Getting at Ephemeral Flaws

Tamara Lopez*, Marian Petre*, Bashar Nuseibeh*,†
*Centre for Research in Computing, The Open University, United Kingdom
† Lero The Irish Software Engineering Research Centre, Ireland
{t.lopez, m.petre, b.nuseibeh}@open.ac.uk

***Abstract*—Software rarely works as intended when it is initially written. Things go wrong, and developers are commonly understood to form theories and strategies to deal with them. Much of this knowledge relates to ephemeral flaws rather than reported bugs, and is not captured in the software record. As a result, these flaws and understanding about them are neglected in software engineering research. In this paper we describe a study designed to elicit stories from software developers about problems they encounter in their daily work. We also offer preliminary thoughts about the utility of retrospective interviewing in getting at information about ephemeral flaws.**

***Keywords*-software dependability; software development; empirical studies**

I. Introduction

In his paper *Dependability: A Unifying Concept*, Randell noted that clarifying the concepts related to failure in software engineering research is difficult. System boundaries are fluid and artefacts are complex. Judgements about what causes failure are subject to perception and attitudes, and the mechanisms designed to prevent failures are themselves failure-prone[1].

Given this, the concept of error is surprisingly stable in the literature. It is described using different terms like *fault*, *defect* or *bug*. However it is consistently taken to mean those elements of software as written which threaten or produce undesirable, unexpected and unintended deviation in behavior[2]. Root-cause analysis is one area of software engineering that has used this meaning to establish a research model for analyzing the sources of errors in software.

Early on, authors of root-cause studies acknowledged that the definition of error to which they worked was simplified[3]. This compromise was seen as necessary to produce measurable process improvement. In the intervening years, however, researchers have struggled to adequately or satisfactorily answer the question of why some errors occur, limited by a definition of error that cannot account for qualitative factors.

Our research reconsiders what errors in software are, developing a model that is human and historical. We acknowledge that things that go wrong may result at some point in faults within source-code[2], and thus leave evidence within the software record. However our model accommodates the fact that errors may also be ephemeral. They may exist only as misunderstandings or miscommunications. They may go wrong and be put right again before a release, commit, or save. Thus, they may leave no clear (or indeed any) representation within code, descriptions or project records. The context and meaning associated with these kinds of flaws are lost over time.

In the rest of this paper, we describe a study that explores one method for researching this expanded notion of error.

II. Why Bother and Where to Look

If they are put right or forgotten, why do things that go wrong matter? A great deal of attention has been paid in the past to finding ways to reduce the number of errors in software. However there has not been corresponding attention paid to existing practices that help to avoid their accumulation in the first place. Similarly, though understanding and coordination are studied in the context of bug fixing[4], the way they unfold at other points when things go wrong is for the most part neglected[1]. As a result, software engineering research misses opportunities to make software "better"[6], and to help developers do better work.

The root-cause analyses examined make two suggestions for additional research that support this view. The first is that data about errors should be collected from the entire development cycle, not just during testing and integration[3][6], and should not be collected too long an interval of time after events have passed[7]. The second is that studies should be made that examine the causes of "human erring"[3, p.331], including factors such as problems of understanding[3], inexperience[8], lack of information[7], and skill mismatch[9].

Unfortunately, though these papers offer clear ideas about what to examine, they do not offer many suggestions for how to go about doing it. They do, however, suggest likely challenges. To get around the fact that time erodes knowledge about errors, Perry suggests that programmers be asked to classify their errors as a part of closing modification and bug reports [6], a technique found not to work particularly well by Leszak et al.[9]. Organizational access is noted to be difficult to attain when it requires sharing information about mistakes[6], and management can seriously constrain study design, in extreme cases resulting in retrospectively gathered, anonymous self-reports[7].

[1]For a recent exception, see [5].

We used these points to formulate several premises for a study designed to get a first look at ephemeral flaws. First, we assumed that self-reports are not necessarily unreliable, particularly if developers are permitted to recount experiences in their own words[10]. Second, we acknowledged the difficulty of gaining sufficient access to allow for detailed observation of the origins, occurrences and outcomes of things going wrong and compromised by seeking reports about recent work. Third, in contrast to the root-cause analyses, we assumed that bug reports are erroneous and incomplete[4] in representing how work happens, and that an understanding of error must look beyond these records.

A. The Critical Decision Method

With these assumptions in mind, the critical decision method was selected for use, as described in *Working Minds: A Practitioner's Guide to Cognitive Task Analysis*[11]. Applied cognitive task analysis is intensive, requiring a good understanding of cognition theory, and it is unclear at this point to what extent the larger study of ephemeral flaws will be able to apply it. Nevertheless, the guidelines provided for conducting critical decision method interviews in this text have served as a good introduction to eliciting retrospective accounts of work.

The critical decision method was designed to understand how people think in natural settings. It was developed to address the fact that how people think in the workplace is not well explained by the findings of experimental studies of cognition. To this end, the critical decision method and other techniques can be used to examine the functions associated with an expanded range of cognitive phenomena that includes sensemaking, planning, coordination, adaptation, and naturalistic decision making. In addition to illuminating how people think on the job, the larger framework of cognitive task analysis assists researchers in understanding expertise in individual domains, by revealing the differences between how experts and novices approach and manage their work.

III. Method

For this study, seven individuals were interviewed over the course of four weeks. Participants perform a range of software development tasks in an established U.K. digital humanities center, described in 2008 by the Council on Library and Information Resources as an environment "where new media and technologies are used for humanities-based research, teaching, and intellectual engagement and experimentation. The goals of the center are to further humanities scholarship, create new forms of knowledge, and explore technology's impact on humanities-based disciplines."[12]

Development roles within this center tend to be fluid, with employees responsible for multiple projects at a single time, in a variety of capacities. In this study, all participants were interviewed about an incident in which they played a discrete role, in audio-recorded sessions that lasted from between forty-five to seventy-five minutes.

Interviews were conducted by a single person, but otherwise followed the basic procedures for conducting a critical decision method interview. These entail examining a single incident in four semi-structured "sweeps". In the first sweep, the participant and the researcher identified an incident, broadly defined as one having taken place in the previous two weeks and in which the participant was a key decision maker. In the second, a timeline was established to note critical decision points. In the third, deepening probes were used to develop comprehensive and detailed understanding about the incident. Though researchers often selectively use probes at this stage to examine one or two cognitive phenomena, this study made opportunistic use of a range of probes, with an aim to identify in analysis those which are most effective for learning about ephemeral flaws. Finally, the participant was asked to consider hypothetical alternatives to decisions taken.

Each interview concluded with questions about the participant's educational and professional background. Participants were asked to give the researcher copies of artefacts mentioned in the discussion.

A. Validity

Given its exploratory aims and its focus on a single organization and method of data collection, the results of this study alone cannot make strong claims of validity. In addition, it must be stated that the primary researcher had prior understanding about the organizational culture in which the participants work. However, this researcher had no direct knowledge of any of the projects discussed.

IV. Discussion

Analysis is in very early stages, and it is not possible to report any conclusive findings. However, the interviews did yield initial insights which resonate with findings in the root-cause analyses. The stories collected suggest that the "Why?" of some errors does, as Endres suggested, operate on several levels, only some of them technical. They also suggest that even bugs that are not critical, costly or difficult do indeed possess "rich" histories[4] that can be investigated beyond the immediate details of the coordination and communication efforts required to fix them.

A. "Secret" bugs cannot be counted

In perhaps the most straightforward bit of evidence gathered, one developer recounted the process of refactoring a piece of software to consolidate and abstract functionality. In the process of locating related code, he found and fixed at least one small bug that had never been reported, and which was not causing any known problems. This issue of secret bugs was noted by Endres[3], and is accounted for by Avižienis et al.[2] , but little evidence is documented in the

literature about what kinds of errors are found this way, and what relation if any they bear to the larger task at hand at the moment of their discovery.

B. Some causes do not make sense

In one case, a developer chose to discuss fixing a recently reported bug. The fix was trivial, involving altering basic conditional behavior in a single function. A root-cause analysis might have interpreted this as a novice programming error, or more generously as one of those things that "just happens". However the developer's training and years of experience suggested that the first explanation did not make sense. The full story suggests that this bug also did not just happen, but instead may have had origins several months earlier, shaped by decisions related to open-source technology selection, the introduction of new communication channels, and design choices taken to meet domain-specific requirements.

C. Not every bug is a mistake

In another instance, an issue identified by a developer surfaced as a bug several times over the course of nearly two years, in open-source tools used by the developer and in different areas of the software being developed. The issue was related to the use of Unicode, which presented particular complexities when introduced to the domain. Though the programmer was well-versed in the technology, a solution was not immediately apparent. However, rather than merely signalling lack of knowledge as the root cause, this story suggests that the developer actively managed the issue over time, implementing incremental pragmatic solutions as required to advance the larger program of work. This strategy allowed him to explore the problem over time, and ultimately to find the "best" solution.

V. Conclusion

Ephemeral flaws are neglected in software engineering research. Getting at them is difficult, in part because of their nature: they are fleeting, not well documented, and poorly accounted for by established models of software error.

The study reported here suggests a promising way to develop understanding about how errors in software are shaped by human effort and time. Retrospective interviewing permits rich access to the people who make software, but the data collected remains temporally removed from development as it happens. Thus, the stories have the potential to both benefit from hindsight and suffer the passage of time. In addition, the basic method employed in this study resulted in accounts of small, quickly resolved issues and others of persistent issues that stretched across many months. This method is being refined for future interviews to target particular timeslices and activities. Other techniques are being investigated to allow us to get at ephemeral flaws as they are being experienced.

Acknowledgment

We thank the software developers who gave their time to our study, and Michael Jackson for his insightful review of this paper. This research is supported in part by SFI grant 10/CE/I1855 (Nuseibeh) and by a Royal Society Wolfson Research Merit Award (Petre).

References

[1] B. Randell, "Dependability-A unifying concept," in *Proceedings of the Conference on Computer Security, Dependability, and Assurance: From Needs to Solutions*. IEEE Computer Society Washington, DC, USA, 1998.

[2] A. Avižienis, J. Laprie, and B. Randell, "Dependability and its threats: a taxonomy," in *Building the information society: IFIP 18th World Computer Congress: Topical sessions 22-27 August 2004, Toulouse, France*. Kluwer Academic Pub, 2004, pp. 91–120.

[3] A. Endres, "An analysis of errors and their causes in system programs," in *Proceedings of the international conference on Reliable software*. ACM, 1975, pp. 327–336.

[4] 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.

[5] A. Ko and P. Chilana, "Design, discussion, and dissent in open bug reports," in *Proceedings of the 2011 iConference*. ACM, 2011, pp. 106–113.

[6] D. Perry, "Where do most software flaws come from?" *Making Software: What Really Works, and Why We Believe It*, pp. 453–494, 2010.

[7] D. Perry and C. Stieg, "Software faults in evolving a large, real-time system: a case study," *Software EngineeringESEC'93*, pp. 48–67, 1993.

[8] D. E. Perry and W. M. Evangelist, "An empirical study of software interface faults — an update," in *Proceedings of the Twentieth Annual Hawaii International Conference on Systems Sciences*, vol. Volume II, January 1987, pp. 113–126.

[9] M. Leszak, D. Perry, and D. Stoll, "Classification and evaluation of defects in a project retrospective," *The Journal of Systems & Software*, vol. 61, no. 3, pp. 173–187, 2002.

[10] M. Eisenstadt, "My hairiest bug war stories," *Commun. ACM*, vol. 40, no. 4, pp. 30–37, 1997.

[11] B. Crandall, G. Klein, and R. Hoffman, *Working minds: A practitioner's guide to cognitive task analysis*. The MIT Press, 2006.

[12] D. Zorich, *A survey of digital humanities centers in the United States*. Council on Library and Information Resources, 2008.