Fix Director for Automated Program Repair

Juhyoung Kim
Department of Computer Science and Engineering
SungKyunkwan University
Suwon, Republic of Korea
kjhkjh612@skku.edu

Eunseok Lee
Department of Computer Science and Engineering
SungKyunkwan University
Suwon, Republic of Korea
leees@skku.edu

***Abstract*—As the scale of software increases, automated program repair (APR) is required as an essential technology to reduce bug-fixing manual effort. One of the most effective APR approaches is the template-based APR. Although this data-driven approach has made many improvements in fixing bugs, there is still a lack of information utilization at each stage of the program repair. In this paper, we propose a fix pattern prioritization APR technique. We do not just use the test cases to determine whether the source file is fault or not, but as a material for why the bug occurred. With this information, we select a more appropriate fix action and apply it first to increase the efficiency of the automated program repair.**

***Keywords*—Software engineering, Automated program repair, Fix pattern prioritization**

I. Introduction

The scale of modern software developments are growing. As a result, the tasks of maintaining and repairing software have been consuming more time and cost. Recent study showed that these process account for about 50% of the software development [1]. To reduce this excessive cost, the demand for system to automatically fix software defects has increased. In this aspect, various approaches of automated program repair techniques have been proposed. Most of them are search-based APR [2, 3, 4, 5, 6, 7, 8]. They generate patch candidates using predefined sets of mutation operators with fault space determined by Fault Localization (FL) techniques. And they search correct patch that passes all given test cases among generated patch candidates from APR techniques. Despite this novel approach, there are some problems. First, search space to generate correct patch is usually huge. Second problem is correct patches may not exist in the search space, so APR techniques cannot fix the program. In addition, even if we increase the search space to find the correct patch, we are not sure that we can find the correct patch. Rather, it can only cause an increases cost in time and effort. To solve this problem, other approaches have been proposed including template-based [9,10,11,12,13,14], machine learning-based [15,16,17,18], etc. In particular, template-based APR techniques have been evaluated as the most effective and superior performance techniques [19]. This approach is strategy of APR to generate concrete patches based on fix patterns (also referred to as fix templates) which are predefined using human written patches from open source repositories. Because these approaches leverage predefined template from human-written patches, we can find more correct patches and reduce the cost. Despite these efforts, APR systems still do not have enough effectiveness for the cost it puts in. For example, many techniques including template-based APR go through three steps to repair the program. First step is the fault localization (FL), which identifies the fault Location. The second step is to generate candidate patches based on the proposed approaches in various ways. Final step is the validation, which checks the candidate patch that passes all given test cases. In this process, information used at individual step is often used only at each step and discarded. Specifically, the occurrence of failed test cases is used to check whether the source code has defects, but the information of the failed test cases is rarely used when going through other repair process. This disconnection in utilizing information can be a critical loss for program repair. In this paper, we leverage the information of the failed test cases to overcome disconnection in utilizing collected information. Using information of the failed test cases, we can infer what is the cause making defects and find a basis for which fix pattern should be prioritized in proceeding program repair. And these processes achieve performance improvements that reduce the cost.

II. Background and Motivation

A. Template-based APR

Template-based APR techniques generate bug-fixing candidate patches automatically using fix patterns extracted from human written patches. Existing template-based APR