Do Automated Program Repair Techniques Repair Hard and Important Bugs?

Manish Motwani Sandhya Sankaranarayanan René Just Yuriy Brun
University of Massachusetts, Amherst
{mmotwani, ssankar, rjust, brun}@cs.umass.edu

The full version of this article [8] can be found at http://dx.doi.org/10.1007/s10664-017-9550-0.

ACM Reference Format:
Manish Motwani Sandhya Sankaranarayanan René Just Yuriy Brun. 2018. Do Automated Program Repair Techniques Repair Hard and Important Bugs?. In *ICSE '18: 40th International Conference on Software Engineering , May 27–June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 1 page. https://doi.org/10.1145/3180155.3182533

Automated program repair techniques use a buggy program and a partial specification (typically a test suite) to produce a program variant that satisfies the specification. While prior work has studied patch quality [10, 11] and maintainability [2], it has not examined whether automated repair techniques are capable of repairing defects that developers consider important or that are hard for developers to repair manually. This paper tackles those questions.

Our study considers nine automated repair techniques for C and Java: AE [12], GenProg [13], a Java reimplementation of GenProg [7], Kali [10], a Java reimplementation of Kali [7], Nopol [1], Prophet [6], SPR [5], and TrpAutoRepair [9].

We analyze popular bug tracking systems and source code repositories to identify parameters relevant to defect importance, independence, and complexity, and test effectiveness. We compute these parameters for two benchmarks of defects often used for automated program repair, ManyBugs [4] (185 C defects) and Defects4J [3] (357 Java defects). We further analyze developer-written patches for these defects to identify characteristics of those patches that may influence automated repair.

Our study answers the following questions: Is a repair technique's ability to produce a patch for a defect correlated with that defect's (RQ1) importance, (RQ2) complexity, (RQ3) effectiveness of the test suite, or (RQ4) dependence on other defects? (RQ5) What characteristics of the developer-written patch are significantly associated with a repair technique's ability to produce a patch? And, (RQ6) what defect characteristics are significantly associated with a repair technique's ability to produce a high-quality patch?

We find that (RQ1) Java repair techniques are moderately more likely to patch higher-priority defects; for C, there is no correlation. There is little to no consistent correlation between producing a patch and the time taken by developer(s) to fix the defect, as well as the number of software versions affected by that defect. (RQ2) C repair techniques are less likely to patch defects that required developers to write more lines of code and edit more files. (RQ3) Java repair techniques are less likely to patch defects with more triggering or more relevant tests. Test suite statement coverage has little to no consistent correlation with producing a patch. (RQ4) Java repair techniques' ability to patch a defect does not correlate with that defect's dependence on other defects. (RQ5) Repair techniques struggle to produce patches for defects that required developers to insert loops or new function calls, or change method signatures. Finally, (RQ6) Only two of the considered repair techniques, Prophet and SPR, produce a sufficient number of high-quality patches to evaluate. These techniques were less likely to patch more complex defects, and they were even less likely to patch them correctly.

The main contributions of this paper are:

- The publicly-released annotation of 409 defects in ManyBugs and Defects4J, to be used for evaluating automated repair applicability.
- A methodology for evaluating the applicability of repair techniques, with the goal of encouraging research to focus on important and hard defects.
- The evaluation of nine automated program repair techniques' applicability to 409 ManyBugs and Defects4J defects.

ACKNOWLEDGMENTS

This work is supported by the National Science Foundation under grants CCF-1453474 and CCF-1564162.

REFERENCES

[1] F. DeMarco, J. Xuan, D. L. Berre, and M. Monperrus. Automatic repair of buggy if conditions and missing preconditions with SMT. In *CSTVA*, pages 30–39, 2014.
[2] Z. P. Fry, B. Landau, and W. Weimer. A human study of patch maintainability. In *ISSTA*, pages 177–187, 2012.
[3] R. Just, D. Jalali, and M. D. Ernst. Defects4J: A database of existing faults to enable controlled testing studies for Java programs. In *ISSTA*, pages 437–440, 2014.
[4] C. Le Goues, N. Holtschulte, E. K. Smith, Y. Brun, P. Devanbu, S. Forrest, and W. Weimer. The manybugs and introclass benchmarks for automated repair of C programs. *IEEE TSE*, 41(12):1236–1256, December 2015.
[5] F. Long and M. Rinard. Staged program repair with condition synthesis. In *ESEC/FSE*, pages 166–178, 2015.
[6] F. Long and M. Rinard. Automatic patch generation by learning correct code. In *POPL*, pages 298–312, 2016.
[7] M. Martinez, T. Durieux, R. Sommerard, J. Xuan, and M. Monperrus. Automatic repair of real bugs in Java: A large-scale experiment on the Defects4J dataset. *EMSE*, 22(4):1936–1964, April 2017.
[8] M. Motwani, S. Sankaranarayanan, R. Just, and Y. Brun. Do automated program repair techniques repair hard and important bugs? *EMSE*, 2018.
[9] Y. Qi, X. Mao, and Y. Lei. Efficient automated program repair through fault-recorded testing prioritization. In *ICSM*, pages 180–189, Sept. 2013.
[10] Z. Qi, F. Long, S. Achour, and M. Rinard. An analysis of patch plausibility and correctness for generate-and-validate patch generation systems. In *ISSTA*, 2015.
[11] E. K. Smith, E. Barr, C. Le Goues, and Y. Brun. Is the cure worse than the disease? Overfitting in automated program repair. In *ESEC/FSE*, pages 532–543, 2015.
[12] W. Weimer, Z. P. Fry, and S. Forrest. Leveraging program equivalence for adaptive program repair: Models and first results. In *ASE*, 2013.
[13] W. Weimer, T. Nguyen, C. Le Goues, and S. Forrest. Automatically finding patches using genetic programming. In *ICSE*, pages 364–374, 2009.

ICSE '18, May 27–June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5638-1/18/05.
https://doi.org/10.1145/3180155.3182533