Automated Program Repair

Introduction:

Automated Program Repair is a rapidly growing research area in computer science [1]. There are numerous tools and techniques that have been developed to automatically find and fix bugs in source code. This topic is directly related to Quality Engineering and improving the quality of software by reducing the cost and effort of fixing bugs. This white paper provides a broad overview of automated program repair, the advantages and disadvantages it, and what it would look like for the Satellite 6 Quality Engineering team at Red Hat.

Background:

The concept of automated program repair has been around for a while, but it recently starting to become a reality in software engineering. The process is similar to the human debugging process, except everything is done programmatically. Automated program repair is made up of four main steps:

1. Fault Identification

The first step is to define what is a bug, which is different for every project, and to determine if the observed behavior is a bug. For automated program repair, this is primarily done by using unit tests. It assumes the test suite adheres to the program specifications and tracks failing test cases. Future work is looking into using natural language processing to parse information from bug reports.

2. Fault localization

Step 2 is to find the incorrect lines causing the bug. This step is complex because there are numerous factors and multiple lines of code that can contribute to unexpected behavior in a program. There have been a number of techniques developed to identify buggy lines using static analysis, code commit history, and analyzing unit test cases.

3. Patch Generation

After determining an issue exists and finding the location in the code, the next step is to generate a fix for the faulty lines. This is probably the most difficult step of the process because it involves automatically performing the human task of writing code. There are several different techniques for creating code to fix bugs. Generate-and-validate uses genetic programming to find a fix by creating variants of the code. Semantic analysis employs symbolic execution and program synthesis to generate patches based on a set of constraints. Semantic code search uses the expected input-output behavior of a program to search for code in a determined database or repository (i.e. Github) that has the desired functionality. Each technique has its own advantages and disadvantages and differ in their emphasis on different qualities of the patches generated.

4. Patch Evaluation

Finally, patch evaluation determines if the generated fix actually fixed the bug. This is also important because changing one line of code often impacts another part of the program, and you want to ensure the generated patch did not cause more issues. Additionally, there can be other issues with computer-generated patches such as readability, scalability, maintainability, reliability, etc.

Advantages:

There are many advantages of automated program repair in practice. The debugging process is a very time-consuming and expensive activity in the software development process, annually costing hundreds of billions of dollars globally [2]. Automatically repairing bugs can greatly reduce this cost and help save companies money and time. A study at UVA analyzed open source C projects and determined their tool GenProg could fix bugs for just under $8 each [3]. Additionally, even though focus on testing and quality is increasing, the total number of bugs in software is also increasing rapidly. Top tech companies such as Microsoft, Google, Mozilla, Facebook, Twitter, Paypal, and more have hosted bug bounty programs which pay bug hunters to report bugs for them [4]. This has it's own pros and cons, but automatically detecting and fixing bugs can help companies find vulnerabilities and make their software more secure.

Disadvantages:

The main disadvantage of automated program repair scaling. The research and tools listed at [1] have shown automated program repair is effective, however the studies are performed on smaller code bases and student programming assignments. The technology isn't there yet to automatically repair larger projects due to their size and complexity. For patch generation the search space for creating a fix can be infinite, which is much harder to account for enterprise software with thousands of lines of code (or more) and other dependencies.

Satellite 6 QE:

In my experience so far with Satellite 6 QE, automation seems to be a main focuses of the team in the weekly meetings, automating tests with Robottelo, and working with satellite-populate. Unfortunately, as mentioned in the previous section, the technology is not there yet to be effective for Satellite due to the large code base and many different components of the product. Additionally a large majority of the current automated program repair tools available have been developed for C or Java, which also doesn't help with Satellite.

If this is an eventual goal for the team, then it would take a large amount of effort to reach but could pay off in the long run. Automatically repairing bugs in Satellite can help reduce the time and effort required when a bug is reported by a customer to releasing the fix. This would allow more time for developers to continue improving Satellite and adding new features and for quality engineers to focus on creating new tools and improving existing resources for finding and preventing issues in Satellite to make sure it maintains a high quality. The process could start by working on smaller projects such as creating an exhaustive test suite (possibly use robottelo), programmatically determining bugs and identifying faulty lines, selecting a patch generation method, etc. while focusing on just one aspect of Satellite like the Hammer CLI.

Sources:

[1] Program repair
[2] Debugging software cost has risen to $312 billion per year globally
[3] https://www.cs.virginia.edu/~weimer/p/weimer-icse2012-genprog-preprint.pdf
[4] Bug bounty program - Wikipedia