Industrial Experience of Genetic Improvement in Facebook

Nadia Alshahwan
Facebook Inc.

***Abstract*—Facebook recently had their first experience with Genetic Improvement (GI) by developing and deploying the automated bug fixing tool SapFix. The experience was successful resulting in landed fixes but also very educational. This paper will briefly outline some of the challenges for GI that were highlighted by this experience as well as a look at future directions in the area of mobile apps.**

INTRODUCTION

While developing and deploying SapFix (the automated end-to-end repair tool) at Facebook, a few open problems were highlighted as the more pressing challenges for this type of technique to be successful. This paper will focus on three of those challenges: fix detection, engineer perception and participation and scalability and timeliness. Details about the tool itself can be found in the paper by Marginean, et al[1].

Fix Detection

Any fault repair tool is only as good as the tests it uses to validate the produced fixes. If the tests have low reproducibility, we can not determine with high confidence if the fault disappeared because it was fixed or because our tests are flaky. Even if the tests were 100% reproducible, there is still a possibility that the fix only masked the fault while another test we are not using could reveal it in a different way.

Moreover, the fix could have introduced another fault that our tests cannot expose. Human reviewers might catch these cases but not as fast as an automated tool. It would be more efficient for such cases to never make it to the review process.

Any development in reproducibility and coverage of testing approaches can help mitigate these issues.

Engineer Perception and Participation

It comes as no surprise that engineers were split in their reaction to the tool between enthusiasts and skeptics. We would only be worried if SapFix was met with indifference. Each of these reaction poses its own risks and challenges.

Skeptics might refrain from engaging with the tool until it delivers the quality they would find satisfying. This could mean that we miss important feedback that can help us improve and develop the tool. The challenge here is to understand and address any concerns these engineers might have and to be open about the technique's strengths and limitations.

Enthusiasts on the other hand pose a risk if they trust the tool too much and maybe accept changes that might have hidden side effects. A way to mitigate that is to add disclaimers and warnings on patches submitted by the tool reminding engineers that these are generated by an error prone automated approach.

Scalability and Timeliness

There is a great tension between the quality of the solution an automated technique can produce and the timeliness of this solution. This is a problem we also face in Sapienz[2] the automated test design tool.

Naturally if you give a technique more resources and time, the solution produced is expected to be of higher quality. However, for an automated repair tool to be successful, it needs to produce a fix before engineers spend time and effort trying to solve the same fault. We saw examples of this in practice when some of our fixes were deemed correct but did not land because the issue was already fixed by an engineer.

The number of mutants produced by a repair tool has a great effect on the scalability and timeliness of the technique. Each mutant has to be built and tested which could be very time consuming. For that reason, SapFix incorporates fix templates mined from human fixes into the mutation process to produce a smaller number of possible patches that have a higher probability of being at least compilable. However, this limits the issues we can fix to those that have fix templates. Being able to produce a large number of random mutants but also have smart rules to filter then down before reaching the build/test step of the process would be ideal.

FUTURE DIRECTION

One of the biggest areas of interest for mobile apps in industry currently is performance improvement. Any reduction in execution or load times or the size of data transferred is important.

Another direction for GI could be improving the layout of an app to enhance usability and engagement. We could utilize time spent data to determine which parts of the app we want to highlight. Once those popular features are determined, the fitness could be how easy those parts are to reach and how close features are if they are usually used together. Determining the fitness could be done by running a random approach such as Android Monkey and calculating the probability of covering those parts over multiple runs.

REFERENCES

[1] A. Marginean, J. Bader, S. Chandra, M. Harman, Y. Jia, K. Mao, A. Mols, and A. Scott, "SapFix: Automated end-to-end repair at scale," in *International Conference on Software Engineering (ICSE) Software Engineering in Practice (SEIP) track*, Montreal, Canada, 2019.

[2] N. Alshahwan, X. Gao, M. Harman, Y. Jia, K. Mao, A. Mols, T. Tei, and I. Zorin, "Deploying search based software engineering with Sapienz at Facebook (keynote paper)," in *SSBSE 2018*, 2018, pp. 3–45.

978-1-7281-2268-7/19/$31.00 ©2019 IEEE
DOI 10.1109/GI.2019.00010