Adaptating Amplified Unit Tests for Human Comprehension

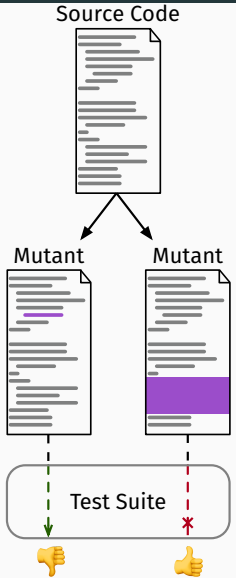
Simon Bihel simon.bihel@ens-rennes.fr

Thursday 8th March, 2018 @ KTH

University of Rennes I
École Normale Supérieure de Rennes

Mutation Testing

Evaluating the quality of a test suite by injecting bugs

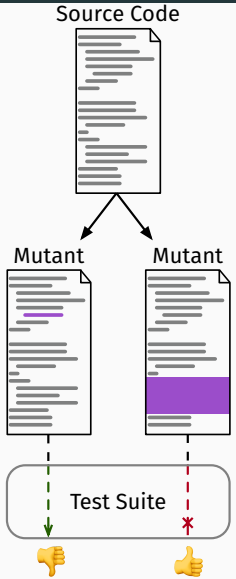


Mutation Testing

Evaluating the quality of a test suite by injecting bugs

Examples of *mutators*:

- change a > condition with <;
- delete the body of a method.



Mutation Testing

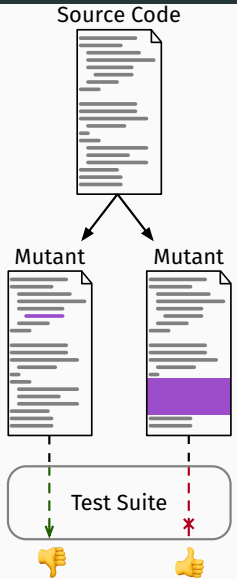
Evaluating the quality of a test suite by injecting bugs

Examples of *mutators*:

- change a > condition with <;
- delete the body of a method.

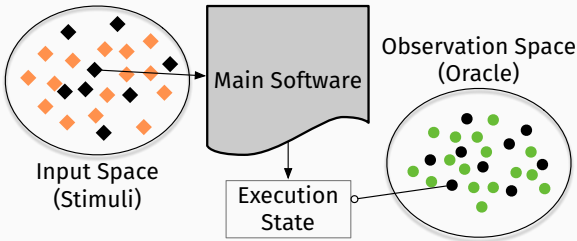
Goal

Enhance test suite by detecting new mutants



DSpot[1]

Randomly modifies test cases:

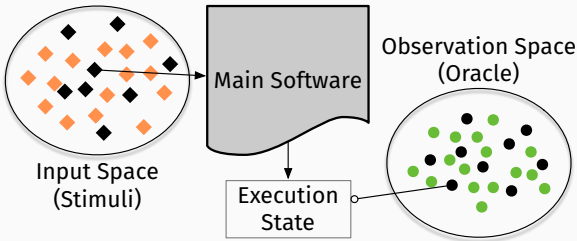


[1] https://github.com/STAMP-project/dspot

DSpot[1]

Randomly modifies test cases:

- new inputs to trigger new behaviors;

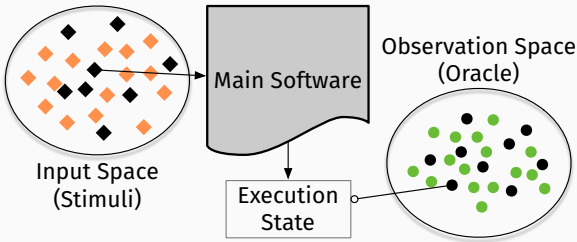


[1] https://github.com/STAMP-project/dspot

DSpot[1]

Randomly modifies test cases:

- new inputs to trigger new behaviors;
- new assertions for unchecked properties;

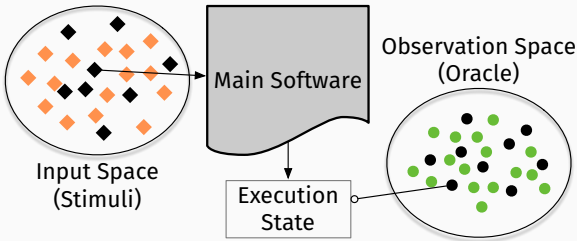


[1] https://github.com/STAMP-project/dspot

DSpot[1]

Randomly modifies test cases:

- new inputs to trigger new behaviors;
- new assertions for unchecked properties;
- targets regression.

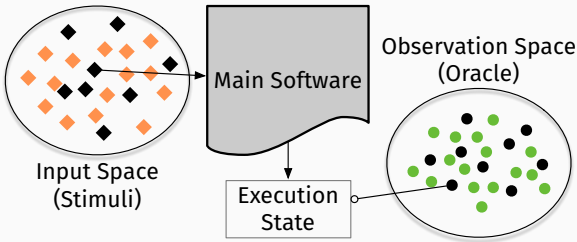


[1] https://github.com/STAMP-project/dspot

DSpot[1]

Randomly modifies test cases:

- new inputs to trigger new behaviors;
- new assertions for unchecked properties;
- targets regression.



Benjamin Danglot, INRIA Lille, France

[1] https://github.com/STAMP-project/dspot

Example[2]

```java
@Test
public void immutableGraph() {
  MutableGraph<String> mutableGraph = GraphBuilder.directed().build();
  mutableGraph.addNode("A");
  ImmutableGraph<String> immutableGraph = ImmutableGraph.copyOf(mutableGraph);

  assertThat(immutableGraph).isNotInstanceOf(MutableValueGraph.class);
  assertThat(immutableGraph).isEqualTo(mutableGraph);

  mutableGraph.addNode("B");
  assertThat(immutableGraph).isNotEqualTo(mutableGraph);
}
```

[2] https://github.com/google/guava/blob/master/guava-tests/test/com/google/common/graph/ImmutableGraphTest.java#L29-L40

Example of Amplification

```java
@Test
public void immutableGraph() {
  MutableGraph<String> mutableGraph = GraphBuilder.directed().build();
  mutableGraph.addNode("A");
  mutableGraph.addNode("C");
  ImmutableGraph<String> immutableGraph = ImmutableGraph.copyOf(mutableGraph);

  assertThat(immutableGraph).isNotInstanceOf(MutableValueGraph.class);
  assertTrue(immutableGraph.nodes().contains("A"));
  assertThat(immutableGraph).isEqualTo(mutableGraph);

  mutableGraph.addNode("B");
  assertThat(immutableGraph).isNotEqualTo(mutableGraph);
}
```

Goal

→ Human-friendly, high-level, natural language description

Goal

→ **Human-friendly, high-level, natural language description**

First objective: an explanation per mutant kill.

Identify Relevant Statements

Identify the killing assertion.

Identify Relevant Statements

Identify the killing assertion.

Simple static slicing:

- starting from the killing assertion;
- control-flow slicing and
- data-flow slicing.

Identify Relevant Statements

Identify the killing assertion.

Simple static slicing:

- starting from the killing assertion;
- control-flow slicing and
- data-flow slicing.

Java slicing tool

T.J. Watson Libraries for Analysis (WALA)[3]

Established library with active development.

[3] https://github.com/wala/WALA

Cleaning the Amplifications

Minimisation phase → less explanation to generate.

Cleaning the Amplifications

Minimisation phase → less explanation to generate.

1. Remove assertions that never fail.

Cleaning the Amplifications

Minimisation phase → less explanation to generate.

1. Remove assertions that never fail.
2. Remove statements not present in a slice.

Cleaning the Amplifications

Minimisation phase → less explanation to generate.

1. Remove assertions that never fail.
2. Remove statements not present in a slice.
3. Remove statements with no impact. → long process

Cleaning the Amplifications

Minimisation phase → less explanation to generate.

1. Remove assertions that never fail.
2. Remove statements not present in a slice.
3. Remove statements with no impact. → long process

Minimizing only amplifications

- Less time consuming.
- Keep the original part intact → better for understanding.

UnitTestScribe[4][5]

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

[5] `https://github.com/boyangwm/UnitTestScribe`

UnitTestScribe[4][5]

🎉 Empirical study and survey → need for automated documentation.

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

[5] `https://github.com/boyangwm/UnitTestScribe`

UnitTestScribe[4][5]

🎉 Empirical study and survey → need for automated documentation.

- Summarises actions in natural language (Software Word Usage Model, method stereotypes).

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

[5] `https://github.com/boyangwm/UnitTestScribe`

UnitTestScribe[4][5]

🎉 Empirical study and survey → need for automated documentation.

- Summarises actions in natural language (Software Word Usage Model, method stereotypes).

Code changes summarisation (i.e. commit message generation)

Focused on general source code (add feature, fix bug, ...).

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

[5] `https://github.com/boyangwm/UnitTestScribe`

Related Works Limit

Problem

No “why” information. Paraphrasing the code.

Related Works Limit

Problem

No “why” information. Paraphrasing the code.

Reason for amplifications to exist → mutant kill.

Natural Description

Better oracle:

- what part of the system was left out;
- avoid using terms as mutant.

Natural Description

Better oracle:

- what part of the system was left out;
- avoid using terms as mutant.

New kind of behaviour:

- differences in traces to explain how a bug is triggered;
- use mutators stereotypes to explain how the inputs are modified.

Experiment Protocol

- Study repos with strong commit message guidelines (e.g. Google).

Experiment Protocol

- Study repos with strong commit message guidelines (e.g. Google).
- Performances.

Experiment Protocol

- Study repos with strong commit message guidelines (e.g. Google).
- Performances.
- Survey with real developers.