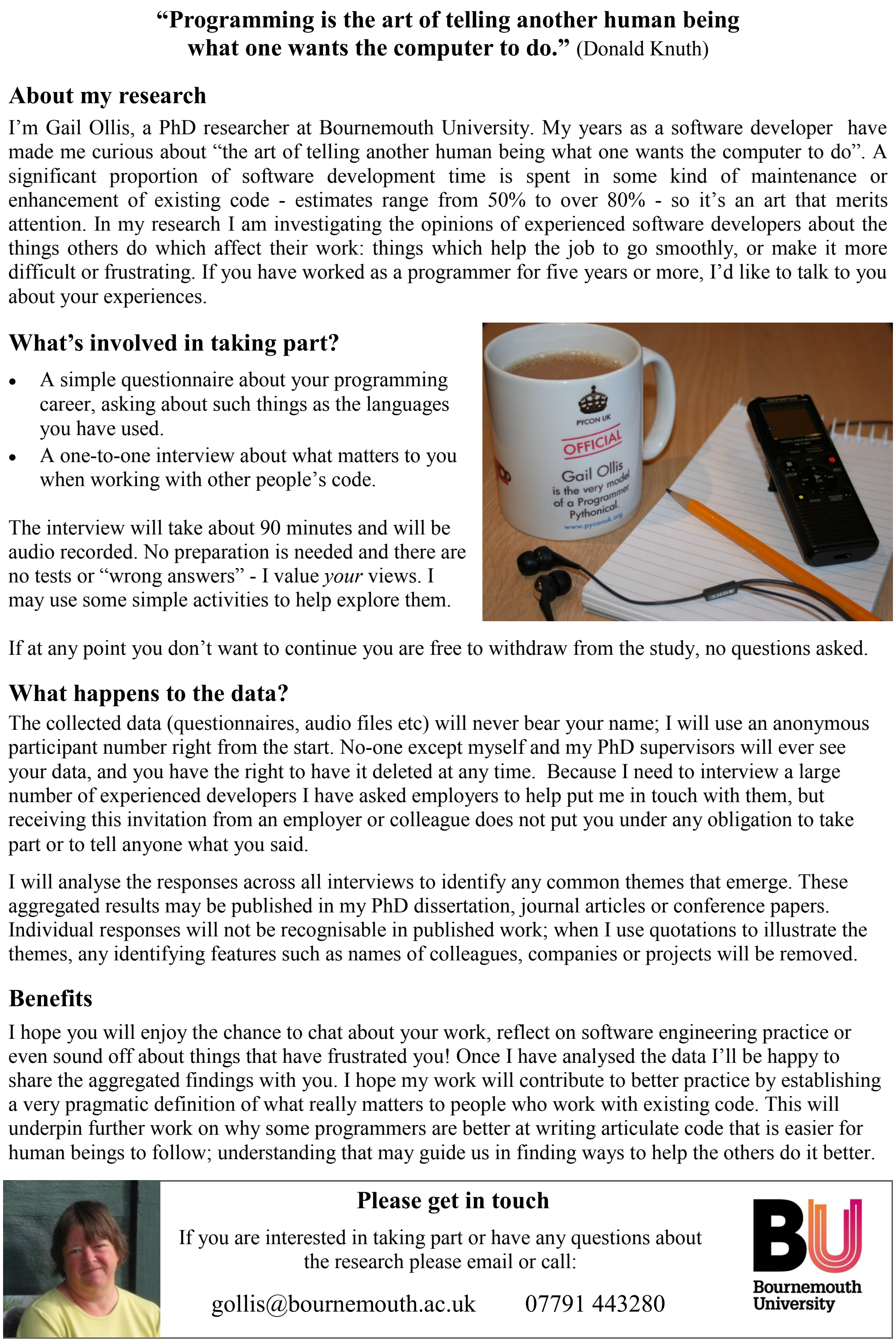
Yourdon, E. (2004). Death march. Upper Saddle River, N.J.: Prentice Hall Professional Technical Reference.

Yourdon, E., & Constantine, L. L. (1979). Structured design: Fundamentals of a discipline of computer program and systems design. Englewood Cliffs: Prentice-Hall.

Zhou, M., & Mockus, A. (2010). Developer Fluency: Achieving True Mastery in Software Projects. In Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering (pp. 137–146). New York: ACM. doi: 10.1145/1882291.1882313

Appendix A

Information sheet for Exploratory Study: individual participants



"Programming is the art of telling another human being what one wants the computer to do." (Donald Knuth)

About my research

I'm Gail Ollis, a PhD researcher at Bournemouth University. My years as a software developer have made me curious about "the art of telling another human being what one wants the computer to do". A significant proportion of software development time is spent in some kind of maintenance or enhancement of existing code - estimates range from 50% to over 80% - so it's an art that merits attention. In my research I am investigating the opinions of experienced software developers about the things others do which affect their work: things which help the job to go smoothly, or make it more difficult or frustrating. If you have worked as a programmer for five years or more, I'd like to talk to you about your experiences.

What's involved in taking part?

- A simple questionnaire about your programming career, asking about such things as the languages you have used.
- A one-to-one interview about what matters to you when working with other people's code.

The interview will take about 90 minutes and will be audio recorded. No preparation is needed and there are no tests or "wrong answers" - I value *your* views. I may use some simple activities to help explore them.

If at any point you don't want to continue you are free to withdraw from the study, no questions asked.

What happens to the data?

The collected data (questionnaires, audio files etc) will never bear your name; I will use an anonymous participant number right from the start. No-one except myself and my PhD supervisors will ever see your data, and you have the right to have it deleted at any time. Because I need to interview a large number of experienced developers I have asked employers to help put me in touch with them, but receiving this invitation from an employer or colleague does not put you under any obligation to take part or to tell anyone what you said.

I will analyse the responses across all interviews to identify any common themes that emerge. These aggregated results may be published in my PhD dissertation, journal articles or conference papers. Individual responses will not be recognisable in published work; when I use quotations to illustrate the themes, any identifying features such as names of colleagues, companies or projects will be removed.

Benefits

I hope you will enjoy the chance to chat about your work, reflect on software engineering practice or even sound off about things that have frustrated you! Once I have analysed the data I'll be happy to share the aggregated findings with you. I hope my work will contribute to better practice by establishing a very pragmatic definition of what really matters to people who work with existing code. This will underpin further work on why some programmers are better at writing articulate code that is easier for human beings to follow; understanding that may guide us in finding ways to help the others do it better.

Please get in touch

If you are interested in taking part or have any questions about the research please email or call:

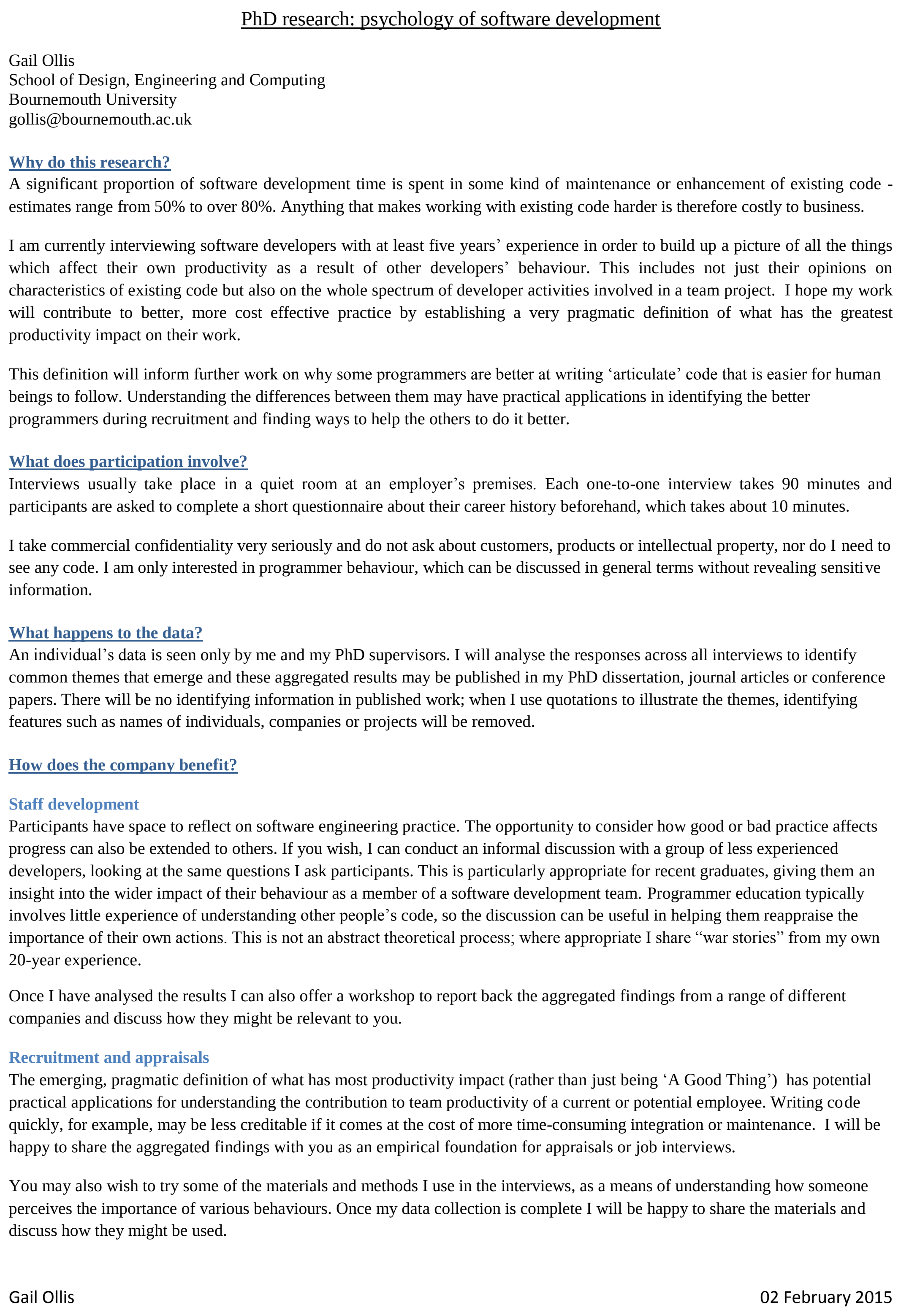
gollis@bournemouth.ac.uk 07791 443280

BU Bournemouth University

Figure 16: Participant information sheet for the Exploratory Study

Appendix B

Information sheet for the Exploratory Study: companies



PhD research: psychology of software development

Gail Ollis
School of Design, Engineering and Computing
Bournemouth University
gollis@bournemouth.ac.uk

Why do this research?

A significant proportion of software development time is spent in some kind of maintenance or enhancement of existing code - estimates range from 50% to over 80%. Anything that makes working with existing code harder is therefore costly to business.

I am currently interviewing software developers with at least five years' experience in order to build up a picture of all the things which affect their own productivity as a result of other developers' behaviour. This includes not just their opinions on characteristics of existing code but also on the whole spectrum of developer activities involved in a team project. I hope my work will contribute to better, more cost effective practice by establishing a very pragmatic definition of what has the greatest productivity impact on their work.

This definition will inform further work on why some programmers are better at writing 'articulate' code that is easier for human beings to follow. Understanding the differences between them may have practical applications in identifying the better programmers during recruitment and finding ways to help the others to do it better.

What does participation involve?

Interviews usually take place in a quiet room at an employer's premises. Each one-to-one interview takes 90 minutes and participants are asked to complete a short questionnaire about their career history beforehand, which takes about 10 minutes.

I take commercial confidentiality very seriously and do not ask about customers, products or intellectual property, nor do I need to see any code. I am only interested in programmer behaviour, which can be discussed in general terms without revealing sensitive information.

What happens to the data?

An individual's data is seen only by me and my PhD supervisors. I will analyse the responses across all interviews to identify common themes that emerge and these aggregated results may be published in my PhD dissertation, journal articles or conference papers. There will be no identifying information in published work; when I use quotations to illustrate the themes, identifying features such as names of individuals, companies or projects will be removed.

How does the company benefit?

Staff development

Participants have space to reflect on software engineering practice. The opportunity to consider how good or bad practice affects progress can also be extended to others. If you wish, I can conduct an informal discussion with a group of less experienced developers, looking at the same questions I ask participants. This is particularly appropriate for recent graduates, giving them an insight into the wider impact of their behaviour as a member of a software development team. Programmer education typically involves little experience of understanding other people's code, so the discussion can be useful in helping them reappraise the importance of their own actions. This is not an abstract theoretical process; where appropriate I share "war stories" from my own 20-year experience.

Once I have analysed the results I can also offer a workshop to report back the aggregated findings from a range of different companies and discuss how they might be relevant to you.

Recruitment and appraisals

The emerging, pragmatic definition of what has most productivity impact (rather than just being 'A Good Thing') has potential practical applications for understanding the contribution to team productivity of a current or potential employee. Writing code quickly, for example, may be less creditable if it comes at the cost of more time-consuming integration or maintenance. I will be happy to share the aggregated findings with you as an empirical foundation for appraisals or job interviews.

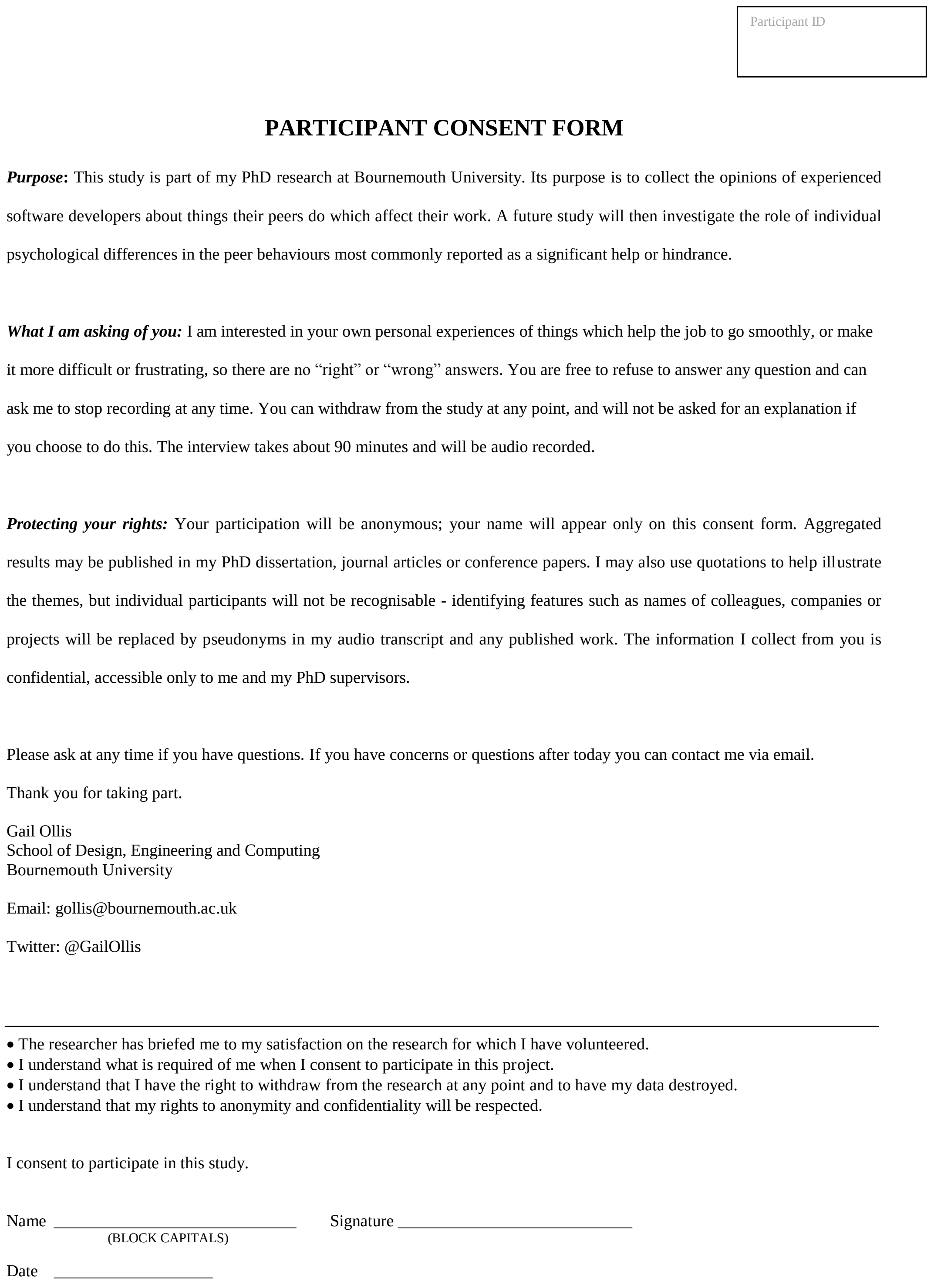
You may also wish to try some of the materials and methods I use in the interviews, as a means of understanding how someone perceives the importance of various behaviours. Once my data collection is complete I will be happy to share the materials and discuss how they might be used.

Gail Ollis 02 February 2015

Figure 17: Company information sheet for Exploratory Study

Appendix C

Consent form for the Exploratory Study



Participant ID

PARTICIPANT CONSENT FORM

Purpose**:** This study is part of my PhD research at Bournemouth University. Its purpose is to collect the opinions of experienced software developers about things their peers do which affect their work. A future study will then investigate the role of individual psychological differences in the peer behaviours most commonly reported as a significant help or hindrance.

What I am asking of you: I am interested in your own personal experiences of things which help the job to go smoothly, or make it more difficult or frustrating, so there are no “right” or “wrong” answers. You are free to refuse to answer any question and can ask me to stop recording at any time. You can withdraw from the study at any point, and will not be asked for an explanation if you choose to do this. The interview takes about 90 minutes and will be audio recorded.

Protecting your rights: Your participation will be anonymous; your name will appear only on this consent form. Aggregated results may be published in my PhD dissertation, journal articles or conference papers. I may also use quotations to help illustrate the themes, but individual participants will not be recognisable - identifying features such as names of colleagues, companies or projects will be replaced by pseudonyms in my audio transcript and any published work. The information I collect from you is confidential, accessible only to me and my PhD supervisors.

Please ask at any time if you have questions. If you have concerns or questions after today you can contact me via email.

Thank you for taking part.

Gail Ollis
School of Design, Engineering and Computing
Bournemouth University

Email: gollis@bournemouth.ac.uk

Twitter: @GailOllis

- The researcher has briefed me to my satisfaction on the research for which I have volunteered.
- I understand what is required of me when I consent to participate in this project.
- I understand that I have the right to withdraw from the research at any point and to have my data destroyed.
- I understand that my rights to anonymity and confidentiality will be respected.

I consent to participate in this study.

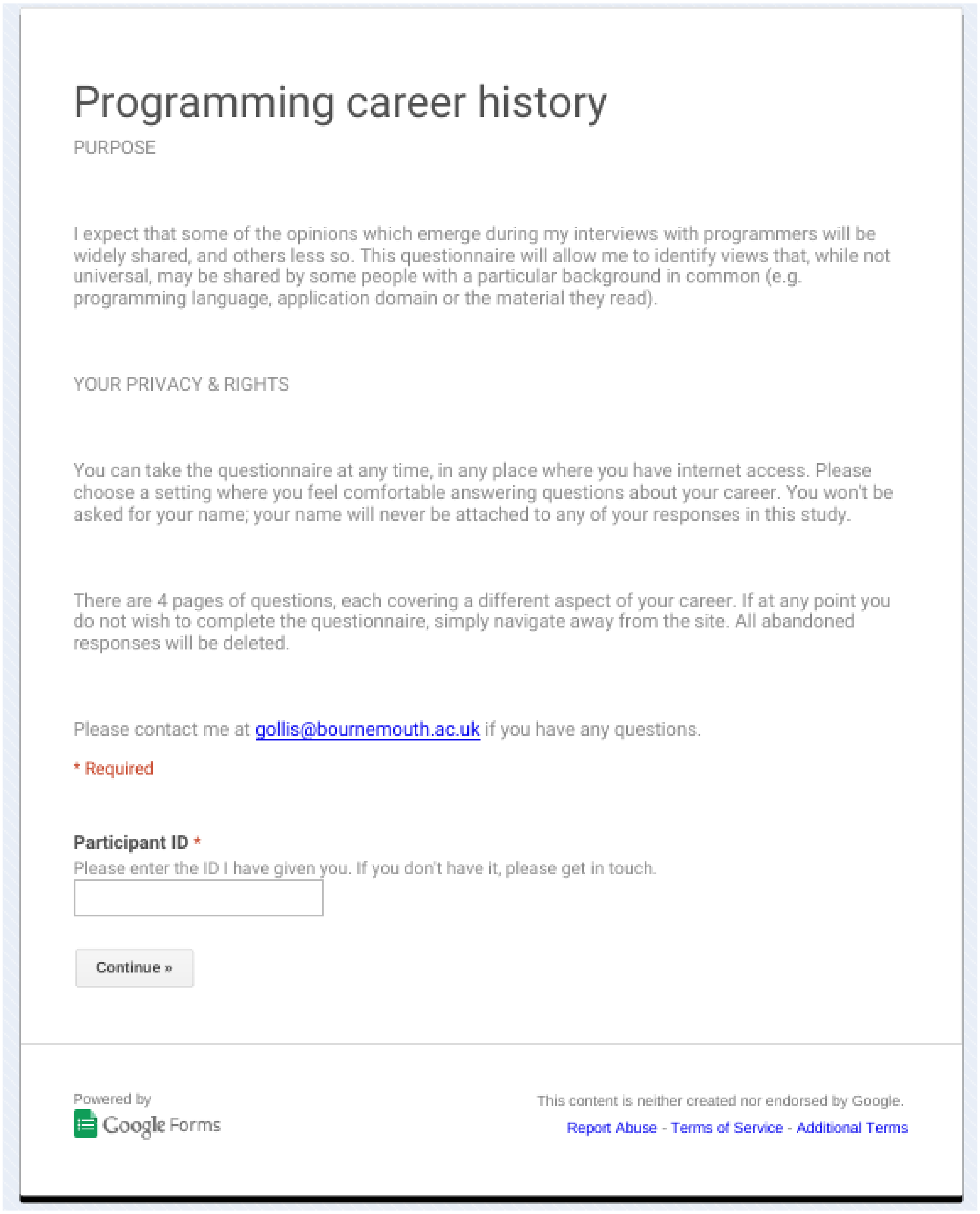
Name ____________________________ Signature ___________________________
(BLOCK CAPITALS)

Date ___________________

Figure 18: Consent form for Exploratory Study

Appendix D

Programming career history questions for the Exploratory Study



Programming career history

PURPOSE

I expect that some of the opinions which emerge during my interviews with programmers will be widely shared, and others less so. This questionnaire will allow me to identify views that, while not universal, may be shared by some people with a particular background in common (e.g. programming language, application domain or the material they read).

YOUR PRIVACY & RIGHTS

You can take the questionnaire at any time, in any place where you have internet access. Please choose a setting where you feel comfortable answering questions about your career. You won't be asked for your name; your name will never be attached to any of your responses in this study.

There are 4 pages of questions, each covering a different aspect of your career. If at any point you do not wish to complete the questionnaire, simply navigate away from the site. All abandoned responses will be deleted.

Please contact me at gollis@bournemouth.ac.uk if you have any questions.

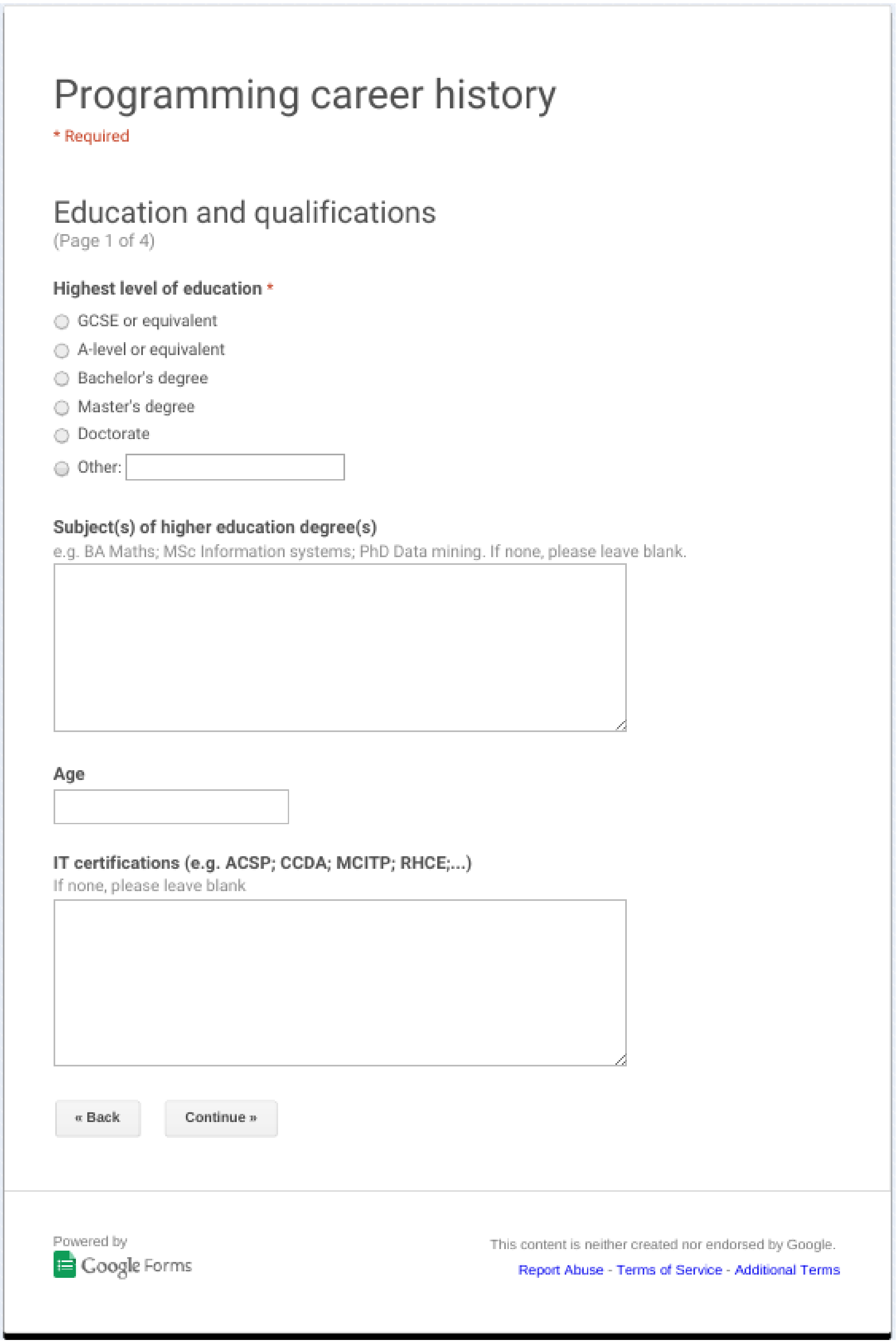
* Required

Participant ID *

Please enter the ID I have given you. If you don't have it, please get in touch.

Continue »
Powered by
Google Forms
This content is neither created nor endorsed by Google.
Report Abuse - Terms of Service - Additional Terms

Figure 19: Career history questionnaire: introductory page



Programming career history

* Required

Education and qualifications

(Page 1 of 4)

Highest level of education *

- ○ GCSE or equivalent
- ○ A-level or equivalent
- ○ Bachelor's degree
- ○ Master's degree
- ○ Doctorate
- ○ Other:

Subject(s) of higher education degree(s)

e.g. BA Maths; MSc Information systems; PhD Data mining. If none, please leave blank.

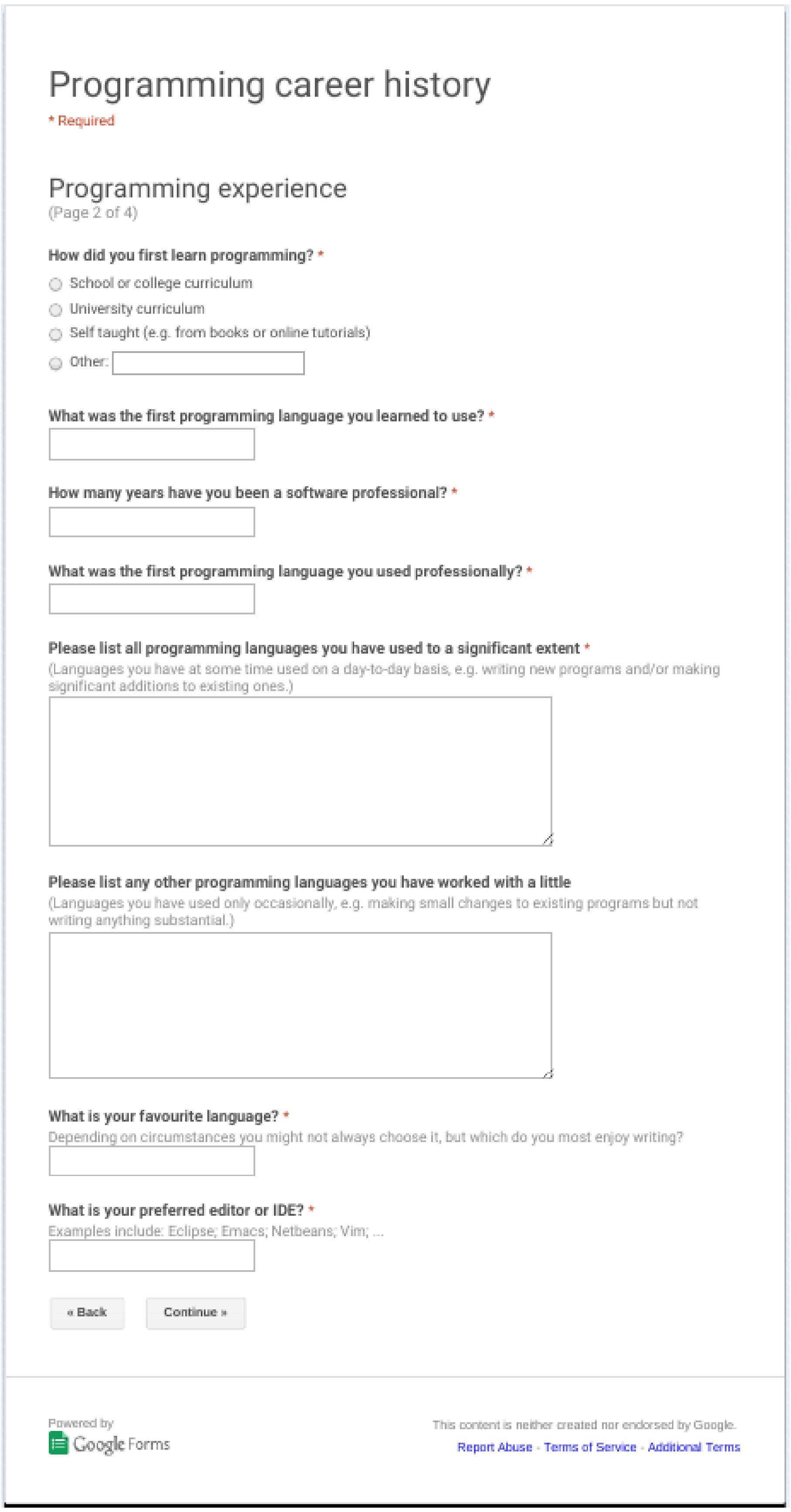
Age

IT certifications (e.g. ACSP; CCDA; MCITP; RHCE;...)

If none, please leave blank

« Back
Continue »
Powered by
Google Forms
This content is neither created nor endorsed by Google.
Report Abuse - Terms of Service - Additional Terms

Figure 20: Career history questionnaire: page 1



Programming career history

* Required

Programming experience

(Page 2 of 4)

How did you first learn programming? *

- ○ School or college curriculum
- ○ University curriculum
- ○ Self taught (e.g. from books or online tutorials)
- ○ Other:

What was the first programming language you learned to use? *

How many years have you been a software professional? *

What was the first programming language you used professionally? *

Please list all programming languages you have used to a significant extent *

(Languages you have at some time used on a day-to-day basis, e.g. writing new programs and/or making significant additions to existing ones.)

Please list any other programming languages you have worked with a little

(Languages you have used only occasionally, e.g. making small changes to existing programs but not writing anything substantial.)

What is your favourite language? *

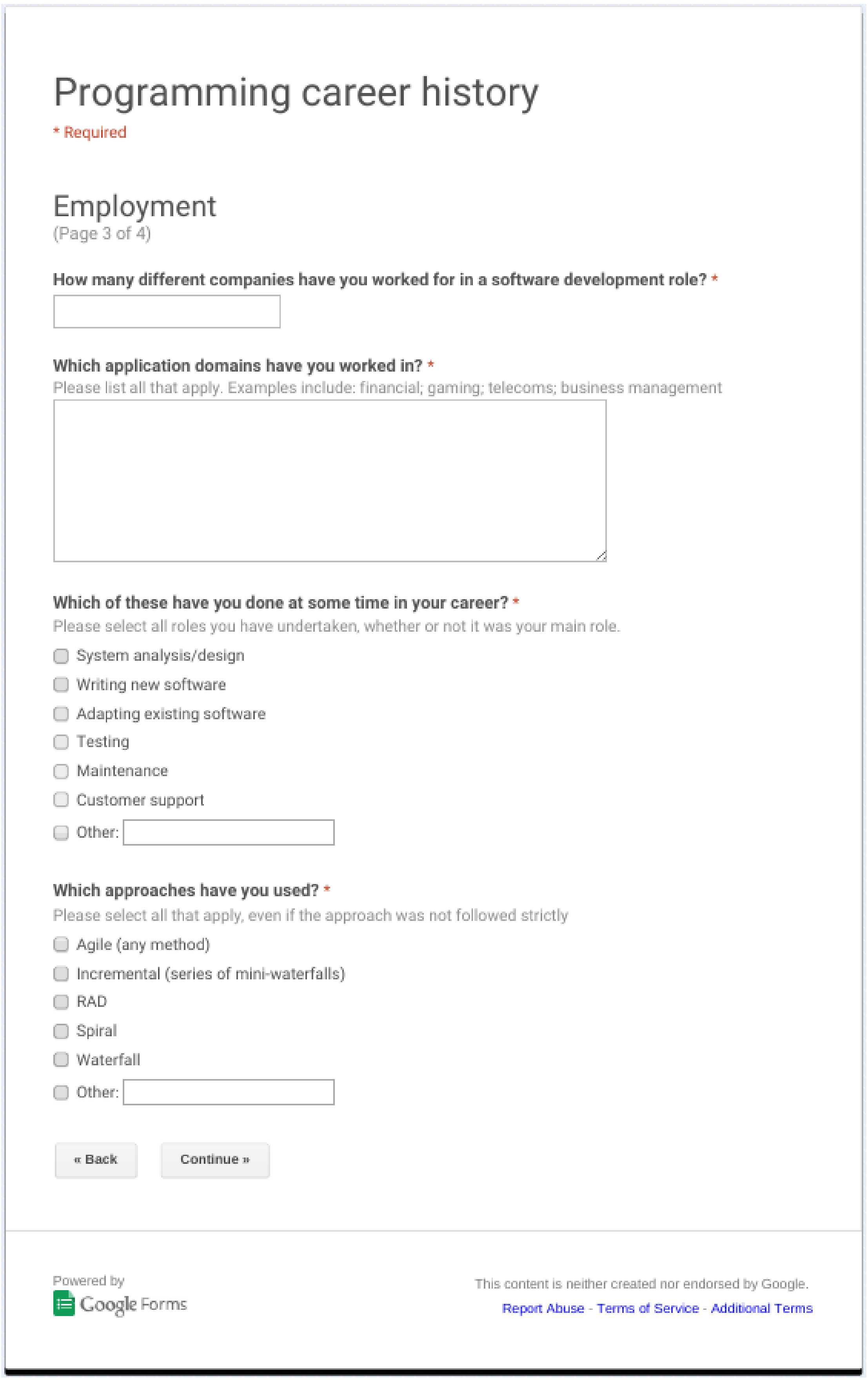
Depending on circumstances you might not always choose it, but which do you most enjoy writing?

What is your preferred editor or IDE? *

Examples include: Eclipse; Emacs; Netbeans; Vim; ...

« Back
Continue »
Powered by
Google Forms
This content is neither created nor endorsed by Google.
Report Abuse - Terms of Service - Additional Terms

Figure 21: Career history questionnaire: page 2


Programming career history
* Required

Employment
(Page 3 of 4)

How many different companies have you worked for in a software development role? *

Which application domains have you worked in? *
Please list all that apply. Examples include: financial; gaming; telecoms; business management

Which of these have you done at some time in your career? *
Please select all roles you have undertaken, whether or not it was your main role.

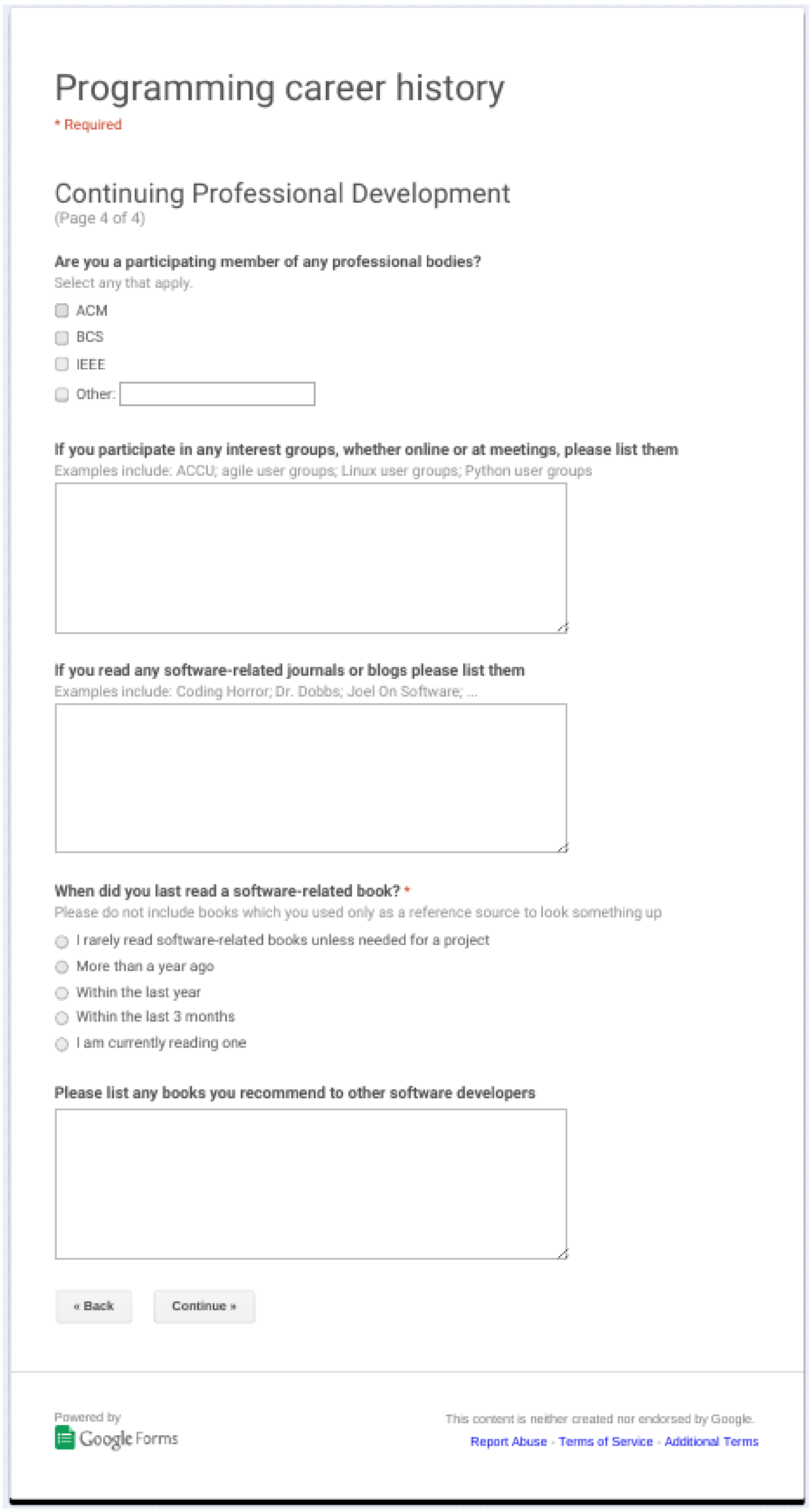
- ☐ System analysis/design
- ☐ Writing new software
- ☐ Adapting existing software
- ☐ Testing
- ☐ Maintenance
- ☐ Customer support
- ☐ Other:

Which approaches have you used? *
Please select all that apply, even if the approach was not followed strictly

- ☐ Agile (any method)
- ☐ Incremental (series of mini-waterfalls)
- ☐ RAD
- ☐ Spiral
- ☐ Waterfall
- ☐ Other:

« Back
Continue »
Powered by
Google Forms
This content is neither created nor endorsed by Google.
Report Abuse - Terms of Service - Additional Terms

Figure 22: Career history questionnaire: page 3



Programming career history

* Required

Continuing Professional Development

(Page 4 of 4)

Are you a participating member of any professional bodies?

Select any that apply.

- ☐ ACM
- ☐ BCS
- ☐ IEEE
- ☐ Other:

If you participate in any interest groups, whether online or at meetings, please list them

Examples include: ACCU; agile user groups; Linux user groups; Python user groups

If you read any software-related journals or blogs please list them

Examples include: Coding Horror; Dr. Dobbs; Joel On Software; ...

When did you last read a software-related book? *

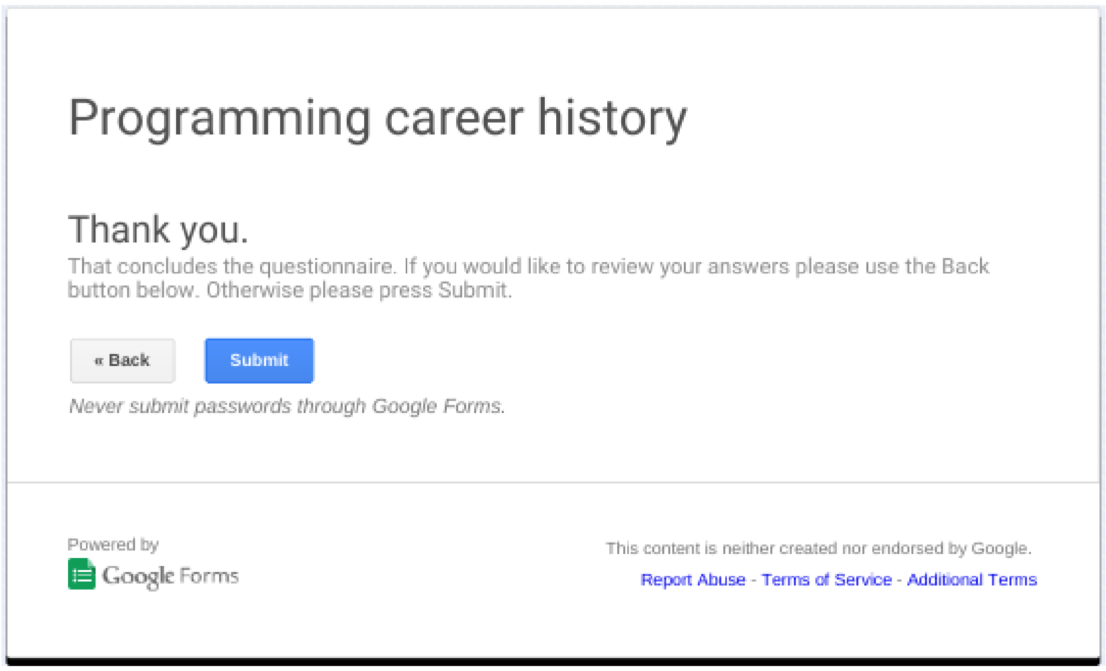
Please do not include books which you used only as a reference source to look something up

- ○ I rarely read software-related books unless needed for a project
- ○ More than a year ago
- ○ Within the last year
- ○ Within the last 3 months
- ○ I am currently reading one

Please list any books you recommend to other software developers

« Back
Continue »
Powered by Google Forms
This content is neither created nor endorsed by Google.
Report Abuse - Terms of Service - Additional Terms

Figure 23: Career history questionnaire: page 4



Programming career history

Thank you.

That concludes the questionnaire. If you would like to review your answers please use the Back button below. Otherwise please press Submit.

Never submit passwords through Google Forms.

This content is neither created nor endorsed by Google.

Figure 24: Career history questionnaire: submission page

Appendix E

Programming languages used by Exploratory Study participants

Languages used to a significant extent

This is a complete list of all the languages included in participants' responses to the question ”Please list all programming languages you have used to a significant extent”.

Ada / Ada 95	ADABAS Natural
Assembler (Intel x86 and x86_64)	Assembler (Motorola 68000)
Assembler (Z80)	Awk
Bash / shell	BASIC
BNCS/Applcore (BBC internal)	C
C++	C
COBOL	Common Lisp
Coral 66	CSS
DCL	Delphi
Eiffel	Elan
FORTRAN / FORTRAN IV-Plus / FORTRAN 77	HTML
IDL	Java
JavaScript	Lex/Yacc
LotusScript	Makefiles
Matlab	Modula-2
MS DOS batch files	Objective-C
Pascal	Perl
PHP	Powershell
Prolog	Python
R	Ruby
SQL	Visual Basic / VBA
VBS	

Additional languages

Participants also had limited experience of some further languages. Those listed below occured only in responses to the question ”Please list any other programming languages you have worked with a little”

ASP	BCPL
D	Erlang
F	Groovy
Haskell	Lisp
Lua	Microcode
Objective-C++	PL1
QBASIC	Scheme
Tcl	VB.Net

Appendix F

Interview card contents for Exploratory Study

Example card

0. A description of developer behaviour. e.g. ”Uses camelCase for variable names”

   Example:

   Some cards give examples to help clarify the described behaviour. These are not intended to be comprehensive; base your response on ANY symptoms you can think of which manifest the behaviour

Card deck

1. Gives each variable the smallest possible lifetime and scope

   Example:

   Declares a local object right before its first usage

2. Writes functions which retain state between calls

   Example:

   strtok (C library string tokeniser) uses static data to maintain the current position in the buffer

3. Seems to write a lot of lines of code to achieve a given outcome

4. Tends to ”own” code

   Example:

   * Does not like others to work on code they wrote

   * ”Resident expert” who always wants to be the one to write the device drivers

5. Uses long multi-part conditions

   Example:

   Uses a complex conditional expression with multiple operators , rather than nested conditions or intermediate boolean variables

6. Is willing to discuss suggestions about their code

   Example:

   * Asks for more information about suggested approach

   * Debates pros and cons

* Makes a change or explains a clear rationale for the existing approach

7. Makes data immutable whenever relevant

   Example:

   const float PI = 3.1415926535;

   final String exitMessage = ”Goodbye”;

8. Prefers <a programming language>and uses its idioms when coding in other languages

9. Automates tasks

   Example:

   * Running of tests

   * Building code

   * Generating documentation,etc

10. Uses horizontal and vertical spacing to align and separate code

    Example:

    * Alignment of related structures

    * An empty line between two sections

11. Includes accurate details of symptoms and how to reproduce the bug in their bug reports

12. Keeps the flow of control easy to follow

    Example:

    * Avoids GOTO

    * Uses break judiciously

    * Avoids deeply nested code

13. Follows encapsulation principles

    Example:

    Data structures and the processes that operate on them are in the same module , e.g. temperature data and the associated methods for conversion between units

14. Tries to provide early outline functionality that other team members can use

    Example:

    Defines APIs and implements some minimum functionality which will compile and run

15. Checks return values from function calls which may return error codes

    Example:

    Checks return value when opening a file for write

16. Follows formal methods to the letter

17. Is good at helping others

    Example:

    * Patient with less experienced developers

    * Explains how to fix a problem rather than taking the keyboard and just fixing it

18. Is rigorous about deallocating allocated resources

    Example:

    * malloc - free

    * file open - file close

    * new - delete, etc

19. Lets existing code constrain future code

    Example:

    * Tries to make minimal changes to existing structure

    * Is reluctant to refactor

20. Ignores build warnings

21. Accompanies each commit with a suitably informative message

22. Includes code features that are not currently needed

    Example:

    * Thinks it might be needed in future

    * Finds it easier to implement it than check if required

23. Uses the idioms of the programming language

    Example:

    * Lambda expressions in functional languages

    * List comprehensions in Python

    * Ternary operators in Java

    * Use of the STL in C++

24. Writes long if... else if... else if... blocks

25. Includes brackets which are not demanded by mathematical operator precedence (BODMAS)

    Example:

    fahrenheit = (celsius * 9.0 / 5.0) + 32.0;

26. Logs it in the issue-tracking system when knowingly making a sub-optimal change

    Example:

    Quick and dirty change to get customer's system working

27. Is always willing to consider that the bug may lie in their code

28. Commonly writes methods with 6 or more parameters

29. Catches exceptions at a level of the code where they cannot be resolved

30. Tries to leave a module a bit better than when they checked it out

    Example:

    * Splits up an over-long function

    * Improves a variable name

    * Boards up "broken windows" if there isn't time to fix them, e.g. display a "not implemented" message

31. Rarely uses exceptions as part of a program's normal flow

    Example:

    Assuming no unexpected events, the program could still run correctly if all exception handlers were removed.

32. Is often the person who breaks the build

    Example:

    * Missing files

    * Missing steps

    * Broken code, etc

33. Puts project goals over individual goals

    Example:

    Willing to give priority to helping fix a problem which is holding others back but does not affect their own current task

34. Prioritises performance in the design of their code

    Example:

    Code is optimised for speed over readability without any benefit to overall system performance

35. Asks questions without giving context

    Example:

    ”I’m getting an exception. Do you know what the problem is?”

36. Writes short, simple functions which perform a single task

37. Uses assignments within expressions

    Example:

    a = 3 / (b = c + 1) % d;

    rather than

    b = c + 1;,a = 3 / b % d;

38. Does not assume that a complex problem necessarily results in complex code

39. Fixes the symptoms without discovering the root cause of a bug

40. Finds out whether functionality is already available before writing their own implementation

    Example:

    * Uses project libraries

    * Uses language libraries

    * Uses frameworks

41. Includes useful logging messages in their code

    Example:

    * Errors, warnings and info are clearly distinguished

    * Content of messages is informative and succinct

    * Quantity of messages is sufficient but not verbose

42. Doesn’t always update or add tests when changing code

    Example:

    * Does not create tests for new code they write

    * Does not update tests for code they modify/extend

    * Does not add a new test to trap the bug they are fixing

43. Espouses ”one true way” of doing things

44. Includes brackets which are not demanded by the language’s operator precedence

    Example:

    result = operand « (a + b);

    if ((day == 31) && (month == 12) && (year == 1999))

45. Assumes that things which ”can’t happen” won’t

46. Practises good housekeeping

    Example:

    * Removes temporary debug statements

    * Does not leave TODOs for others to deal with

47. Is willing to ask questions

    Example:

    * Asks whether there is existing code to do X

    * Asks for advice on programming issues

    * Asks for explanation of domain-specific concepts

    * Asks for clarification of a requirement rather than making assumptions about the correct interpretation

48. Codes using implementation terms rather than domain terms

    Example:

    if (list.contains(user.getId()))

    rather than

    if (user.isAuthorised())

49. Follows the DRY (Don’t Repeat Yourself) principle

    Example:

    * Avoids ”copy-and-paste” coding

    * Contributes code to project libraries when they notice common use of functionality

50. Tends to work in isolation

    Example:

    * Checks in their work infrequently

    * Rarely integrates their work with others

51. Makes APIs easy to use correctly

    Example:

    * Documents the API

    * Designs APIs which seem natural and obvious, not for the convenience of the underlying implementation

52. Tends to apply a favourite pattern regardless of context

53. Uses code comments in ways that aid understanding

    Example:

    * Explains the domain logic of the code

    * Documents design decisions, e.g. assumptions; alternatives discarded; trade-offs; workarounds

    * Explains the task done by the following section of code

    * Updates existing comments to match changes to the code

    * Does not obscure the structure of the code

    * Comments only what the code cannot say

54. Chooses identifiers which are not succinct, meaningful and distinct

    Example:

    * Spelling mistakes: String passwrod;

    * Silent differences: oldPwd & oldpwd; userInput & user_input

    * Meaningless: String s; float ex; int foo; doStuff();

    * Misleading: getData() actually writes data to disk

    * ”Cute”: int avadaKedavra; // exit code when system dies

    * Superfluous: temperatureData vs temperature

    * Verbose: performReconciliation() vs reconcile()

    * Nouns for function names: conversion() vs convert()

Appendix G

Complete template

Since the final template does not fit on a single page, an outline is given first to show the top level codes and their immediate sub-codes. This is followed by sections of the template showing a complete hierarchy for each of the top level codes.

- Consequences
  - Ease of comprehension
  - Ease of debugging
  - Ease of maintenance
  - Fitness for purpose
  - Hard to test
  - Knowledge and understanding
  - Non-issues
  - Progress
  - State of mind
  - Volume of code
- Dialogue
  - Alignment
  - Importance of feedback
  - Level of formality
  - Working together
- External factors
  - Company culture
  - Practices
  - Prototypes
  - Team environment
  - Time
- Future topics
  - Motivation
  - Use as an appraisal/development tool
- Nature of the job
  - Lack of empirical data
  - Most work is with existing code
  - Understanding requirements
- Personal qualities
  - Aptitude
  - Lacking empathy
  - Learning and experience
  - Outlook and approach
  - Role
  - Ways of behaving
- Reflection on process
  - Benefits to participant
  - Card content
  - Card relevance
  - Card topic raised spontaneously
- Programmer behaviours
  - New topics
  - Topics derived from cards

Figure 25: Template outline: top level codes and immediate sub-codes

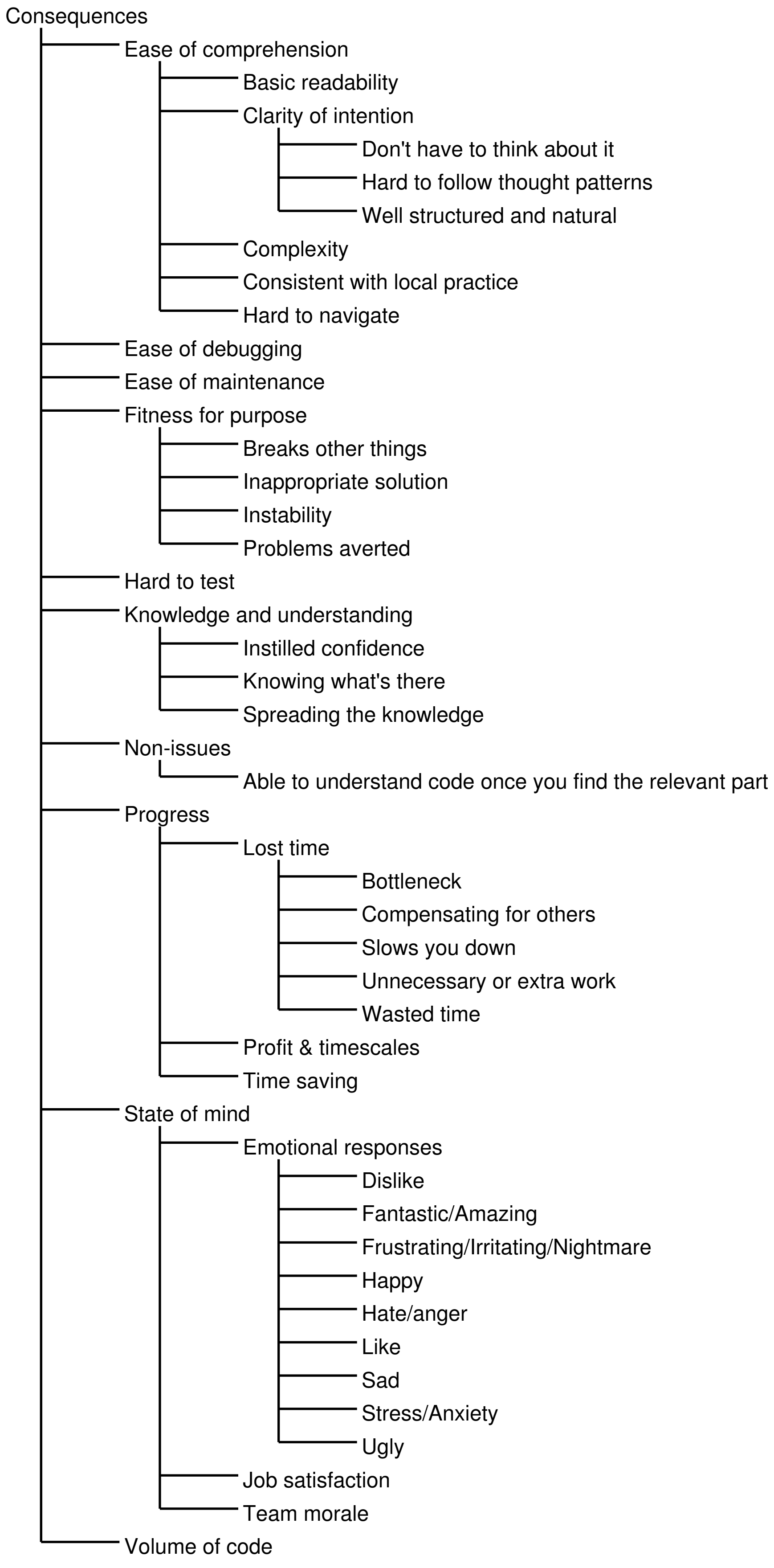
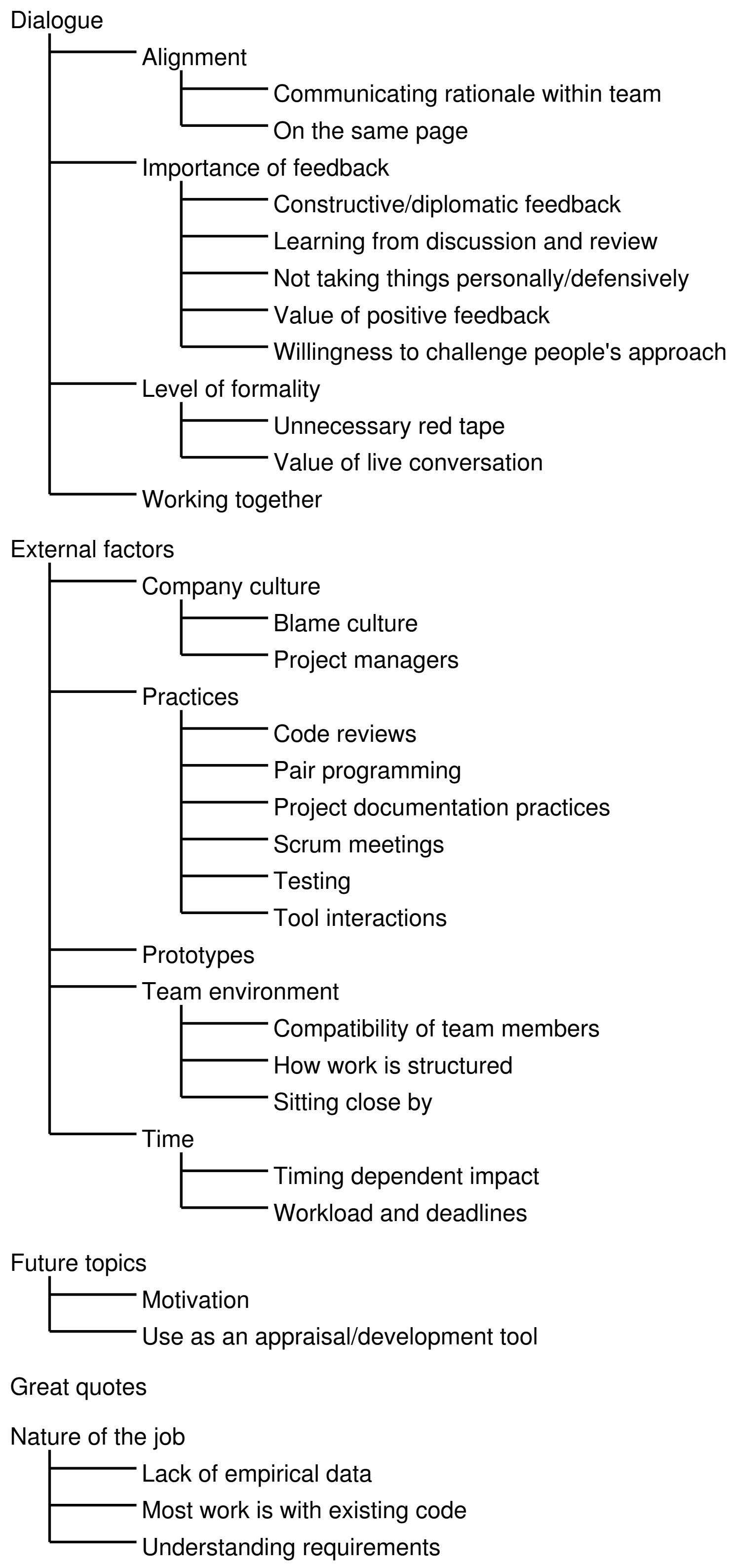


Figure 26: Template section: consequences



- Dialogue
  - Alignment
    - Communicating rationale within team
    - On the same page
  - Importance of feedback
    - Constructive/diplomatic feedback
    - Learning from discussion and review
    - Not taking things personally/defensively
    - Value of positive feedback
    - Willingness to challenge people's approach
  - Level of formality
    - Unnecessary red tape
    - Value of live conversation
  - Working together
- External factors
  - Company culture
    - Blame culture
    - Project managers
  - Practices
    - Code reviews
    - Pair programming
    - Project documentation practices
    - Scrum meetings
    - Testing
    - Tool interactions
  - Prototypes
  - Team environment
    - Compatibility of team members
    - How work is structured
    - Sitting close by
  - Time
    - Timing dependent impact
    - Workload and deadlines
- Future topics
  - Motivation
  - Use as an appraisal/development tool
- Great quotes
- Nature of the job
  - Lack of empirical data
  - Most work is with existing code
  - Understanding requirements

Figure 27: Template section: dialogue, external factors, future topics and nature of the job

- Personal qualities
  - Aptitude
    - Good intentions
    - Craft skills
    - Non-technical people
  - Lacking empathy
  - Learning and experience
    - Inexperience
    - Insight gained from experience
      - Healthy scepticism
    - Learned habits
    - Missing full product lifecycle
    - Willingness to learn
  - Outlook and approach
    - Don't look for alternative approach
    - New toys
    - Reluctance to change solution
    - See the bigger picture
  - Role
    - Job protection
    - Loss of influence
  - Ways of behaving
    - Adult
    - Negativity
    - Professional responsibilities
    - Team player
- Reflection on process
  - Benefits to participant
  - Card content
  - Card relevance
  - Card topic raised spontaneously

Figure 28: Template section: personal qualities and reflection on interview process

- Programmer behaviours
  - New topics
    - Excessive configurability
    - Insufficient documentation
    - New technologies without good reason
    - Over-complex process
    - Rules vs principles
    - Runs appropriate tests
    - Slapdash code
    - Takes responsibility to follow through

Figure 29: Template section: programmer behaviours (part 1 of 3)

- Programmer behaviours (continued)
  - Topics derived from cards
    - Asks questions
    - Assignments in expressions
    - Automates tasks
      - Over-automates
    - Block size
    - Breaks build
    - Comments
      - Anti-comments
        - Good code doesn't need comments
        - Irrelevance of comments
        - Misleading comments
        - Too many comments
        - Unhelpful comments
      - Pro-comments
        - Situations needing comments
        - Too few comments
        - Useful act for the writer
      - Size of comments
    - Commit messages
    - Complex conditionals
    - Constrained by existing code
      - Downside of refactoring
      - Unsympathetic changes
      - When to refactor
    - DRY
    - Deallocating resources
    - Detailed bug reports
    - Discusses suggestions
    - Encapsulation
      - Easy APIs
      - Modularity
      - Too much indirection
    - Error handling
      - Assumes errors can't happen
      - Checks error codes
    - Excessive code
      - Creating a monster
      - Over-complex solution
    - Fixes symptoms - not root cause
    - Functions which retain state
    - Helping others
      - Superhero syndrome
    - Identifiers
      - Long names
      - Meaningful name
      - Mis-spelled
      - Poor or downright misleading names

Figure 30: Template section: programmer behaviours (part 2 of 3)

- Programmer behaviours (continued)
  - Topics derived from cards
    - Idiomatic code
    - Ignores build warnings
      - Insists on removal of warnings
    - Immutable data
    - Irrelevant code
      - Dead wood
      - You ain't gonna need it
    - Leave it better
    - Limits scope
    - Logging
      - Informative logging
      - Misuse of logging
      - Too little logging
      - Verbose or excessive logging
    - Logs suboptimal change
    - Looks for existing functionality
    - Makes control flow easy to follow
    - Many parameters
    - Misused exceptions
    - Non-essential brackets
    - One True Way
    - Outline functionality
    - Over-uses favourite pattern
    - Owns code
    - Poor abstraction
    - Preferred tools
    - Premature optimisation
    - Prioritises project goals
    - Simplifies complex problems
    - Strict adherence to methods
    - Tests not written
    - Vague questions
    - Whitespace
    - Willing to admit fault
    - Works in isolation

Figure 31: Template section: programmer behaviours (part 3 of 3)

Appendix H

Workshop quotation card contents

1. Anyone who thinks it's ”their” code is missing the point.

2. They'll just want to get their own way, rather than do what's best for th e product.

3. Automating's beautiful.

4. Until you've actually investigated a bug report that someone else has wri tten you don't realise how easy it is to write bad ones.

5. A lot of code looks like nobody went back and re-read it. It's all just s tream of consciousness; there's no narrative flow to it.

6. If someone makes it harder to approach then you're forced to choose less frequent questions.

7. Implement the code to do what you're trying to achieve, not try and guess what might be used in the future.

8. If you think the bug is always in somebody else's code you'll be spending a lot of time looking in the wrong place.

9. It's a pain in the butt if the build's broken and you've got to fix it be fore you can get on with your real work.

10. I like small files, small functions, small classes.

11. The simple solution is sometimes the best

12. It's a bit of a quick fix and then you come around to it three times agai n. And it's ”maybe we should have just sorted it out in the first place”

13. Someone not reinventing the wheel? Cor! Who'd have thought it!

14. Quality of log messages is more important than number of messages

15. People get very dogmatic. If you disagree or voice concerns you're a here tic and you're burned at the stake

16. People just do what they think is right because they don't want to be ars ed

talking to someone.

17. Finding the same block repeated several times makes it a lot harder to un derstand... you're trying to mentally diff.

18. I think you do need to discuss stuff with people.

19. If you have an interface that is natural and obvious then it leaves you f reer to get on with your job. It does what it says on the tin.

20. Identifiers are very, very important.

21. A lot of times I find people are more driven by the latest technologies t han by understanding the problem.

Appendix I

Workshop instructions

These cards are direct quotes of things experienced developers have said about how they are affected by things other developers do. They represent common themes that came up across the different companies I've talked to.

Please spend some time looking throught the cards and ask yourself: How does it affect ME when someone does this?" This is not about good practice; it's about the impact on you, good or bad, when someone behaves in a certain way.

I'll give you time to choose a card that reminds you of something you've experienced. Think about why it speaks to you. When everyone has one, I'll ask you to explain its significance.That should not be good practice, or code performance, or profit, or anything like that. You should be entirely self-interested about its impact on you!

Others might not necessarily have experienced it the same way so there are a few ground rules for talking about the cards:

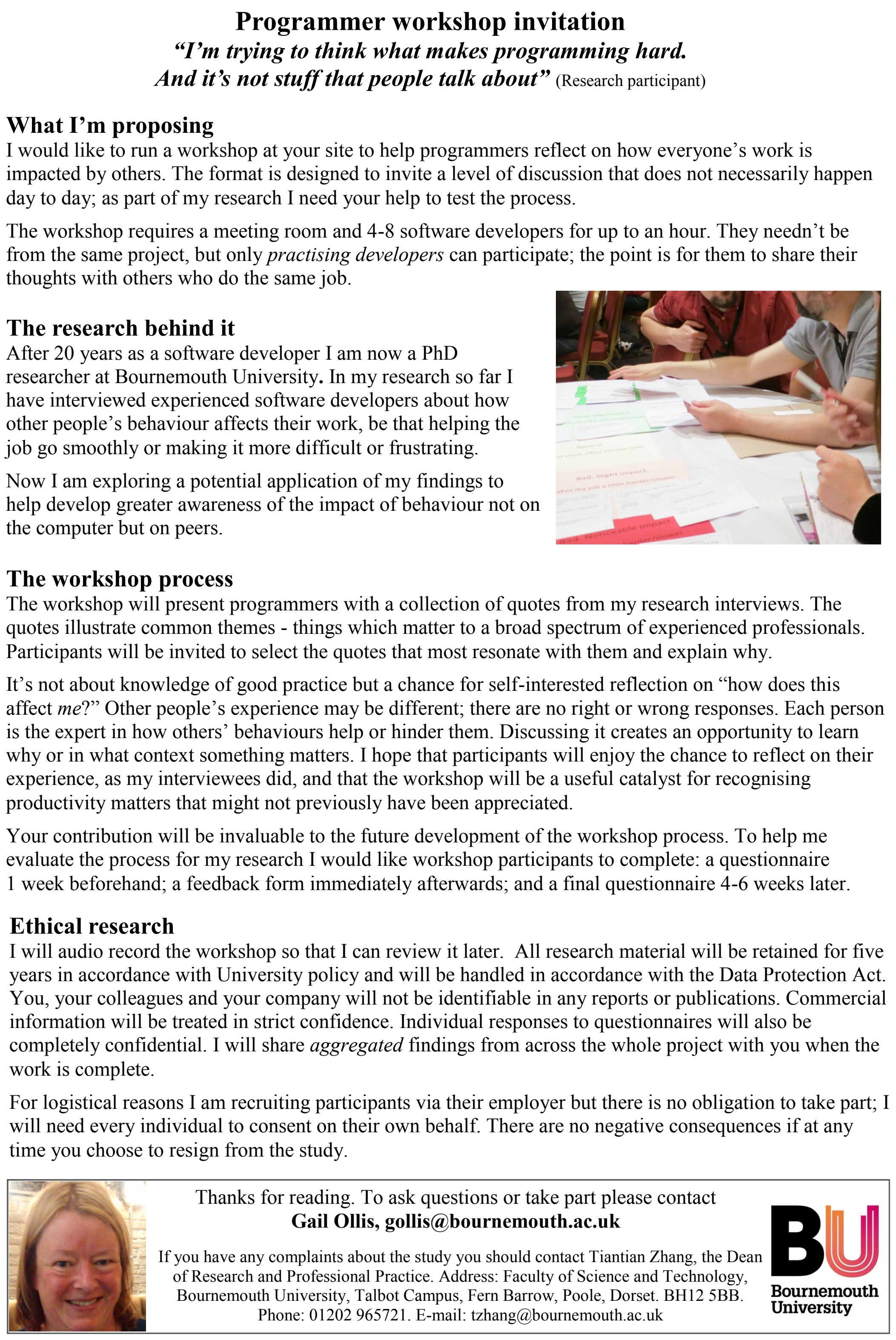
1. Try not to assume anything is "obvious". Explain the impact this thing has on you. The "side-effects" of doing something aren't necessarily obvious unless you're the one affected, so explain why this thing matters. (The ass is here to remind you that there should be no ASSumptions!)

2. No-one is "wrong" about how they experienced something. If your personal experience is different, you can explain how and explore the circumstances that might account for the differences (e.g., context). (The steamroller is here to remind you not to steamroller anyone's account of what they've experienced!)

3. Remember, the question is not "is this good practice?" but "how does it affect ME when someone does this?" Choose a card for the impact that practice has on YOU, and not because it reflects some "good practice" principle. (The cow is here to remind you: no sacred cows.)

Appendix J

Information sheet for workshop participants



Programmer workshop invitation

"I'm trying to think what makes programming hard. And it's not stuff that people talk about" (Research participant)

What I'm proposing

I would like to run a workshop at your site to help programmers reflect on how everyone's work is impacted by others. The format is designed to invite a level of discussion that does not necessarily happen day to day; as part of my research I need your help to test the process.

The workshop requires a meeting room and 4-8 software developers for up to an hour. They needn't be from the same project, but only *practising developers* can participate; the point is for them to share their thoughts with others who do the same job.

The research behind it

After 20 years as a software developer I am now a PhD researcher at Bournemouth University**.** In my research so far I have interviewed experienced software developers about how other people's behaviour affects their work, be that helping the job go smoothly or making it more difficult or frustrating.

Now I am exploring a potential application of my findings to help develop greater awareness of the impact of behaviour not on the computer but on peers.

The workshop process

The workshop will present programmers with a collection of quotes from my research interviews. The quotes illustrate common themes - things which matter to a broad spectrum of experienced professionals. Participants will be invited to select the quotes that most resonate with them and explain why.

It's not about knowledge of good practice but a chance for self-interested reflection on "how does this affect *me*?" Other people's experience may be different; there are no right or wrong responses. Each person is the expert in how others' behaviours help or hinder them. Discussing it creates an opportunity to learn why or in what context something matters. I hope that participants will enjoy the chance to reflect on their experience, as my interviewees did, and that the workshop will be a useful catalyst for recognising productivity matters that might not previously have been appreciated.

Your contribution will be invaluable to the future development of the workshop process. To help me evaluate the process for my research I would like workshop participants to complete: a questionnaire 1 week beforehand; a feedback form immediately afterwards; and a final questionnaire 4-6 weeks later.

Ethical research

I will audio record the workshop so that I can review it later. All research material will be retained for five years in accordance with University policy and will be handled in accordance with the Data Protection Act. You, your colleagues and your company will not be identifiable in any reports or publications. Commercial information will be treated in strict confidence. Individual responses to questionnaires will also be completely confidential. I will share *aggregated* findings from across the whole project with you when the work is complete.

For logistical reasons I am recruiting participants via their employer but there is no obligation to take part; I will need every individual to consent on their own behalf. There are no negative consequences if at any time you choose to resign from the study.

Thanks for reading. To ask questions or take part please contact
Gail Ollis, gollis@bournemouth.ac.uk

If you have any complaints about the study you should contact Tiantian Zhang, the Dean of Research and Professional Practice. Address: Faculty of Science and Technology, Bournemouth University, Talbot Campus, Fern Barrow, Poole, Dorset. BH12 5BB. Phone: 01202 965721. E-mail: tzhang@bournemouth.ac.uk

Bournemouth University

Figure 32: Workshop participant information sheet

Appendix K

Workshop planning routemap

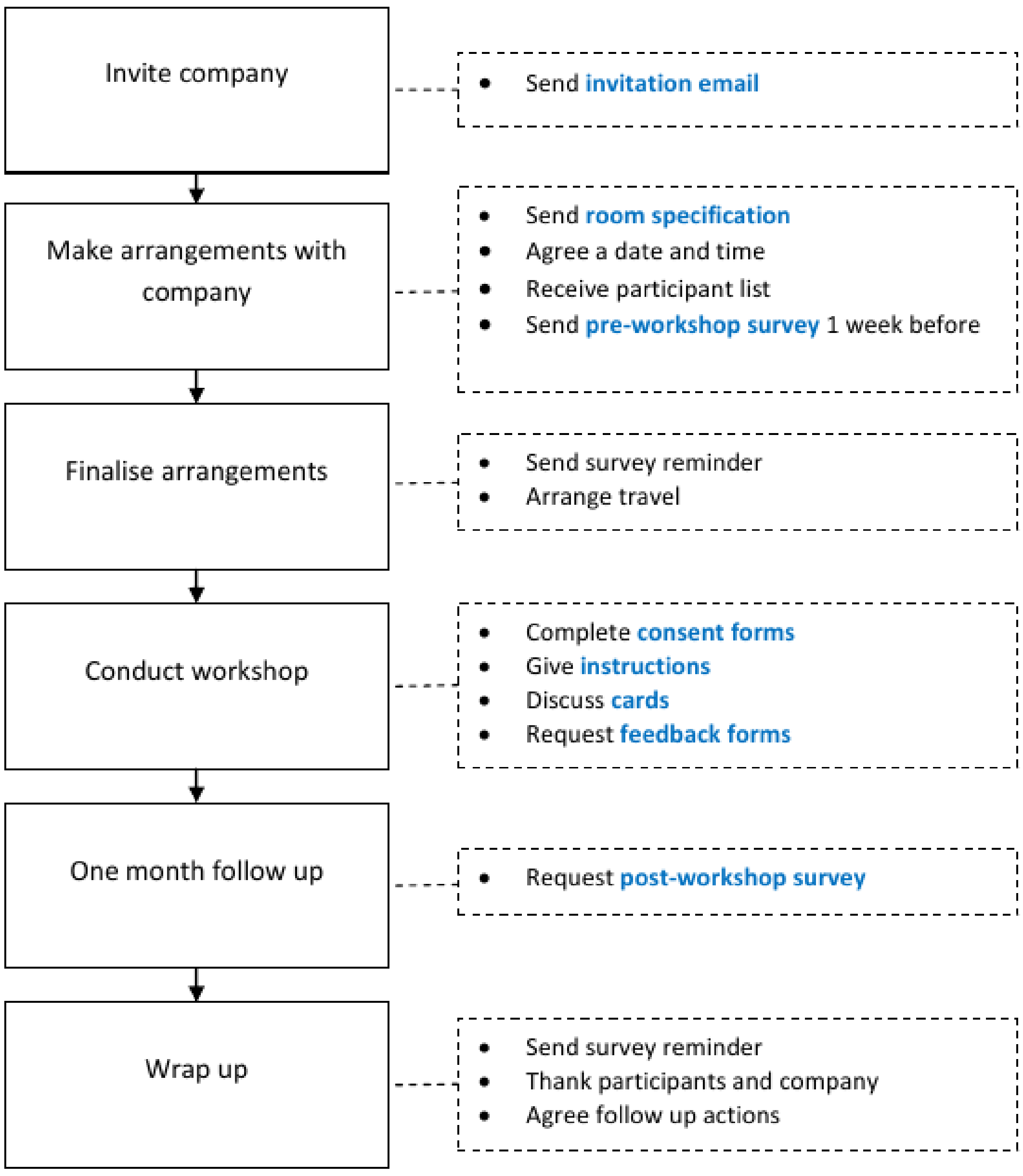


Figure 33: Workshop plan

Appendix L

Pre-workshop career experience questions

The pre-workshop survey was a single online survey consisting of two sections. The first section, whose questions are listed here, asked about the participant's career experience. The second contained the team stage questionnaire (Appendix M).

This section asks about your current job and your career to date.

1. How would you describe the industry domain you work in? (e.g. telecommunications; games; enterprise...)

2. How would you describe your role? (e.g. junior software developer; software team lead...)

3. Which one of these activities accounts for most time in your current job?

   a. System analysis/design
   b. Writing new software
   c. Adapting existing software
   d. Testing
   e. Maintenance
   f. Customer support
   g. Other

   If you selected "Other"; what is the activity that accounts for most time in your current job?

4. How many people are in your current team, approximately?

5. How many years have you been working as a software developer?

6. How many companies have you worked at before this one?

7. What other industry domains have you worked in?

8. What activities have you undertaken previously, at any point in your career? Select all that apply.

( ) System analysis/design
( ) Writing new software
( ) Adapting existing software
( ) Testing
( ) Maintenance
( ) Customer support
( ) Other

If you included "Other"; what other activities have you undertaken in your career?

Appendix M

Pre-workshop team stage questions

The questionnaire (Clark, 2016) was used without modification. It is reproduced here under the terms of the Creative Commons Attribution-Noncommercial 2.0 Generic license (Creative Commons, n.d.).

Questions included in the working environment survey

For each statement click one answer to indicate how often the group you work in displays that behaviour.

1	Almost never
2	Seldom
3	Occasionally
4	Frequently
5	Almost always

If you wish to add any comments you'll be able to do so at the end.

1. We try to have set procedures or protocols to ensure that things are orderly and run smoothly (i.e. minimize interruptions, all get the opportunity to have their say)

2. We are quick to get to the task at hand and do not spend much time in the planning stage

3. Our team members feel that we are all in it together and we share responsibility for the team's success or failure

4. We have thorough procedures for agreeing on our goals and planning the way we will perform our tasks

5. Team members are afraid to ask others for help

6. We take our team's goals literally and assume a shared understanding

7. The team leader tries to keep order and contributes to the task at hand

8. We do not have fixed procedures; we make them up as the task or project progresses

9. We generate lots of ideas, but we don't use many of them because we fail to listen carefully and tend to reject them without fully understanding them

10. Team members do not fully trust the other members and tend to closely monitor others who are working on a specific task

11. The team leader or facilitator ensures that we follow the procedures, do not

argue, do not interrupt, and keep to the point

12. We enjoy working together; we have a fun and productive time
13. We have accepted each other as members of the team
14. The team leader is democratic and collaborative
15. We are trying to define the team's goals and what tasks need to be accomplished
16. Many of the team members have their own ideas about the team's process; personal agendas are rampant
17. We fully accept each other's strengths and weaknesses
18. We assign specific roles to team members (team leader, facilitator, time keeper, note taker, etc.)
19. We try to achieve harmony by avoiding conflict
20. The team's tasks are very different from what we imagined and seem very difficult to accomplish
21. There are many abstract discussions of the concepts and issues; some members are impatient with these discussions
22. We are able to work through group problems
23. We argue a lot even though we agree on the real issues
24. The team is often tempted to go beyond the original scope of the project
25. We express criticism of others constructively
26. There is a close attachment to the team
27. It seems as if little is being accomplished towards the team's goals
28. The goals we have established seem unrealistic

29. Although we are not fully sure of the project's goals and issues, we are excited and proud to be on the team

30. We feel like we can share personal problems with each other whenever we need to do so

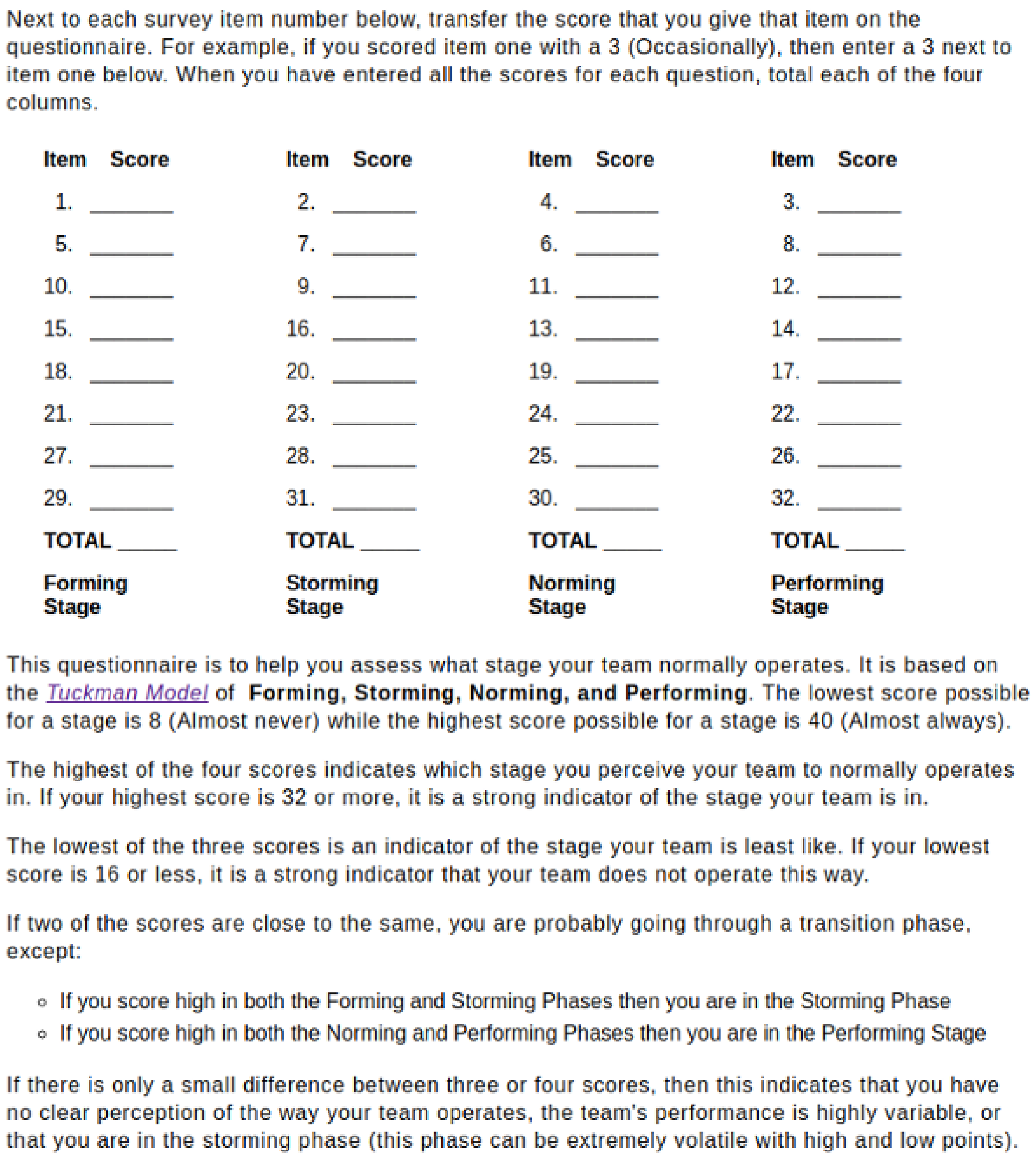
31. There is a lot of resistance to the tasks at hand or to quality improvement approaches

32. We get a lot of work done

If you wish to add any comments please write them here.

Scoring

Scoring was calculated as shown in Figure 34.



Next to each survey item number below, transfer the score that you give that item on the questionnaire. For example, if you scored item one with a 3 (Occasionally), then enter a 3 next to item one below. When you have entered all the scores for each question, total each of the four columns.

Item	**Score**	**Item**	**Score**	**Item**	**Score**	**Item**	**Score**
1.	_______	2.	_______	4.	_______	3.	_______
5.	_______	7.	_______	6.	_______	8.	_______
10.	_______	9.	_______	11.	_______	12.	_______
15.	_______	16.	_______	13.	_______	14.	_______
18.	_______	20.	_______	19.	_______	17.	_______
21.	_______	23.	_______	24.	_______	22.	_______
27.	_______	28.	_______	25.	_______	26.	_______
29.	_______	31.	_______	30.	_______	32.	_______
TOTAL	_____	**TOTAL**	_____	**TOTAL**	_____	**TOTAL**	_____
Forming Stage		**Storming Stage**		**Norming Stage**		**Performing Stage**	

This questionnaire is to help you assess what stage your team normally operates. It is based on the *Tuckman Model* of **Forming, Storming, Norming, and Performing**. The lowest score possible for a stage is 8 (Almost never) while the highest score possible for a stage is 40 (Almost always).

The highest of the four scores indicates which stage you perceive your team to normally operates in. If your highest score is 32 or more, it is a strong indicator of the stage your team is in.

The lowest of the three scores is an indicator of the stage your team is least like. If your lowest score is 16 or less, it is a strong indicator that your team does not operate this way.

If two of the scores are close to the same, you are probably going through a transition phase, except:

- If you score high in both the Forming and Storming Phases then you are in the Storming Phase
- If you score high in both the Norming and Performing Phases then you are in the Performing Stage

If there is only a small difference between three or four scores, then this indicates that you have no clear perception of the way your team operates, the team's performance is highly variable, or that you are in the storming phase (this phase can be extremely volatile with high and low points).

Figure 34: Questionnaire scoring

Appendix N

Immediate post-workshop questions

DEBRIEF

Thank you very much for participating in this test of the workshop format. The purpose was to see if the format can facilitate conversation among software developers about things other than inherent technical complexity that affect the difficulty of their job. I hope you found it interesting.

There were no “right” or “wrong” answers; this study is not about participants’ opinions on the topics they choose to discuss, but whether the workshop format works. To help me evaluate this please can you give some feedback below on your experience and impressions today? Finally, I’ll ask for your help with a short questionnaire in a few weeks’ time to see if there was any lasting impression. You are free to withdraw from this study at any time and have data which is not already anonymous deleted.

LOGISTICS

For multiple choice questions, please circle the answer which most closely matches your opinion. There is space for you to expand on any answer if you wish. Skip any question if you prefer not to answer it.

1. Was it clear what you were supposed to do?
   - Very unclear
   - A little unclear
   - Reasonably clear
   - Very clear
2. Did you get a chance to contribute?
   - Not at all
   - Very little
   - Yes, but wanted to say more

- Yes, I had my say

3. Did anyone who rarely speaks up share their thoughts?

   - No, they were as quiet as usual
   - Said more than usual, though less than others
   - They were at least equal contributors
   - Not applicable

4. Did the discussion remain focused on what has most impact on people?

   - Went off track
   - Strayed a little but quickly came back on topic
   - Stayed on topic most of the time

5. How was it to have an external facilitator rather than someone from your organisation?

   - Very unhelpful
   - Somewhat unhelpful
   - Made no difference
   - Somewhat helpful
   - Very helpful

6. Did all the cards make sense? If you can recall them, please list any that didn't.

7. What aspects of the procedure would you keep?

8. What aspects of the procedure would you change?

USEFULNESS

Please answer yes or no, and expand on that if you wish. Skip any question if you prefer not to answer it.

1. Would you do the workshop again (e.g., to continue the discussion or talk with a different group of people)?

2. Would you recommend others to try it? For what reason?

3. Did the discussion cover topics which are rarely talked about in depth or can be difficult to raise?

4. Are you considering doing something differently in your work after the discussion today?

5. Do you think any other participants might now consider doing something differently?

6. Could the material be helpful in recruitment (e.g., to learn more about an interviewee)?

7. Could the material be helpful in appraisals (e.g., 360-degree feedback or self-assessment)?

8. Anything else you'd like to add?

Thank you very much for participating!

Appendix O

One month follow-up post-workshop questions

Reflection on the workshop

This survey asks you to reflect on the outcome of workshop. Please elaborate as much as you like on yes/no answers.

1. Have you noticed yourself think or do anything differently since the workshop?

2. Have you noticed other participants do anything differently since the workshop?

3. Have there been informal discussions prompted by the workshop?

4. Would you consider using the workshop format again in future (e.g. to continue the discussion, or with a different group of people)?

5. Would you recommend others to try it?

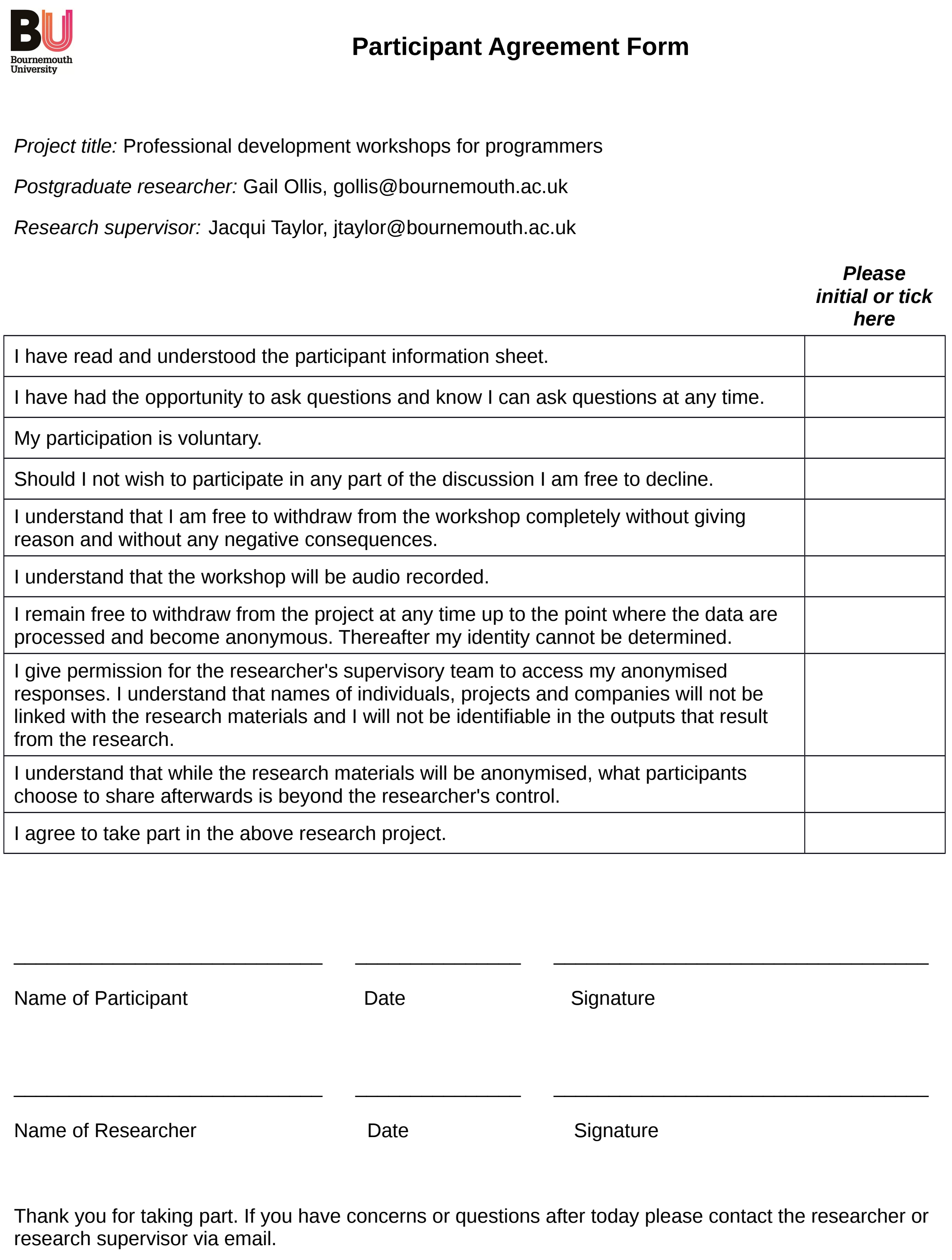
6. Is there anything you would change about the workshop format?

7. Have you had any thoughts about other potential uses of the workshop materials (e.g., recruitment, appraisals)?

8. Anything else you'd like to add?

Appendix P

Workshop consent form



Bournemouth University

Participant Agreement Form

Project title: Professional development workshops for programmers

Postgraduate researcher: Gail Ollis, gollis@bournemouth.ac.uk

Research supervisor: Jacqui Taylor, jtaylor@bournemouth.ac.uk

	Please initial or tick here
I have read and understood the participant information sheet.	
I have had the opportunity to ask questions and know I can ask questions at any time.	
My participation is voluntary.	
Should I not wish to participate in any part of the discussion I am free to decline.	
I understand that I am free to withdraw from the workshop completely without giving reason and without any negative consequences.	
I understand that the workshop will be audio recorded.	
I remain free to withdraw from the project at any time up to the point where the data are processed and become anonymous. Thereafter my identity cannot be determined.	
I give permission for the researcher's supervisory team to access my anonymised responses. I understand that names of individuals, projects and companies will not be linked with the research materials and I will not be identifiable in the outputs that result from the research.	
I understand that while the research materials will be anonymised, what participants choose to share afterwards is beyond the researcher's control.	
I agree to take part in the above research project.	

Name of Participant	Date	Signature
Name of Researcher	Date	Signature

Thank you for taking part. If you have concerns or questions after today please contact the researcher or research supervisor via email.

Figure 35: Workshop consent form

Appendix Q

Workshop card choices

N	Workshops	Freq	Sense	Text
1	A, B	2	0	Anyone who thinks it's "their" code is missing the point.
2	B	1	0	They'll just want to get their own way, rather than do what's best for the product.
3	A, B, C2	3	0	Automating's beautiful.
4	C1, D	2	1	Until you've actually investigated a bug report that someone else has written you don't realise how easy it is to write bad ones.
5	B, C2	2	0	A lot of code looks like nobody went back and re-read it. It's all just stream of consciousness; there's no narrative flow to it.
6		0	3	If someone makes it harder to approach then you're forced to choose less frequent questions.
7	A, C1, D	3	0	Implement the code to do what you're trying to achieve, not try and guess what might be used in the future.
8		0	0	If you think the bug is always in somebody else's code you'll be spending a lot of time looking in the wrong place.
9	A	1	0	It's a pain in the butt if the build's broken and you've got to fix it before you can get on with your real work.
10	A, B, D	3	0	I like small files, small functions, small classes.
11	B	1	0	The simple solution is sometimes the best.
12	A, B, C1, C2	4	0	It's a bit of a quick fix and then you come around to it three times again. And it's "maybe we should have just sorted it out in the first place"
13	A	1	2	Someone not reinventing the wheel? Cor! Who'd have thought it!
14	C1, C2	2	0	Quality of log messages is more important than number of messages.
15		0	1	People get very dogmatic. If you disagree or voice concerns you're a heretic and you're burned at the stake.
16		0	0	People just do what they think is right because they don't want to be arsed talking to someone.
17		0	1	Finding the same block repeated several times makes it a lot harder to understand ...you're trying to mentally diff.
18	C1	1	1	I think you do need to discuss stuff with people.
19	A, C1	2	1	If you have an interface that is natural and obvious then it leaves you freer to get on with your job. It does what it says on the tin.
20		0	1	Identifiers are very, very important.
21	D	1	0	A lot of times I find people are more driven by the latest technologies than by understanding the problem.
Totals		A=8, B=7, C1=6, C2=4, D=4, Sense=11		

Table 17: Frequency of card selections

Appendix R

Immediate post-workshop feedback

The responses to Likert scale questions in the immediate post-workshop questionnaire were as follows.

1. Was it clear what you were supposed to do?	Very unclear	1
	A little unclear	3
	Reasonably clear	7
	Very clear	14
2. Did you get a chance to contribute?	Not at all	0
	Very little	0
	Yes, but wanted to say more	9
	Yes, I had my say	24
3. Did anyone who rarely speaks up share their thoughts?	No, they were as quiet as usual	0
	Said more than usual, though less than others	10
	They were at least equal contributors	8
	Not applicable	6
4. Did the discussion remain focused on what has most impact on people?	Went off track	0
	Strayed a little but quickly came back on topic	10
	Stayed on topic most of the time	14
5. How was it to have an external facilitator rather than someone from your organisation?	Very unhelpful or Somewhat unhelpful	0
	Made no difference	2
	Somewhat helpful	9
	Very helpful	13

Table 18: Frequency of responses to immediate post-workshop questions

Appendix S

How workshop cards might be used in recruitment

The potential use of the workshop cards in recruitment was discussed with an experienced developer who has not been involved in the research. The text below, from a personal communication received in July 2018, is his analysis based on that discussion and the way he has approached interviewing to date.

> “In recent years I have tried to focus more on getting developers to talk in interviews about the issues they run into and how they tackle them. Ultimately I want to know that the way they work is based on conscious actions and not just arbitrarily following ‘best practice’ or just fumbling around until something works. Knowledge is knowing the Singleton pattern creates more problems than it solves, wisdom is knowing why the pattern is dangerous — I want potential candidates to show they have the wisdom as well as the knowledge; or at least wisdom (humility?) in their approach.
>
> While I have managed to comprise various scenarios that help lead candidates towards certain conversations that I personally feel are good indicators, in retrospect it has limitations, most notably that the candidate is not in control and therefore may not be comfortable which is a key aspect early in an interview if you want to get the best out of them. What I believe this technique offers that I had certainly not considered before is choice — giving the candidate the ability to choose their own topics to discuss. My questions are effectively biased towards my own journey, not the candidate’s, so the wrong opening question may immediately put them on the back foot. Recognising this has happened is difficult and tricky to recover from.
>
> Apart from the format — the speech bubbles are a simple, yet effective representation — the other aspect I like is the fact that the statements come from a variety of experienced people. I suspect that letting the candidate know the topics don’t come from a single source will add gravitas to the notion that the interviewers respect a variety of opinions and that there aren’t any wrong answers. The hope is that they will find it easier to open up knowing that (hopefully) they are in good company — the authors of the statements, not necessarily the interviewers.”