TSE Final Project Proposal

Team APR

Members:

1. George Cherian (UNI: gc2920)
2. Mavis Athene Chen (UNI: mu2288)
3. Yin Zhang (UNI: yz4053)

We have listed down our motivations for each team member in picking up APR.

1. George

My motivation in picking up APR is mainly to see the current methods being used in APR and how they can be applied to industry problems to cut down time spent debugging. I hope to be able to develop a mechanism which is usable by developers to easily highlight buggy locations in the code and try to provide fixes for the bugs to the developer in an expedient manner.

The final project would be related to my midterm paper since both were related to APR and the first proposal would be very much linked to the work in my midterm paper and would be a deeper dive into the thought process of creating an end-to-end solution adding in correctness as a major factor as well

2. Mavis

I wanted to pick up APR as a topic in my final project because my midterm paper revealed many areas that APR could be improved upon. One area is the varying performance of APR tools across bug categories and I believe proposal 1 could lead to interesting insights on that. Generally, through this final project, I would like to have a more “hands on” experience with the workflows and technical details of developing an APR tool.

3. Yin

I studied the APR topic in my midterm paper and as I have mentioned in the paper, there are several aspects that could be improved in the final project:

1) the APR process pipeline usually includes two core phases - detection and fixes, but it still has various works that surround it, such as validation and overfitting validation. The whole framework with complete process of the repair would benefit the developers more than decentralized tools. Even many related techniques have been made to detect, fix, validate and so on, we still need to explore the complete product that integrates the whole process to support a high degree of automation. Thus, proposal 1 could help us explore more possibilities on the topic of fully automated program repairs.

2) From the experiment that I have conducted on the static analysis tool SpotBugs, I found that there are several limitations of its bug-searching functionality that might be able to further improved, details are included in proposal 2.

We had two proposals and we wanted to get some feedback about likeliness and feasibility as well as range of implementation before committing to one of them for the final project.

Proposal 1: Automated Program Repair – Combination of Multiple Approaches, Automated Correctness Evaluation and User Studies

Our first proposal would be creating an extension of multiple approaches like Cardumen, jGenProg, ARJA in a single system which would run these mechanisms and then create a ranking mechanism for identifying the best fix which along with metrics which we will define like # of lines changed will also evaluate correctness vs plausibility using the framework defined by this paper :
https://dl.acm.org/doi/pdf/10.1145/3180155.3180182

We hope to be able to not only generate a fix for more bugs than previous works were able to do individually but also then be able to evaluate the best fix for a bug and then finally decide on the correctness vs plausibility of the bug itself.

If we are successful in building out this tool it would help save on time for developers to fix bugs and also save on time reviewing “bad” bug fixes suggested by the system

We would be running the program itself over Defects4J’s library and comparing the fixes created by the tool against the developer suggested fix for a “Ground Truth” analysis

We would be trying to evaluate against the base versions of the tools themselves and also look into other tools like JaRFly, Genesis and QuixBugs amongst others which we would use to evaluate the success of our tool

Time permitting , Once we have a system which can take in the bugs and run them through these different approaches and find an appropriate fix which can be marked as correct then we would like to conduct a user study with the new tool created to get user feedback on a subset of the bug fixes with regards to the fix itself and how much time they believe that the fix would save as well as understanding if they feel a particular fix is deficient (Plausible but not correct) in some way

Proposal 2: Automated Program Repair –SpotBugs Integration with APR methodology

Spotbugs is a buggy code detection program, it performs a static analysis of java code which divides the buggy codes into several categories and gives detailed error descriptions. Spotbugs will not eliminate all errors in the program, but it will reduce the number of errors found in production. It performs well both in time cost and numbers of bugs found. One problem of Spotbugs is that it defines the template and patterns to the bug searching, the pre-defined bug modes may limit the exploration of the buggy code due to large granularity. Some bugs may be classified into inappropriate categories or directly ignored. Furthermore, the description of the bug may not conform to the actual situation, but will still be limited by the template and be given an inappropriate description. (https://spotbugs.github.io/)

One approach that we thought of improving the template/fix patterns in spot bugs is maybe to strip the current templates and try implementing a tool that would provide template based fixes elsewhere. This would mimic the performance of Phoenix (https://doi.org/10.1145/3338906.3338952) and Getafix (https://doi.org/10.1145/3360585). This may be a taxing job because we are essentially building an APR tool from scratch based on a completely new static analysis tool.

Overall, we have a less concrete proposal for extending/modifying spotbugs, but am open to any ideas and suggestions in this direction as well.