# Bumps in the Code: Error Handling during Software Development

**Tamara Lopez**
The Open University

**Helen Sharp**
The Open University

**Marian Petre**
The Open University

**Bashar Nuseibeh**
The Open Univeristy
Lero - The Irish Software Research Centre

***Abstract*—Problems come up during software development all the time. When developers hit these bumps, situations can be surprising and new, and they must figure out what — if anything — has gone wrong. Error handling often resolves small, immediate concerns, however, findings from three ethnographically-informed studies suggest that the way developers experience errors has implications for professional growth and development. Through these experiences, developers refine ideas, strengthen collective practice, and learn.**

## Introduction

How do professional developers fix problems that come up during software development? Research commonly examines how developers learn to program with tools [1] or how they undertake tasks like bugfixing [2] or code maintenance [3]. These studies have established that many issues have at their origin one of several common human errors, including slips-of-action, memory lapses and more complex mistakes in understanding [1]. When questions come up during development, developers have been found to gather information from tools, but struggle to piece together answers from responses the systems give [3]. During bugfixing, developers can think code is wrong that turns out to be correct, a time intensive, costly side effect of the need to seek and process information [2]. These articles examine human error and problem solving within discrete tasks associated with software engineering, but do not reveal as much about the ways in which everyday problems are personally experienced [4] or socially managed [5] in professional environments.

Problems in everyday work are actively experienced — they are bumps that come up during software development (for an overview of terminology, see the "Faults, Bugs and Bumps" box). These issues require developers, like workers in other socio-technical environments [6], to make decisions on the spot. They must solve problems using available tools and existing capabilities, and within the circumstances surrounding their teams, projects and organisations. In these cases, individuals must

 Published by the IEEE Computer Society  

balance the need to write good software alongside their understanding of what is required to finish tasks and keep work moving [7].

This article presents findings from three empirical studies that examinined how developers handle errors. Using qualitative methods to investigate error from the perspective of developers, analytical focus in the studies was redirected from the impact errors have on software in operation, toward factors that influence performance of developers.

### Faults, Bugs and Bumps

Error in software engineering is commonly described using terms like **fault** or **bug**. Faults produce undesirable deviations in software that is in operation. Bugs are a consequence of development activity. They have a presence in code; it is possible to track them down and remove them. Removal improves the dependability of software and reduces the risk of operational failure [8].

Errors are also actively experienced by people [9], with effects that can be felt [10]. In software engineering, these are **bumps** that come up during development. Bumps are handled through **error handling** [11], the problem-solving process people undertake to recover from human errors [12] (see also **Figure 1**). These issues are ephemeral, and are often resolved before software is released.

The next section presents an overview of the methods used to conduct the studies. Following this, a catalog of representative instances of error handling is given. Factors that shaped and influenced error handling among study participants are described. The article closes with three implications for professional growth in developers.

## METHOD

An examination of human errors requires naturalistic data that is observable or that can be reported by the people who experience them [9]. To consider error from the perspective of developers, it is necessary to follow activity forward in time rather than performing deductive analyses to determine the causes of negative events [6]. To meet these aims, a series of three studies were undertaken to examine the actions developers take after problems arise. **Table 1** provides details about data that were collected, including the sites, participants, and data sources.

As in other ethnographically-informed studies [13], investigations were not experimental or hypothesis-driven. Instead, the studies observed practice and accounts of practice in everyday contexts. Collection was unobtrusive: data were gathered from sources that included proprietary and participant-created video, semi-structured interviews and observation. Studies were designed to consider development holistically over time, rather than in terms of discrete engineering tasks that are commonly associated with faults or bugs. The account given here provides a rich narrative of practice, and challenges received views about the nature and role of human error in software engineering.

Using theory drawn from psychology [10] and safety science [6], incidents were isolated in data across the corpus by identifying verbal and visual evidence that participants recognised that something was wrong, and that they subsequently followed a process to remove effects of the problem. Analyses produced a catalog of error handling incidents observed and reported in three contexts: during conceptual design, in work performed at the desk and in reflection after problems were resolved.

In the following section, three types of handling are highlighted. Next, a description is given of factors that were found to influence handling among study participants.

## HOW ERRORS ARE HANDLED

Incidents gathered from each site and context of software development broadly correspond to the error handling process identified in psychology research as depicted in **Figure 1**. To summarise: first, a person realises that a problem has occurred — they hit a bump. Next, the person must identify what was done

**Table 1. Data Corpus**

| Site (Context), Name, Desc | Participants, Role/Edu (Exp) | Sources |
|---|---|---|
| **Site A (Design): The AmberPoint Design Session**<br>Lab-based, set design task undertaken by colleagues followed by a brief interview in which the designers reflected on the session. | - Bill, design professional (10+)<br>- Kasia, design professional (10+) | - 1 video recording, 2.5 hours long<br>- 1 transcription |
| **Site B (Deskwork): Open-Source Acceptance Test Framework**<br>Publicly available video depictions of several months of intermittent development on an open-source project. In-depth analysis performed on a one-month subset of development. | - Joe, agile consultant and engineer (10+)<br>- Marcus, agile consultant and engineer (10+) | - 60 video recordings<br>- 20 transcriptions<br>- Blog posts and website information, social media alerts and photographs, open-source code archive |
| **Site C (Reflection): Research Software Engineering**<br>Semi-structured interviews collected *in-situ* about a challenge in recent work. | - Joachim, Computing,Educational Software (10)<br>- Evan, Computing postgraduate, GIS (5)<br>- Valentin, Computing post- graduate, Web Media, Financial Industries (11)<br>- James, Humanities Computing (20+)<br>- Marisa, Humanities + Postgraduate computing diploma (3.5)<br>- Richard, Humanities Computing (15) | - 7 audio recordings<br>- 6 transcriptions<br>- Field notes taken after site visits<br>- Photographs of work spaces, design diagrams, email exchanges, snippets of code |
| **Site D (Reflection): Higher Education Course Planning**<br>Semi-structured interviews collected *in-situ* with three individuals about a challenge in recent work; 1/2 day team observation. | - Robert, Computer Science/ Software Engi- neering; E-commerce; Aviation (12)<br>- Dereck, Computing & Accounting; Media (6)<br>- Thomas, Degree Unknown, Commercial development (20) | - 4 audio recordings<br>- 3 transcriptions<br>- Field notes taken after site visits.<br>- Drawings, diagrams, email exchanges. |

wrong and what should have been done, a process that has been described within software engineering research in terms of hypothesis formation and modification [2]. Recovery follows when actions are taken that remove the effects of the error [12] and permit the task at hand to continue. Within software development, mitigating actions include making changes to code, configurations, technical environments or other artifacts.

One way to gauge error handling is by how simply the effects of problems can be removed. Slips-of-action are common, and though numerous, are typically easy to detect and straightforward to handle [9]. Other problems are more difficult and require people to actively reason within particular situations and circumstances [10]. Representative examples of handling that demonstrate different degress of simplicity are given in the subsections that follow.

References are used to situate examples within field sites (e.g., Site A), the context in which the example was observed or reported (e.g., design, desk work or reflection). Where appropriate, a pseudonym is given (e.g., Marisa). These details correspond to information in **Table 1**.

### Rules-of-Thumb

Problems that were observed in desk work were, at times, quickly solved using rules-of-thumb [6]. The circumstances in these situations are familiar, making it possible for a person to draw upon a prior experience that matches the current situation so closely, a fix can be applied as a "recipe" or rule.

In cases like this, recovery is straightforward, but ease in handling often only comes after prior instances in which the same error was surprising, disruptive or difficult to solve.

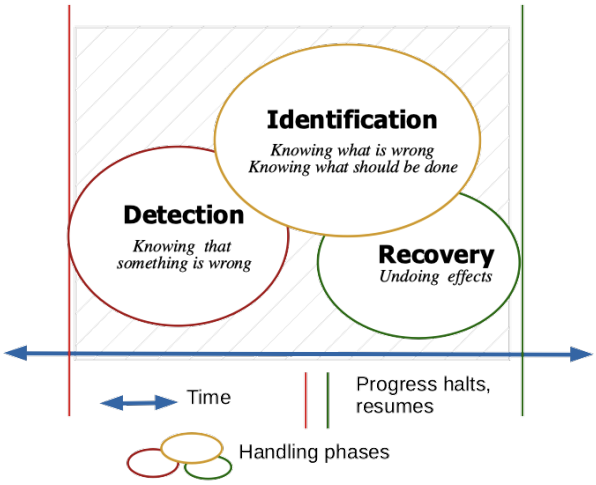


**Figure 1.** Error handling as defined within human error studies in psychology (for an overview, see [10]). Detection indicates that a person realises that something is wrong, identification is the process of knowing what should have been done. Effects are removed in recovery [12].

Rules-of-thumb are not formalised or mandated procedures, but are developed by practitioners. They are cultural and experiential [6], and can reflect preferred ways of working [14], or transmission of know-how between developers:

> "So we have a problem there...that I've noticed happens sometimes. If you actually stop it, now go back to Eclipse and stop it. And then start it again..." (Marcus speaking to Joe, Site B)

### Conscious Command

Other problems that were observed and reported across the sites were novel, "new" to the developers who encountered them. Novel issues command attention [9] and require drawn out, effortful problem solving [10]. Participants at Sites C and D described the identification phase in these cases using terms like "hacking around", "trying things" or "trial-and-error". In desk work, the pair at Site B gave verbal indications that they had focused their attention on a particular error message given by a tool, and were observed in these cases to make multiple changes to a single area of code or piece of configuration.

Within the corpus, many issues that commanded attention also often required multiple cycles of detection, identification, and recovery. Evidence indicated that this cyclical process was immersive, and required developers to manage and remember sequences of actions and outcomes. Valentin described it as "nested problems":

> "You find something, and then you find something else related, you find something related and you are deep in a tree where you [are] never at the end and you must come back." (Site C)

In these instances, handling was also reported to require effort that stretched beyond a single programming session, and to involve local and online social support as depicted in **Figure 2**.

### Getting Lost

Things can go horribly wrong during error handling. In instances collected at Sites B, C and D, handling began in the same way: an error arose that was unexpected, and the developers began a process of identification. However, in these cases, efforts did not remove the effects of problems. Participants indicated that they doubted early actions, forgot where they were within technical environments, or that they had lost control. Robert explained the interplay between action and doubt like this:

> "[Y]ou'll go 'Oh well that had no effect' but you have to be sure... because sometimes when you are dealing with web things, something might have been cached ... maybe you forgot to save the file, maybe your change hasn't been picked up." (Site D)

The most severe incidents in the catalog had simple origins. In one instance, Marcus and Joe lost track of a file in a complicated set of directories (Site B). At Site C, Evan recounted a day in which he had trouble in-

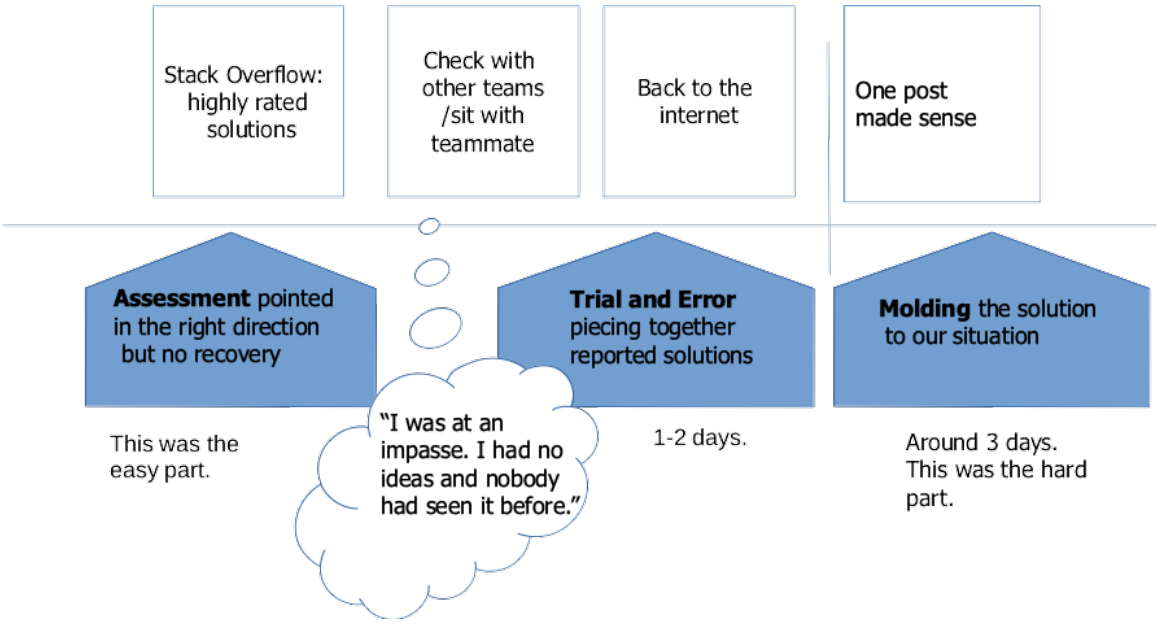


**Figure 2.** Instances of error handling often span multiple days. Within this process, developers seek support from team members and online.

stalling dependencies for an application framework. Dereck described hours spent trying to recover from a slip committed while copying a service to a production server (Site D). In all of these cases, the findings reflect human error research more generally [10]: novices and experts can get lost, and when they do, they exhibit similar, ineffective behavior. As Evan put it:

> "My approach was kind of attrition. The list wasn't appearing, which was annoying and confusing. I'd installed it before without having this issue. I had to try and work out what was happening." (Site C)

## THE SHAPE OF ERROR HANDLING

In contrast to the model presented in **Figure 1**, error handling can be lengthy, cyclical and complex. As Evan's description exemplifies, handling takes a shape that is unique to an individual's experience and bound to the particular situations in which problems arise. It is influenced by a number of qualitative features including emotion and passing time, depicted in **Figure 3** and described in the paragraphs that follow.

In the midst of handling, participants gave indications that feelings helped them expand and constrain activity. Joachim avoided taking responsibility for improper event handling in a webpage with blame, commenting "It must be a memory handling issue in the browser!" (Site C). Dereck initially deflected responsibility for a service he brought down saying "I've checked everything on my side...Maybe they've broken it on their end" (Site D). Feelings also informed decision making in broader terms. For example, Valentin described rejecting alternative solutions on the basis that they were "very ugly", something to be used only as a "last resort" (Site C).

Participants in the design session and at the desk expressed surprise and disappointment when changes did not produce a fix. In a similar way, participants reported feeling stuck in the midst of complex or difficult issues (see also **Figure 2**), that they were "poking around in the dark" (Evan, Site C). When recovery was perceived to be successful, designers and developers from every site gave verbal indications that they were satisfied. However, three other points related to recovery should be noted:

**Recovery is not serendipitous.** Insight was often described by developers at Sites

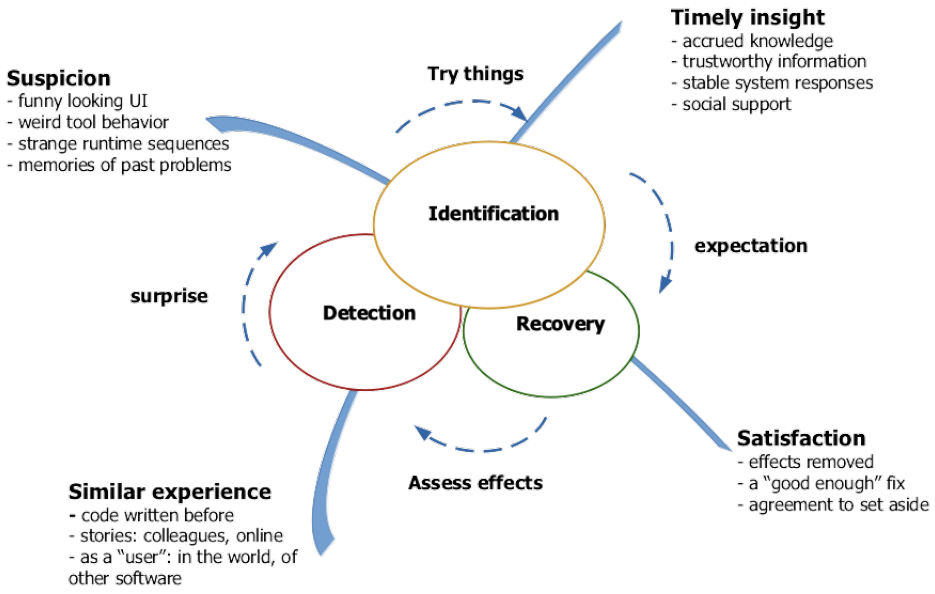


**Figure 3.** Factors that give an individual shape to error handling in software development. Handling is an iterative, cyclical process, moderated by emotion and passing time.

C and D as being sudden or serendipitous. This was not supported in the data. Instead, analysis of interviews and video data from Site B suggest that insight, and by extension recovery from issues is achieved through outcomes of problem solving that are perceived to be timely — the combination at one point of accrued knowledge, information, memory and assesment.

**Recovery does not equal resolution.** In all three contexts, instances were examined that were not resolved. This happened in several cases in which participants got lost, and problem solving was aborted or failed. However, recovery was also deferred. After the fact, Valentin reported that he tolerated multiple manifestations of an error and employed temporary solutions for over a year to gain time for problem solving (Site C). Within the design session between Kasia and Bill (Site A), deferral was evident in verbal exchanges that included the use of fluid terminology, questionning, circling back, and uneven capture in diagrams (for an example, see **Figure 4**).

**Recovery is positive and negative.** Bad feelings linger after handling. Even though things worked out in the end, Dereck felt down on himself for bringing a service down and for leaving it broken overnight (Site D). Evan completed his tasks, but considered the incident to be a personal failure. He was aware that the code he wrote was "pretty dirty" (Site C). However, participants also viewed their experiences positively. Evan reported that his incident helped him understand that "I need to write cleaner code" (Site C). Dereck noted that going forward, "[W]e are just going to re-architect into a single solution" (Site D). After his incident, Joachim released an open source application programming interface (API), remarking "I thought there ought to be a way for others to use this code" (Site C).

## BUMPS IN THE CODE — IT IS THE JOURNEY THAT COUNTS

The path developers take to finished software is bumpy. Though error handling often manages small, immediate concerns, findings in these studies suggest that *the way* developers experience errors has three broader implications for professional growth and development.

**Errors reveal cracks in ideas.** Conceptual issues thread through development practice. Most prominent within the design session (Site A), evidence suggests that similar errors

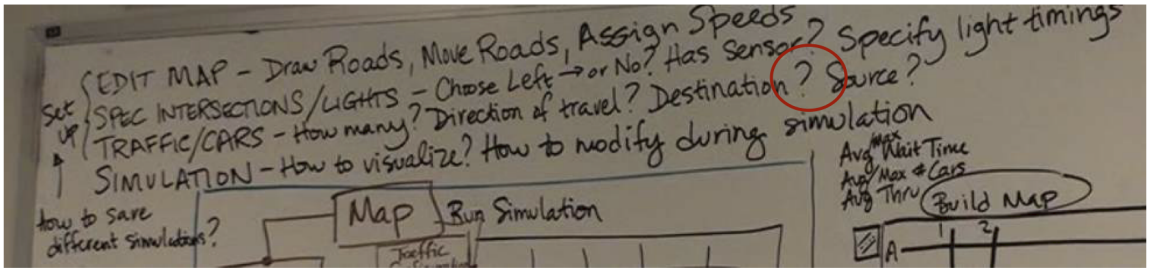


**Figure 4.** Drawn from the design session (Site A), this photo detail shows evidence of incomplete recovery. Unlike other parts of the specification, this area was not diagrammed but comprised a list of questions.

of understanding also occur within coding sessions (Site B) or when tasks are handed between team members (Site D, Thomas). Generally detected through unsatisfactory explanations, examples of "thought errors" [11] were found in every case to be managed through conversation, and were often temporarily set to the side when participants reached verbal consensus.

However, these issues can indicate that a developer does not yet understand what needs to be done in a design or implementation [11]. They are signals that an idea doesn't hold up over time, or transfer from one area of the code to another. This kind of error tends to reoccur, and to be sneaky, apparent to one person, but hidden to another. Marcus and Joe provided an example of this at the desk, in which a piece of wiki syntax looked "okay" to Joe, but was clearly wrong to Marcus. Joe was right, the syntax was correct, but Marcus knew that the markup did not properly reflect a website's information architecture.

**Errors strengthen collective practice.** Just as social activity underpins bug fixing [5], social support underpins error handling. In this corpus, support was reported to come from colleagues, in-house documentation, and commercial and social media sites on the internet. All forms of support were valued. Developers reported at Sites C and D that they relied on the expertise of colleagues, shared information and asked each other for help. However, multiple participants also explained that information shared by developers on the internet, however partial and scattered, made their own work easier and smoother.

Findings also indicate that growth is fostered during error handling through trusting, open connections between colleagues. At the desk, Marcus and Joe committed the same error three times in sessions that spanned a week [14]. The pair were not observed to outwardly penalise each other in any of the instances. Gentle treatment of one another was also evident in a case when a mistaken idea resulted in code that had to be reverted, and in a severe incident in which one of the pair got lost and needed help implementing code using a difficult API.

**Errors give developers space.** Bugs belong to development process, to management, and to users, but errors and error handling belong to developers. Though they place constraints on progress within the development lifecycle, errors give developers freedom to manage tasks. Evan described relying on errors that came up during software installation to direct his process (Site C). In the same way, Marcus and Joe left code in a failed state as a pointer for picking up work at a later time (Site B). Rather than push through to a fix, Kasia and Bill simplified their design task by saving a conceptual problem for "the next version" (Site A).

Opportunity to informally learn [15] is, perhaps, the greatest benefit of error handling. Developers across the contexts displayed an awareness that recovery from difficult problems pushed them beyond the limits of their understanding and experience. Valentin described letting his problem persist as a strategic choice he made to buy time to learn about client priorities and to find the best solution

(Site C). As Robert explained, finding solutions online is relatively easy (Site D). Understanding how to fit them to local conditions is when the hard part really begins.

## CONCLUSION

Developers bump into problems when they stand together at the whiteboard, when they write or make changes to code, when they use libraries and configure application frameworks. Though faults are problematic when software is deployed, the qualitative studies reported here show that human error is an integral part of software development. Through error handling, developers grow as professionals, becoming more capable and competent as they confront and refine their ideas, deepen connections with one another, and learn.

## ACKNOWLEDGMENT

We thank the developers who informed this work. We also thank the reviewers of this paper for their careful reading and generous insights. This research was financially supported in part by the EPSRC (UK), and the Science Foundation Ireland.

## REFERENCES

1. A. Ko and B. Myers, "A framework and methodology for studying the causes of software errors in programming systems," *Journal of Visual Languages & Computing*, vol. 16, no. 1, pp. 41–84, 2005.
2. J. Lawrance, C. Bogart, M. Burnett, R. Bellamy, K. Rector, and S. D. Fleming, "How programmers debug, revisited: An information foraging theory perspective," *Software Engineering, IEEE Transactions on*, vol. 39, no. 2, pp. 197–215, 2013.
3. J. Sillito, G. C. Murphy, and K. De Volder, "Asking and answering questions during a programming change task," *Software Engineering, IEEE Transactions on*, vol. 34, no. 4, pp. 434–451, 2008.
4. M. Eisenstadt, "My hairiest bug war stories," *Communications of the ACM*, vol. 40, no. 4, pp. 30–37, 1997.
5. J. Aranda and G. Venolia, "The secret life of bugs: Going past the errors and omissions in software repositories," in *Proceedings of the 2009 IEEE 31st International Conference on Software Engineering*. IEEE Computer Society, 2009, pp. 298–308.
6. J. Rasmussen, "The role of error in organizing behaviour," *Ergonomics*, vol. 33, no. 10-11, pp. 1185–1199, 1990.
7. D. Piorkowski, S. D. Fleming, C. Scaffidi, M. Burnett, I. Kwan, A. Z. Henley, J. Macbeth, C. Hill, and A. Horvath, "To fix or to learn? how production bias affects developers' information foraging during debugging," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2015, pp. 11–20.
8. A. Avižienis, J.-C. Laprie, and B. Randell, "Dependability and its threats: A taxonomy," in *Building the Information Society*, ser. IFIP International Federation for Information Processing, R. Jacquart, Ed. Springer Boston, 2004, vol. 156, pp. 91–120.
9. D. Norman, "Categorization of action slips." *Psychological review*, vol. 88, no. 1, pp. 1–15, 1981.
10. J. Reason, *Human Error*. New York: Cambridge University Press, 1990.
11. D. A. Hofmann and M. Frese, *Errors in organizations*. Routledge, 2011.
12. A. J. Sellen, "Detection of everyday errors," *Applied Psychology*, vol. 43, no. 4, pp. 475–498, 1994.
13. H. Robinson, J. Segal, and H. Sharp, "Ethnographically-informed empirical studies of software practice," *Information and Software Technology*, vol. 49, no. 6, pp. 540 – 551, 2007, qualitative Software Engineering Research.
14. T. Lopez, M. Petre, and B. Nuseibeh, "Examining active error in software development," in *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2016, pp. 152–156.
15. M. Eraut, "Informal learning in the workplace," *Studies in continuing education*, vol. 26, no. 2, pp. 247–273, 2004.

**Tamara Lopez** is a post-doctoral research associate at the Open University. Her research examines human factors of professional software development including error, security and resilience. Lopez completed her PhD at the Open University in 2016. Prior to this, she worked as a research software engineer in the digital humanities at King's College London and Indiana University. She can be contacted at: tamara.lopez@open.ac.uk

**Helen Sharp** is Professor of Software Engineering at the Open University, UK. Her research investigates professional software practice with a focus on human and social aspects of software development. Sharp studied at University College London as an undergraduate in Mathematics and as a postgraduate in Computer Science. She is a chartered engineer and serves on the Advisory Board for IEEE Software. She can be

contacted at helen.sharp@open.ac.uk

**Marian Petre** is a professor of computing at The Open University, UK, and a visiting distinguished professor at University of California, Irvine, US. Her research focuses on empirical studies of expert software developers. Petre received her PhD in computer science from UCL. She is an Associate Editor of IEEE Software. Contact her at m.petre@open.ac.uk.

**Bashar Nuseibeh** is Professor of Computing at the OU, a Professor of Software Engineering at Lero – The Irish Software Research Centre, and a Visiting Professor at UCL and the National Institute of Informatics (NII), Japan,. His research is multi-disciplinary, underpinned by an interest in software and requirements engineering, security and privacy, and adaptive systems. He holds a Royal Society-Wolfson Merit Award. He can be contacted at: bashar.nuseibeh@open.ac.uk