Human Factors in the Study of Automatic Software Repair

Future Directions for Research with Industry

Emily Winter
e.winter@lancaster.ac.uk
Lancaster University

David Bowes
d.h.bowes@lancaster.ac.uk
Lancaster University

Steve Counsell
steve.counsell@brunel.ac.uk
Brunel University

Tracy Hall
tracy.hall@lancaster.ac.uk
Lancaster University

Saemundur Haraldsson
saemundur.haraldsson@stir.ac.uk
Stirling University

Vesna Nowack
v.nowack@qmul.ac.uk
Queen Mary, University of London

John Woodward
j.woodward@qmul.ac.uk
Queen Mary, University of London

ABSTRACT

Automatic software repair represents a significant development in software engineering, promising considerable potential change to the working procedures and practices of software engineers. Technical advances have been the focus of many recent publications. However, there has not been an equivalent growth of studies of human factors within automatic software repair. This position paper presents the case for increased research in this area and suggests three key focuses and approaches for a future research agenda. All three of these enable industry-based software engineers not just to provide feedback on automatic software repair tools but to participate in shaping these technologies so that they meet developer and industry needs.

CCS CONCEPTS

• **Software and its engineering → Software development techniques**; • **Human-centered computing → Human computer interaction (HCI)**.

KEYWORDS

human factors, automatic software repair

ACM Reference Format:
Emily Winter, David Bowes, Steve Counsell, Tracy Hall, Saemundur Haraldsson, Vesna Nowack, and John Woodward. 2020. Human Factors in the Study of Automatic Software Repair: Future Directions for Research with Industry. In *IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3387940.3392176

ICSEW'20, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7963-2/20/05…$15.00
https://doi.org/10.1145/3387940.3392176

1 INTRODUCTION

Automatic software repair (ASR) responds to what has been described as 'a pressing need for automatic techniques to supplement manual software development with inexpensive tools' [4]. There has been a rapid growth of technical innovation in this area, motivated by a desire to reduce the temporal and financial resources involved in practices, such as manual testing and debugging.

Despite the growing domain of ASR, there has been very little focus on the human factors or aspects in this field. However, within software engineering more generally, there is increasing recognition of the need to study human factors in order to fully understand software engineering practices. Recent work has considered, for example, such topics as motivation [1], software engineers' values [8], and key software engineering skills [5]. Central to such work is the notion that software engineering is never just about technical decisions, but is an inherently human activity, involving complex interactions of the human and the technical.

Whilst some work on ASR mentions potential developer attitudes [2] [4], there have been very few empirical studies of human factors in this domain. There are, however, a small number of empirical studies in the closely related field of automated testing [3] [7]. From a consideration of such work, this position paper suggests three important directions for future research.

2 FUTURE DIRECTIONS FOR RESEARCH

Beyond Usability: The existing research highlights what software developers and testers consider to be the advantages and disadvantages of automated testing but is limited to a focus on the usability of automated testing [3]. Similar concerns are demonstrated in the ASR literature, such as whether developers can understand fixes [2] [4].

This is an important research focus, but there are also other components that a study of human factors in ASR should consider. One key research question, for example, might be how ASR tools impact upon developer and tester job satisfaction. The adoption of new tools will often displace certain tasks and work patterns and lead to the development of new ones. ASR, for example, may reduce the need for manual debugging while yielding patches that need to be validated and that the developer may need to maintain. This is

likely to have implications for software professionals' perceptions of their work life and their job satisfaction. To fully consider developer attitudes to ASR, it is therefore important that a broad range of themes are considered, not just usability.

An openness to considerations beyond usability also indicates a broader conceptual shift. This is marked by a move away from merely eliciting feedback from software professionals to an approach that is more responsive to software developers' needs, opening space for increased dialogue and iteration of tools.

This is important given that some recent work has cautioned against paying too much attention to developer attitudes. Monperrus [6], for example, contends that 'we should not be afraid of alien ways of writing code' and also critiques evaluative studies of ASR tools for their reliance on developers' perceptions of solutions that 'look good'. Whilst ASR may well produce unfamiliar-looking fixes or patches that will ultimately require a process of developer adaptation, it is of vital importance that developers' concerns and priorities are not disregarded. Studies of software developers' perceptions and attitudes should have the openness to support software developers in shaping ASR tools; such studies shouldn't aspire simply to evaluation, or to persuading developers of the tools' efficacy. As an example, developers' suggestions might be used to shape how patches are presented to the developer.

Longitudinal Studies: Most empirical studies in the area of automated testing consider attitudes to such tools at a fixed point in time. Whilst these studies are still valuable and provide significant insights, there are limitations, and studies of human factors in ASR should be aware of this. More longitudinal studies could offer real benefit in providing insight into how attitudes might change over time. This could include studying attitudes at three key stages: prior to the introduction of a particular ASR tool; during the early use of a tool; and several months later. This could enable research into how concerns expressed by developers at the beginning might either diminish, continue or increase over the following two stages.

A longitudinal research approach also offers greater possibility for software professionals to interact more with the developers and designers of ASR tools. This allows industry-based software engineers to potentially feed into the process and shape the development of these tools. Through deeper and longer-term engagement with industry, academics can iterate to produce ASR tools that more fully serve the needs of developers, rather than such tools being seen as externally imposed.

Diversity of Appropriate Social Research Methods: Most empirical studies in automated testing have used surveys and interviews. Alongside such methods, the study of human factors in ASR calls for others, such as focus groups and ethnographic research. This requires appropriate expertise in qualitative methods.

Ethnographic research in particular has the potential to yield key insight into how tools are used in situ and also how they might be discussed in more informal work contexts. Given that software engineering is rarely a solo activity, focus groups offer an opportunity to consider the group dynamics at work, including social norms and conventions that may shape individual attitudes and perceptions of ASR tools. Reception of new tools does not occur in a vacuum, but is likely to be influenced by many aspects of workplace culture.

There is also space for the kind of research methods rarely used in empirical studies of human factors in software engineering. For example, facilitated workshops using design thinking techniques could be an effective way to engage developers around a specific ASR tool and gain their input. Again, this helps to position the research encounter in a more dialogical way, rather than positioning industry-based engineers as more passive providers of feedback.

3 OBSTACLES TO CONDUCTING USER STUDIES

The main barrier to conducting thorough user studies in the way that has been described here is the fact that gaining access to industry can be difficult. When access to industry is achieved, it may also be challenging to persuade key stakeholders that it is worthwhile for employees to spend their time interacting with researchers.

The directions presented here intend to combat some of these difficulties. In particular, longitudinal studies enable the building of trust and relationships. Additionally, a more developer-centered approach yields clearer potential benefit to industry partners.

4 CONCLUSION

The rapid technical advances in the field of ASR promise significant changes to the practice of software engineering. This rise should be accompanied by a focus on the human elements of this picture. This position paper has argued for the need for studies of human factors in ASR that move beyond a focus on usability, are longitudinal, and use a variety of social research methods. All of these could contribute to a more developer-centered approach that pays greater attention to developer attitudes and perceptions and also allows software professionals to be involved in shaping ASR tools and techniques so that they are more fitting to industry needs.

5 ACKNOWLEDGMENTS

This work is funded by an Engineering and Physical Sciences Research Council grant EP/S005730/1.

REFERENCES

[1] Cesar França, Fabio Q. B. da Silva, and Helen Sharp. 2018. Motivation and Satisfaction of Software Engineers. *IEEE Transactions on Software Engineering* (2018), 1–1. https://doi.org/10.1109/TSE.2018.2842201

[2] Luca Gazzola, Daniela Micucci, and Leonardo Mariani. 2019. Automatic Software Repair: A Survey. *IEEE Transactions on Software Engineering* 45, 01 (jan 2019), 34–67. https://doi.org/10.1109/TSE.2017.2755013

[3] Azham Hussain, Hamidah Abdul Razak, and Emmanuel O. C. Mkpjiogum. 2017. The perceived usability of automated testing tools for mobile applications. *Journal of Engineering, Science and Technology* 12 (2017), 86–93.

[4] Claire Le Goues, Stephanie Forrest, and Westley Weimer. 2013. Current challenges in automatic software repair. *Software Quality Journal* 21, 3 (1 9 2013), 421–443. https://doi.org/10.1007/s11219-013-9208-0

[5] Paul Luo Li, Andrew J. Ko, and Jiamin Zhu. 2015. What Makes a Great Software Engineer?. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1* (Florence, Italy) *(ICSE '15)*. IEEE Press, 700–710.

[6] Martin Monperrus. 2014. A Critical Review of "Automatic Patch Generation Learned from Human-Written Patches": Essay on the Problem Statement and the Evaluation of Automatic Software Repair. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 234–242. https://doi.org/10.1145/2568225.2568324

[7] Raphael Pham, Stephan Kiesling, Leif Singer, and Kurt Schneider. 2017. Onboarding inexperienced developers: struggles and perceptions regarding automated testing. *Software Quality Journal* 25, 4 (2017), 1239–1268.

[8] Emily Winter, Stephen Forshaw, Lucy Hunt, and Maria Angela Ferrario. 2019. Advancing the Study of Human Values in Software Engineering. In *Proceedings of the 12th International Workshop on Cooperative and Human Aspects of Software Engineering* (Montreal, Quebec, Canada) *(CHASE '19)*. IEEE Press, 19–26. https://doi.org/10.1109/CHASE.2019.00012