Adapting Amplified Unit Tests for Human Comprehension

Internship — M2 SIF

Supervisor: Benoit Baudry
KTH, Sweden
Wednesday 7th February, 2018–Friday 22nd June, 2018

Simon Bihel `simon.bihel@ens-rennes.fr`

Wednesday 27th June, 2018

University of Rennes I
École Normale Supérieure de Rennes

Introduction

Test Suites

Context

- Software projects are now accompanied by strong test suites
- Takes time to write
- Still missing some bugs due to focus on nominal paths when writing test cases

Test Suites

Context

- Software projects are now accompanied by strong test suites
- Takes time to write
- Still missing some bugs due to focus on nominal paths when writing test cases

Related works

- Measure the quality of test suites
- Automatically write test suites
- Amplify existing test suites

Concepts

System-Under-Test: function, class, whole program...

Inputs E.g. function parameters, method calls to setup and stimulate an object

Assertions Used to test whether the function's output is correct, that the object is in the right state

Test Example

```
1  public class TreeListTest {
2      @Test
3      public void testIterationOrder() {
4          TreeList tl = new TreeList(10);
5          for (int i = 0; i < 10; i++) {
6              tl.add(i);
7          }
8          ListIterator it = tl.listIterator();

9          int i = 0;
10         while (it.hasNext()) {
11             Integer val = it.next();
12             assertEquals(i++, val.intValue());
13         }
14     }
```

Metrics for Test Suites

Goal

Detect parts that are not tested.

Metrics for Test Suites

Goal

Detect parts that are not tested.

Code Coverage

Number of instructions or branches executed by the test suite.

Metrics for Test Suites

Goal

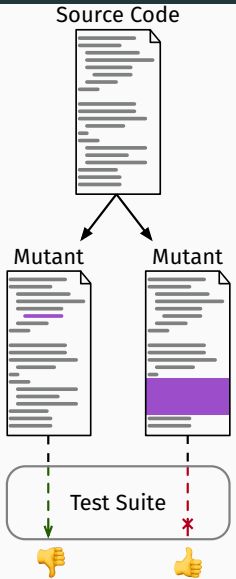
Detect parts that are not tested.

Code Coverage

Number of instructions or branches executed by the test suite.

Mutation Testing

1. Create *mutants* (i.e. bugged versions) of the main software (e.g. change a > with a <=).
2. Count how many mutants for which the test suite fail.



Automated Test Generation

Goal

Generate tests from scratch to fulfill a given metric.

Large search space of instructions and values.

Search-based techniques[1]

Random, iterative and heuristic-based techniques (e.g. Genetic Algorithms, simulated annealing).

[1] McMinn, “Search-based software testing: Past, present and future”, 2011.
[2] Barr et al., “The oracle problem in software testing: A survey”, 2015.

Automated Test Generation

Goal

Generate tests from scratch to fulfill a given metric.

Large search space of instructions and values.

Search-based techniques[1]

Random, iterative and heuristic-based techniques (e.g. Genetic Algorithms, simulated annealing).

The oracle problem[2]

What should the output of a test be?

→ Avoid this by focusing on regression testing.

[1] McMinn, "Search-based software testing: Past, present and future", 2011.
[2] Barr et al., "The oracle problem in software testing: A survey", 2015.

Test Suite Amplification

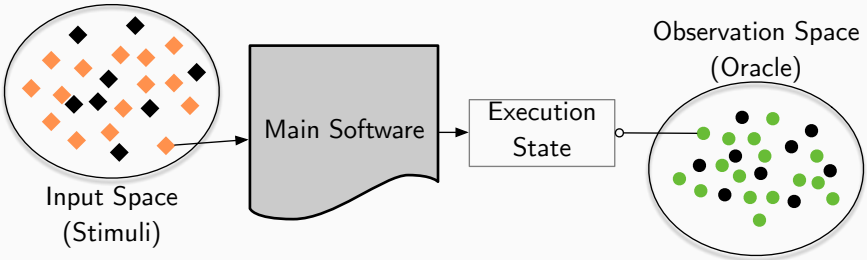
Motivations[3]

- Reduce search-space by using the existing test suite as a (good) starting population.
- Use knowledge in hand-written tests for a better oracle.

[3] Danglot et al., “The Emerging Field of Test Amplification: A Survey”, 2017.

Goal

Create tests for undetected mutants.



[4] Baudry et al., "DSpot: Test Amplification for Automatic Assessment of Computational Diversity", 2015.

DSpot — Amplification Operators

Input amplification

Literals → replaced with neighbor values.

Method calls → duplicated, removed or made-up (with random or default parameters).

DSpot — Amplification Operators

Input amplification

Literals → replaced with neighbor values.

Method calls → duplicated, removed or made-up (with random or default parameters).

Assertion amplification

Capture the state of the system after the test's execution.

Usage in the DevOps Context

New tests ought to be approved by the developers.

Generating Descriptive Messages

Software Artefact Summarization

Includes: documentation for source code, code changes, test cases.

[5] Li et al., “Automatically documenting unit test cases”, 2016.

Software Artefact Summarization

Includes: documentation for source code, code changes, test cases.

UnitTestScribe[5] Summarises actions in natural language (Software Word Usage Model, method stereotypes).

[5] Li et al., "Automatically documenting unit test cases", 2016.

Contribution

Pull Request message.

Explain:

1. the modifications made,
2. the reason why the new test was kept (i.e. mutation score).

Logging Amplifications

Category	`ASSERT`, `ADD`, `DEL`, `MODIFY`
Parent	Type of the parent of modified AST node.
Role	Role for the parent node (e.g. argument for a method call).
Old value	Textual representation of the old node.
New value	Textual representation of the new node.

Logging Amplifications

Category	`ASSERT`, `ADD`, `DEL`, `MODIFY`
Parent	Type of the parent of modified AST node.
Role	Role for the parent node (e.g. argument for a method call).
Old value	Textual representation of the old node.
New value	Textual representation of the new node.

Logging call in each amplifier.

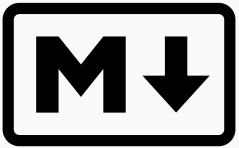
Mutation Score Information

Location Modified method and line.

Description High-level, sometimes vague, natural language description from PIT[6].

[6] Coles et al., “PIT: a practical mutation testing tool for java”, 2016.

Target Platform



Markdown markup language allows for enhanced messages. Code snippet, links to code lines, better rendering, ...

Major platforms use it.

GitHub GitLab

But not all. Bitbucket

Demo

Demo XWiki

Evaluation

No case study with users. Only discussions with my supervisor.

Conclusion

Summary

Too early to tackle the topic of generated tests explanation.

- Lack of precise information (e.g. killed mutants *per* test).
- Small user-base.

Summary

Too early to tackle the topic of generated tests explanation.

- Lack of precise information (e.g. killed mutants *per* test).
- Small user-base.

What I should have focused solely on.

- Cleaning amplified tests.
- Independent mutation score explanation.

[7]

- Test Suite Generation *from scratch*.
- State-of-the-Art & industry grade.

Differences

- Treats test suites as a whole.
- GA with tests cases as genes.

Automatic Documentation

Automatic name generation[8].

[7] Fraser and Arcuri, "Evosuite: automatic test suite generation for object-oriented software", 2011.

[8] Daka, Rojas, and Fraser, "Generating unit tests with descriptive names or: Would you name your children thing1 and thing2?", 2017.

Is Documentation Essential? — Yes.

Developers surveys[9] and experiments[10].

Documentation (e.g. JavaDoc, naming) has many advantages[11], especially for generated tests[12]:

- faster to get familiar with the test;
- faster fault localisation; and
- helps to build trust in a test generator if it can provide a proof for its result.

[9] Daka and Fraser, "A survey on unit testing practices and problems", 2014; Prado et al., "WAP: Cognitive aspects in unit testing: The hunting game and the hunter's perspective", 2015; Prado and Vincenzi, "Advances in the Characterization of Cognitive Support for Unit Testing: The Bug-Hunting Game and the Visualization Arsenal", 2016; Prado and Vincenzi, "Towards cognitive support for unit testing: a qualitative study with practitioners", 2018; Li et al., "Automatically documenting unit test cases", 2016.

[10] Panichella et al., "The impact of test case summaries on bug fixing performance: An empirical investigation", 2016.

[11] Daka, Rojas, and Fraser, "Generating unit tests with descriptive names or: Would you name your children thing1 and thing2?", 2017.

[12] Rojas and Fraser, "Is search-based unit test generation research stuck in a local optimum?", 2017; Shamshiri et al., "How Do Automatically Generated Unit Tests Influence Software Maintenance?", 2018.