

# Understanding Context: Creating a Lasting Impact in Experimental Software Engineering Research

Emerson Murphy-Hill  
North Carolina State  
University  
emerson@csc.ncsu.edu

Gail C. Murphy  
University of British  
Columbia  
murphy@cs.ubc.ca

William G. Griswold  
University of California  
San Diego  
wgg@ucsd.edu

## ABSTRACT

Software is developed for and in a vast number of contexts. Some software systems are small in size; some large. Some systems are developed by small teams; some large. Some projects are sensitive to schedule, others to safety of the users. In this position paper, we argue that to make a lasting impact with the software engineering research we conduct, we must understand, make explicit, and vary the context in which our conclusions are drawn. Moreover, we need a better understanding of how research results can be translated or generalized to other contexts, as it is not economically feasible to replicate results across all contexts. We argue that a successful solution to this problem will allow researchers to conduct research within particular contexts, richly characterize those contexts in their writings, and allow other researchers to predictably build on those in differing contexts.

## Categories and Subject Descriptors

D.2.0 [Software Engineering]: General

## General Terms

Experimentation, Measurement

## Keywords

Software Engineering Research, Methodology

## 1. THE PROBLEM

Researchers often perform studies to demonstrate that the concepts and tools that they propose will have a significant impact on the practice of software engineering. The goal is that the study should give the community confidence that the concepts and tools will have a significant positive effect on how software is developed, both in the short-term and in the long-term. However, this is difficult to achieve for

many reasons, one of which is that it is difficult to perform an empirical study that is convincing.

For example, consider Müller and Tichy's study of extreme programming [8], which investigates the successes and challenges of the extreme programming software process. One of their findings was that pair programming is difficult to implement without coaching. If this result is generally true for groups of software developers other than the ones investigated during this study, then this research finding is likely to have a high research impact, because software teams should hire a coach when adopting pair programming. But what confidence do we have that that their result really is generally true? To answer this question, we have to understand a bit about *context*.

By context, we mean the setting in which software engineering is practiced. Examples include the working styles of software developers (for example, hypothesis-driven or cause/effect driven), the values held by a development team (coding heroics or team builders), the cultural background of the developers (American or Chinese), the paradigm of the code (object-oriented or procedural), and the kind of industry for which the software is being developed (entertainment or health care). Context, then, is a multi-dimensional space with an infinite number of points, each point defining a particular software project at a particular time.

So to answer the question about what confidence we can have about Müller and Tichy's results, consider three dimensions of context in which the study took place:

- The study was conducted with students as software developers rather than professional software developers. This participant group reduces our confidence that the results generally hold because most software developers are professionals, not students.
- The participants implemented software over about eight hours. This duration reduces our confidence that the results generally hold because most software projects take place over more than eight hours.
- The participants completed small programming exercises. This task set reduces our confidence that the results generally hold because most deployed software is not developed as part of an exercise.

These issues illustrate the difficulty in performing convincing studies (as well as the ease with which a reader can criticize them).

The flippant answer to this problem is that Müller and Tichy did the wrong study, that they should have instead

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

FoSER 2010, November 7–8, 2010, Santa Fe, New Mexico, USA.  
Copyright 2010 ACM 978-1-4503-0427-6/10/11 ...\$10.00.

done their study with professional software developers rather than students, should have conducted the study over a period of years, and should have had participants complete real software projects rather than exercises. While these suggestions might have been feasible, the study that the researchers performed was nonetheless appropriate because at the time the study was conducted; the research community knew little about the then-nascent practice of pair programming. At the time, this more limited, cost-effective study was warranted to postulate and provide some preliminary evidence for the efficacy of pair programming. Afterwards, follow-up studies in a broader context – such as with professional developers – could evaluate and refine Müller and Tichy’s findings. Yet, as a community, we often demand that researchers perform studies in “more realistic” contexts without considering whether those contexts are warranted.

This example suggests that the issue of context is subtle and can easily be overlooked in the quest for better software engineering research. However, we argue that software engineering research should pay more careful attention to the context in which software engineering is performed, to increase the impact of the research and the improve our ability to build upon that research.

## 2. CALL TO ACTION

Here we present five ways that we can pay more attention to the context in which software engineering is practiced. Our suggestions are based on our own personal experiences as researchers and reviewers. We state our suggestions as calls to action for the researcher, but these suggestions should also be considered by reviewers when weighing the merits of a research result.

### 2.1 Building Foundational Results

Our first suggestion is that the community should expend more effort on exposing the foundations of software engineering. By foundations, we mean results that are likely to apply in both present and future software engineering contexts.

For example, much research has been done on tools that find design defects in software (for example, [3], [9], and [10]). While these tools may perform well in, for example, contexts where object-oriented programming is the dominant paradigm, it is difficult to judge from this research whether the tools will apply equally well in other contexts. On the other hand, research from Mäntylä and Lassenius has shown that work history affects how developers perceive design defects and that individual developers sometimes have conflicting opinions of design defects [7]. These results are more foundational in that, rather than reporting on the effect of specific tools, they characterize the way software developers work and think.

For researchers to build foundational results, they should consider how the context that they study will exist in the future. For example, when picking a code base to analyze, the researcher might choose a web application rather than a desktop application, arguing that the former will become relatively more prevalent as time goes on.

### 2.2 Establishing Causes and Effects

In the quest for results that have strong external validity, many software engineering studies attempt to make their study contexts as realistic as possible. While the benefits

of realism are clear, the drawbacks are not. One drawback in doing studies in realistic contexts is that such contexts make it difficult for the researcher to establish causes and effects. This difficulty arises because in realistic contexts, many possible causes may affect a result, making it difficult to distinguish one cause from another. Since establishing causes and effects are critical components to building foundational results, it follows that limit or controlled contexts could yield more impactful results.

For example, consider Green and Petre’s study, which compared textual programs against visual programs in terms of comprehensibility [4]. The authors could have opted for a high degree of external validity by having programmers work with textual and visual programs in their respective development environments to complete a realistic task. Instead, the researchers asked programmers to compare static screen images of visual and textual programs. By conducting their study in this way, the results provide strong evidence that the representation of the programs were the causes of programmers’ superior understanding of textual programs over visual programs. This result can be further investigated and built upon, which makes this research a foundation for future research.

For researchers to establish causes and effects, they should consider how likely causes and effects can be established, given a choice of contexts. For example, suppose that you are conducting a study about the attitudes of managers towards software design tools. When deciding whether to observe software managers during a week when they are facing an immediate deadline versus a deadline further into the future, you might choose to study the latter, because managers may be more focused and reflective during the study’s interviews.

### 2.3 Triangulating Results

Most researchers recognize that replication is a good thing, because it can instill confidence. However, two studies that reach the same conclusions in the same context give us less confidence than two studies that use different contexts. This is because, when different contexts are used, confirmatory results demonstrate the robustness of the result. For example, if you did one study with a small team and another with a large team, and both studies had the similar findings, we would have reason to believe that the findings will hold across team sizes.

For example, Briand and colleagues replicated a previous study which explored relationships between design measures – such as that frequency of method invocations predict fault-proneness – in the context of student projects [1]. In their replication, the authors showed that some of the relationships held in another context by studying software developed by professionals. Through their replication, the authors showed how these relationships hold across developers’ experience levels.

For researchers to triangulate results when replicating previous studies, they should consider what aspects of the research context can be varied. For example, rather than performing a *literal replication*, where the researcher designs a study to achieve similar results as a previous study, the researcher could instead perform a *theoretical replication*, where the study context is varied so that opposite results are expected [11]. For example, suppose we are interested in replicating Dehnadi and Bornat’s study which suggested

that a simple test predicts whether or not students will succeed in an introductory programming class [2]. The theory is that those students who pass the test have an innate ability to program. Rather than performing a literal replication, we could instead vary the context and perform a theoretical replication; rather than giving the test to a group of students who are about to take a programming class, give the same test instead to a group of students who are about to take a history class. In this situation, we propose an alternative theory, that the test simply predicts general academic ability. If this alternative theory is correct, then the results of the test should predict how the students perform on the history test.

## 2.4 Clarifying the Context

Simply explaining the context in which a study occurs goes a long way towards creating impactful research. This is both because a practitioner who wants to decide whether your research applies to her can compare her software engineering context to the one you describe in your study, and also because future researchers can make intelligent choices about how to vary the context in future replicated studies.

As an example of research that clarifies the context of a study, consider “An Ethnographic Study of Copy and Paste Programming Practices in OOPL” by Kim and colleagues [5]. This paper, which describes a study that results in a taxonomy of copy-and-paste usage patterns, clearly explains that the results were obtained in the context of object-oriented programs in small research software.

For researchers to clarify the context of their studies, they should be explicit about the context when writing up the study. Often, due to space limitations, this means making auxiliary material available by means of a URL or a technical report. Clarifying the context also means not trying to rationalize away the risks in a study, but accepting them as they are and encouraging other researchers to exploit them in future studies. In the long term, we speculate that a more common standard for how to present and discuss the context – a vocabulary of research patterns – might help researchers to more thoroughly understand the context in which a study was conducted and built upon it in a reliable way.

## 2.5 Accepting Limited Contexts

The investigation of some phenomena are so intensive that one either must be willing to accept a limited context, or must forgo the study altogether. While such studies are normal in other fields, software engineering is rarely accepting of limited contexts – and this, we argue, is a mistake.

As an example from another domain, consider Kurtenbach and Buxton’s study of how users become experts at using marking menus, a type of menu where items appear around a circle and the user gestures to select an item [6]. Because the authors were interested in the effects of learning over time in a realistic context, they performed their study with only two users. This limited context allowed them to more thoroughly analyze the results in detail.

For researchers to accept limited contexts, they should consider whether the research question that they are trying to answer can be better answered in depth by using a limited context, rather than more shallowly in a broader context.

## 3. CONCLUSIONS

In this position paper, we have argued that software engineering research is sometimes difficult to evaluate in a convincing way. If the community is going to surmount this difficulty, we argue that we must pay special attention to the context in which we evaluate. To do so, we believe that future software engineering research should build a practice of stating, characterizing, and using context intelligently. If we ignore context, our research contributions will languish in isolation and obscurity, but if we embrace it, our research will flourish and have a significant impact.

## 4. ACKNOWLEDGEMENTS

Thanks to Thomas Fritz and our anonymous reviewers for providing comments on early versions of this paper.

## 5. REFERENCES

- [1] L. C. Briand, J. Wüst, and H. Lounis. Replicated case studies for investigating quality factors in object-oriented designs. *Empirical Software Engineering*, 6(1):11–58, 2001.
- [2] S. Dehnadi and R. Bornat. The camel has two humps. Unpublished, 2006.
- [3] E. v. Emden and L. Moonen. Java quality assurance by detecting code smells. In *Proceedings of the Ninth Working Conference on Reverse Engineering*, pages 97–106, Washington, DC, USA, 2002. IEEE Computer Society.
- [4] T. R. G. Green and M. Petre. When visual programs are harder to read than textual programs. In *6th European Conference on Cognitive Ergonomics*, pages 167–180, 1992.
- [5] M. Kim, L. Bergman, T. Lau, and D. Notkin. An ethnographic study of copy and paste programming practices in oopl. In *ISESE '04: Proceedings of the 2004 International Symposium on Empirical Software Engineering*, pages 83–92, Washington, DC, USA, 2004. IEEE Computer Society.
- [6] G. Kurtenbach and W. Buxton. User learning and performance with marking menus. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 258–264, New York, NY, USA, 1994. ACM.
- [7] M. V. Mäntylä and C. Lassenius. Drivers for software refactoring decisions. In *ISESE '06: Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering*, pages 297–306, New York, 2006. ACM.
- [8] M. M. Müller and W. F. Tichy. Case study: Extreme programming in a university environment. In *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, pages 537–544, Los Alamitos, CA, USA, 2001. IEEE Computer Society.
- [9] C. Parnin and C. Görg. Lightweight visualizations for inspecting code smells. In *Proceedings of the 2006 ACM Symposium on Software Visualization*, pages 171–172, New York, 2006. ACM.
- [10] R. Wetzel and M. Lanza. Visually localizing design problems with disharmony maps. In R. Koschke, C. D. Hundhausen, and A. Telea, editors, *SOFTVIS*, pages 155–164. ACM, 2008.

- [11] R. K. Yin. *Case Study Research : Design and Methods*. SAGE Publications, 1986.