ABSTRACT

Tests generated by automated test generators are seldom processed for further use in software testing. The goal of this research is to find techniques for processing such generated tests. These techniques should produce new test cases by transforming generated tests, or use information in the tests to guide future test generation.

The first part of this dissertation intends to devise a technique to use the information in the generated tests to improve test generation. The proposed technique would improve the frequency of coverage for important code targets in swarm testing. We evaluate the effectiveness of this technique by comparing the frequency of coverage in (traditional) un-directed swarm testing and directed swarm testing.

In the second part of this dissertation, we propose a set of test reduction criteria to generate new test cases by reducing already generated test cases. The goal of the proposed reduction criteria is to speed up testing while preserving (most of) fault-detection capability of the original tests. Effectiveness of the reduction techniques is measured by comparing the size, coverage and mutation-killing power of the reduced tests with the original tests.

1. INTRODUCTION

Automatically generated tests have been successful in uncovering new bugs in many important programs and libraries. As the cost of generating new test cases decreases, tendency to discard already generated tests increases. For example, in most fuzzing scenarios, the tests that do not find bugs are discarded.

Generated tests, in particular randomly generated tests, have rarely been subject to analysis and transformation to create new tests or improve the test generators. Our vision is these generated tests, like other software artifacts should be analyzed and processed for ....

Goal of this research is to generate (better) new tests by analysis of the existing generated tests. Particularly, we are interested in finding and exploiting associations between test features and behavior of software under test (SUT).

2. RESEARCH PROBLEM AND POSSIBLE SOLUTIONS

Figure 1 illustrates high-level goals of this research. The first goal is to analyze the generated test cases, find associations, and use the information in generated tests and devise a technique to improve the coverage of less-frequently covered a program entity. We define hitting fraction as

\[
\text{Hitting Fraction (HF)} = \frac{\text{Number of test cases covering } t}{\text{Total number of test cases}}
\]

If \( HF(t) = f \), where \( f \) is relatively small, e.g. \( f < 0.15 \), the goal of focused testing is to generate new test suites such that \( HF(t) = f' \), such that \( f < f' \). Note that \( HF \) can lose meaning if it is manipulated to become a large number by reducing the size of test suite (i.e. denominator). Thus, it is important to avoid this pitfall by considering the suites that are generated within sensible time limit. In contrast with targeted test generation techniques, such as search based test generation and symbolic execution, focused tests are aimed to cover already covered targets \( t \), where \( HF(t) \) is small.

The rational behind increasing the frequency of coverage is to increase the likelihood of triggering failures in a buggy statement. When a bug is covered multiple times using different execution paths, it is more likely to manifest a failure than it is covered only once by a single execution path.

The most related work to the first goals are targeted testing techniques, most notable: (dynamic) symbolic execution [] and search based test generation techniques []. Perhaps the most known random test generator is Randoop Randoop [8] which generates tests for object-oriented programs by calling random APIs. Randoop uses feedback to guide test sequence creation. Nighthawk [2] utilizes genetic algorithms to optimize random tester given a goal (i.e., fitness function). Adaptive random testing [3, 4] uses a distance measure to select more diverse (i.e. uniformly distributed) tests.

The second goal of this research is to process generated random tests for a useful task. Usually, in a test generation session, thousands of tests are generated. These test cases
usually are large and redundant. Thus, in most cases, the generated tests that do not failure are thrown away. In this research we want to explore possibilities reusing these tests. To this end, we intend to propose reduction criteria to distill important part of test cases by removing redundancies. This criteria could be test coverage and mutation coverage which are widely considered as proxy for fault detection of test cases.

There are several other techniques for test case minimization such as [11, 10, 12]. Perhaps the most well-know test case minimization technique is delta debugging [12]. Given a property of interest in a test case, delta debugging reduces the test case to a minimal test case that preserves the property. The property of interest usually is a failure in a failing test case. Test suite reduction techniques have been studied for a long time [7, 6, 9], but they have pursue different goal than test case reduction that we are pursuing.

3. EXPECTED CONTRIBUTIONS

By using the tests to generate new test cases, or to direct test generation, we attempt to prove that generated tests are software artifacts worth processing. We expect this dissertation to improve our understanding of the extent to which information in generated tests can be used to improve software testing itself.

4. PROGRESS

So far, we have developed a technique for focused testing called directed swarm testing [1]. Directed swarm testing, in non-trivial test subjects like GCC, could improve the hitting fraction to 2.5x, on average and in some cases up to 8x.

We have proposed coverage-based reduction of test cases for quick regression testing [5]. Recently, we proposed non-adequate reduction of test cases which relaxes the coverage requirement and also is able to preserve a large portion of mutants detected by test cases.

5. FUTURE PLAN

In future, we want to address two problems. First, we are interested to see if the reduced test cases can be used to improve the seeded symbolic execution (aka dynamic symbolic execution). Previous work [13] has shown that reducing test cases based on coverage and prioritize them can improve the overall speed of path exploration in DSE. We want to evaluate the impact of using non-adequately reduced test cases as seed in DSE. Second, a problem in test generation is when a certain parts of code is error-prone for some reasons (e.g. new unstable feature) most of tests that hit those parts fail. The result is a large number of failing test cases that really do not add extra value in software process, because developers already know that those parts of code are buggy. We want to explore possibility of using the technique in directed swarm testing to avoid those parts of code.

6. REFERENCES