The Future of Software Engineering: Enhancing Human Expertise in Tackling Software Quality

Margaret Burnett
Oregon State University
School of EECS
Corvallis, Oregon 97331 USA
burnett@eecs.oregonstate.edu

ABSTRACT
This paper argues that the essential point of software engineering is not to (semi-automatically) improve the quality of software, but rather to help human problem solvers to improve the quality of software. This paper argues that this essential point of software engineering research has been forgotten, and that without regaining that perspective, significant progress is not likely.

Categories and Subject Descriptors
D.2.0 [**Software Engineering**] General

General Terms: Reliability, Human Factors.

Keywords: Software Engineering

1. INTRODUCTION
As the challenges of software quality grow, with huge software the size of cities, a diverse population of developers, and increasing malicious activity, the future of software engineering lies with human problem solvers. Unfortunately, this point is being largely ignored in software engineering research. This paper argues that ignoring the fact that the point of software engineering research is to *enhance* human problem-solving—not to *replace* it—obscures our vision, thereby imprisoning much of software engineering research in myopic activities.

First, why am I arguing that human problem solving is being ignored? After all, we can point to a growing empirical studies presence in software engineering whose focus is on how humans behave with software engineering problems. Further, these works are being published in mainstream software engineering venues in increasing quantities.

The problem is, these studies tend to be too myopic for long-term vision, because they necessarily deal with low-level questions. For example, formative empirical research looks at questions like how developers refactor or to what extent they navigate when performing maintenance. Using these results, researchers then build tools to solve these low-level needs. Summative empirical work then investigates how well the tools helped, in terms of outcome measures such as time spent or bugs fixed. These are all valuable, but they do not usually promote *big* steps forward. Specifically, they do not (and usually cannot) consider whether the studied approach enhances human problem solving ability with software engineering problems in a general way, so that when a very difficult problem comes along, a human-tool partnership is available that has been honed to handle *those* kinds of problems. Addressing this kind of question cannot be done solely by empirical work—it requires the involvement of human-oriented theories from the outset.

Second, why do I think human problem solvers are so important? The current trend is instead to continually try to make the tools (rather than the humans) smarter. The implied motivation for this approach is that eventually, tools will become smart enough to replace the need for human problem solving entirely—an argument that I think none of us actually believes. For example, it seems unlikely that software engineering tools[1] would be able to cope without human intelligence when malicious or unprecedented events cause a tool's knowledge base to have errors, cause requirements to change suddenly, or require different inferences on similar data than were required in the past.

2. A VISION OF HUMAN/TOOL PARTNERSHIPS IN SOFTWARE ENGINEERING
What is needed is a true partnership between humans and software engineering tools. In such a partnership, the software engineering tool is regarded as a human intelligence *enhancer*, not a replacement. Further, when the tool goes awry, the human is positioned not to be a slave to the tool, but rather to lead a partnership in which the tool contributes, but each partner knows how much to trust the other, and when to step in and argue.

To accomplish such a vision, at least four steps must be taken: (1) to develop or borrow *theories of human problem-solving* as they apply to software engineering, (2) to employ under-the-hood software engineering reasoning approaches around what is needed to *advance*

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, or 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.

[1] In this paper, I will use the term "software engineering tools" to denote embodiments of software engineering research. For example, if a new formal reasoning approach is developed, it often will also be implemented in tools for purposes of further investigation or dissemination.

human effectiveness at solving software engineering problems, (3) to introduce *transparency* into software engineering tools, so that humans have the information and intuitions necessary to know where and how to step in, and (4) to take into account *all* software engineering populations, not just the ones who are "like me".

Regarding the first two steps, developing or borrowing theories of human problem-solving in software engineering and then basing tools on those theories, an obvious objection arises: most researchers in software engineering are not trained in social sciences. This objection may be one reason progress in this area has been so slow, but it is not a good argument against the *need* for this step. Avenues for addressing it include multidisciplinary collaborations with social scientists or HCI researchers, or borrowing existing theories from other disciplines and systematically investigating their applicability to software engineering. Examples of both approaches have been emerging in recent literature, such as information foraging work applied to debugging [4], the WhyLine for debugging [3], and an emerging working-set notion of supporting humans doing software development [1]. A key attribute of the work I am envisioning is that it aims for tools to be designed from the outset around thoroughly researched principles of human problem-solving needs, not around "what is something we can automate".

The third step involves transparency. By this I mean that software engineering tools would communicate the reasoning they were conducting, including *how* they were conducting it. Without transparency, humans are in a poor position to judge tools' outcomes or reason about which aspects might be wrong. Further, without transparency, human abilities in reasoning can atrophy. For example, how many of us have run across cashiers who are unable to make change if their cash register's calculation device malfunctions, or students unable to read cursive handwriting because they rarely use it? Transparency can help to balance the fine line between a tool contributing to a human's reasoning versus supplanting it entirely for better or worse.

Unfortunately, transparency is not necessarily easy to achieve. Many software engineering tools are based on complex algorithms that are difficult to explain to humans. Cases in point include some of the more sophisticated machine learning algorithms used to inform software engineering reasoning tools. Finding ways to achieve transparency to enhance human reasoning power, given sophisticated software engineering reasoning tools and huge amounts of data, is a challenging problem.

Finally, we must take into account the problem-solving needs of *all* populations of developers, not just the ones closely resembling the predominantly white, well-educated, male, software engineering research population of today. For example, research has already shown that female software developers do not necessarily approach software development problems the same way males do [2]. Further, the amount of software development done by end users greatly exceeds the amount of software development done by professional developers [5]. Without taking into account the human problem-solving needs of these diverse populations of software developers, the chances of software engineering research making real-world impact seem slim.

3. SUMMARY

The vision advocated in this paper is a human/tool partnership—one that is able to cope as a team even in the face of unexpected events, suddenly changing requirements, malicious activity, or erroneous information in a tool's knowledge base.

To achieve this vision will require promoting the needs of human problem solving to become first principles in software engineering research, which will need to be achieved by drawing upon theories that describe what these human needs are. Achieving this vision will also require addressing the issue of transparency, and taking into account the diversity of reasoning approaches by the human software developers expected to benefit from the research.

Moving toward this vision entails solving difficult challenges, and further will require both a culture and a methodological shift in the software engineering research community. Despite the challenges, I believe such a shift is necessary, because without it, the human developers intended to ultimately benefit from our research will be in a weak position to face the difficult software engineering challenges ahead.

4. ACKNOWLEDGMENTS

This work has been supported in part by NSF 0803487 and 0917366, AFOSR FA9550-09-1-0213, and an IBM OCR grant.

5. REFERENCES

[1] Bragdon, A., Zeleznik, R., Reiss, S., Karumuri, S., Cheung, W., Kaplan, J., Coleman, C., Adeputra, F., and LaViola Jr., J. 2010. Code Bubbles: A working set-based interface for code working set-based interface for code understanding and maintenance, In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, (Atlanta, GA, April 10-15, 2010.) CHI'10, ACM, New York, NY, 2503-2512. DOI=http://doi.acm.org/10.1145/1753326.1753706.

[2] Burnett, M., Fleming, S., Iqbal, S., Venolia, G., Rajaram, V., Farooq, U., Grigoreanu, V. and Czerwinski, M. 2010. Gender differences and programming environments: Across programming populations, In *Proceedings of the IEEE International Symposium on Empirical Software Engineering and Measurement*, (Bolzano, Italy, September 2010.) ESEM'10, IEEE, New York, NY, to appear.

[3] Ko A. and Myers, B. 2008. Debugging reinvented: Asking and answering why and why note questions about program behavior, In *Proceedings of the ACM/IEEE International Conference on Software Engineering*, (Leipzig, Germany, May 10-18, 2008.) ICSE'08, ACM, New York, NY, 301-310. DOI=http://doi.acm.org/10.1145/1368088.1368130.

[4] Lawrance, J., Burnett, M., Bellamy, R., Bogart, C., and Swart, C. 2010. Reactive information foraging for evolving goals, In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, (Atlanta, GA, April 10-15, 2010.) CHI'10, ACM, New York, NY, 25-34. DOI=http://doi.acm.org/10.1145/1753326.1753332.

[5] Scaffidi, C., Shaw, M. and Myers, B. 2005. Estimating the numbers of end users and end user programmers, In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing*, (Dallas, TX, September 20-24, 2005.) VL/HCC'05. IEEE, New York, NY, 207-214. DOI=10.1109/VLHCC.2005.34.