Finding Failure-Inducing Changes in Java Programs using Change Classification

Maximilian Stoerzer, Barbara G. Ryder and Xiaoxia Ren, Frank Tip

Presenter: Sait Tuna Onder
Overview

- Introduction
- Approach
- Definitions
- Implementation and Evaluation
- Case Studies
- Discussion
Introduction - I

● Difficulties to detect failure origin
● Proposal: Change Classification to catch programmers’ attention.
● Provides an automated assistance to detect failures
● Relies on Change Impact Analysis
● Possible Outcomes:
  ○ Improved
  ○ Degraded
  ○ Unchanged or some combination
● Design of number of classifiers that assign Colors
  ○ Red: Highly likely to be Failure-Inducing
  ○ Green: Highly unlikely to be Failure-Inducing
  ○ Yellow: Somewhere in between
Introduction - II

- **JUnit/CIA (Change Impact Analysis)**
  - Divides an edit into its atomic changes
  - Identifies tests affected by the edit
  - Determining affecting changes for each of these tests

- **Case Study**
  - Manual Examination of the code
  - Measure the effectiveness of each of the classifiers

- **Case Study 1: Student Projects**
  - Initial Development of an application
  - Small differences between versions

- **Case Study 2: Daikon**
  - Mature application
  - Larger changes
Atomic Changes - I

- Added Classes: AC
- Deleted Classes: DC
- Added Methods: AM
- Deleted Methods: DM
- Changed Method Bodies: CM
- Added Fields: AF
- Deleted Fields: DF
- Lookup Changes: LC (Y, X.m())

Note: This changes might be dependent to each other.
public class A {
    public A(int i) { x = i; }
    public void foo() { x = x + 0; }
    public void bar() { y = x; }\textsuperscript{3}
    public void zap() {
    }
    public void zip() { y = x; }\textsuperscript{5}
    public int x;
    public static int y;\textsuperscript{4}
    public static int getY() { return y; }\textsuperscript{6,7}
}

class B extends A {
    public B(int j) { super(j); }
    public void foo() {
    }
    public void bar() { x++; }\textsuperscript{8}
}

class C extends A {
    public C(int k) { super(k); }\textsuperscript{9,10,11}
    public void zap() { x = 5; }
}

class D extends A {
    public D(int l) { super(l); }
    public void foo() { x--; }\textsuperscript{12}
}
Unit Tests

(a)

```java
public class A {
    public A(int i) { x = i; }
    public void foo() { x = x + 0; }
    public void bar() { y = x; }
    public void zap() {
    }
    public void zip() { y = x; }
    public int x;
    public static int y;  
    public static int gety() { return y; }  
}

public class B extends A {
    public B(int j) { super(j); }
    public void foo() {
    }
    public void bar() { x++; }
}

public class C extends A {
    public C(int k) { super(k); }
    public void zap() { x = 5; }
}

class D extends A {
    public D(int l) { super(l); }
    public void foo() { x--; }
}
```

(b)

```java
public class Tests extends TestCase {
    public void testPassPass() {
        A a = new A(5);
        a.foo(); a.bar();
        Assert.assertTrue(a.x == 5);
    }
    public void testPassFail() {
        A a = new C(7);
        a.foo(); a.zap(); a.zip();
        Assert.assertTrue(a.x == 7);
    }
    public void testFailPass() {
        A a = new B(8);
        a.foo(); a.bar(); a.zip();
        Assert.assertTrue(a.x == 9);
    }
    public void testFailFail() {
        A a = new B(6);
        a.foo(); a.bar();
        Assert.assertTrue(a.x == 11);
    }
    public void testCrashFail() {
        A a = new D(5); a.foo();
        int i = a.x / (a.x - 5);
        Assert.assertTrue(a.x == 5);
    }
```
Determining Affected Tests

Call Graphs for the original program
Determining Affecting Changes

Call Graphs for the edited program

Dashed boxes indicate changed/added methods, dashed arrows indicate changed calling relationships between methods.
Change Classification

Which of those 12 changes are the likely reason(s) for the rest failure(s)?

- A change that affects only improving tests
  - testFailPass(), testCrashFail()
- A change that affects only worsening tests
  - testPassFail()
- A change that affects both
Definitions

- **Test Result Model**
  - $R = \{\text{PASS, FAIL, CRASH}\}$
  - $\text{CRASH} < \text{FAIL} < \text{PASS}$

- **Test Classification**
  - $WT = \{ T \in \mathcal{T} \mid R_{\text{orig}}(T) > R_{\text{edit}}(T) \}$
  - $IT = \{ T \in \mathcal{T} \mid R_{\text{edit}}(T) > R_{\text{orig}}(T) \}$

- **Change Influence**
  - $\text{Worsening} = \{ A \mid A \in \mathcal{A}, WT \cap AT(A) \neq \emptyset \}$
  - $\text{Improving} = \{ A \mid A \in \mathcal{A}, IT \cap AT(A) \neq \emptyset \}$
  - $\text{SomeFailFail} = \{ A \mid \exists T \in AT(A), R_{\text{orig}}(T) = R_{\text{edit}}(T) \in \{\text{FAIL, CRASH}\} \}$
  - $\text{SomePassPass} = \{ A \mid \exists T \in AT(A), R_{\text{orig}}(T) = R_{\text{edit}}(T) = \text{PASS} \}$
  - $\text{OnlyPassPass} = \{ A \mid \forall T \in AT(A), R_{\text{orig}}(T) = R_{\text{edit}}(T) = \text{PASS} \}$

- $AT(A)$ represents tests affected by atomic change $A$
- $AC(T)$ represents the atomic changes affecting a given test $T$
Classifiers

- Simple, Rr, Rs, Gr and Gs

<table>
<thead>
<tr>
<th>Coloring</th>
<th>Criteria</th>
<th>strict</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>$R_r$: $(A \not\in Improving \land A \in Worsening)$</td>
<td>$R_s$: $(A \not\in Improving \land A \in Worsening \land A \not\in SomePassPass)$</td>
</tr>
<tr>
<td>Green</td>
<td>$G_r$: $A \in OnlyPassPass \lor (A \in Improving \land A \not\in Worsening)$</td>
<td>$G_s$: $(A \in Improving \land A \not\in Worsening \land A \not\in SomeFailFail)$</td>
</tr>
<tr>
<td>Yellow</td>
<td>$A \not\in Red, A \not\in Green, AT(A) \neq \emptyset$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Change</th>
<th>simple</th>
<th>$R_r/G_r$</th>
<th>$R_s/G_r$</th>
<th>$R_r/G_s$</th>
<th>$R_s/G_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Yellow</td>
<td>Red</td>
<td>Yellow</td>
<td>Red</td>
<td>Yellow</td>
</tr>
<tr>
<td>2</td>
<td>Yellow</td>
<td>Green</td>
<td>Green</td>
<td>Yellow</td>
<td>Yellow</td>
</tr>
<tr>
<td>3</td>
<td>Green</td>
<td>Green</td>
<td>Green</td>
<td>Green</td>
<td>Green</td>
</tr>
<tr>
<td>4</td>
<td>Yellow</td>
<td>Yellow</td>
<td>Yellow</td>
<td>Yellow</td>
<td>Yellow</td>
</tr>
<tr>
<td>5</td>
<td>Yellow</td>
<td>Yellow</td>
<td>Yellow</td>
<td>Yellow</td>
<td>Yellow</td>
</tr>
<tr>
<td>6</td>
<td>Gray</td>
<td>Gray</td>
<td>Gray</td>
<td>Gray</td>
<td>Gray</td>
</tr>
<tr>
<td>7</td>
<td>Gray</td>
<td>Gray</td>
<td>Gray</td>
<td>Gray</td>
<td>Gray</td>
</tr>
<tr>
<td>8</td>
<td>Red</td>
<td>Red</td>
<td>Red</td>
<td>Red</td>
<td>Red</td>
</tr>
<tr>
<td>9</td>
<td>Red</td>
<td>Red</td>
<td>Red</td>
<td>Red</td>
<td>Red</td>
</tr>
<tr>
<td>10</td>
<td>Red</td>
<td>Red</td>
<td>Red</td>
<td>Red</td>
<td>Red</td>
</tr>
<tr>
<td>11</td>
<td>Gray</td>
<td>Gray</td>
<td>Gray</td>
<td>Gray</td>
<td>Gray</td>
</tr>
<tr>
<td>12</td>
<td>Red</td>
<td>Green</td>
<td>Green</td>
<td>Green</td>
<td>Green</td>
</tr>
</tbody>
</table>
Implementation: JUnit/CIA
Case Studies

● Student Projects
  ○ 40 small student projects
  ○ Time-out after 10 seconds resulting CRASH
  ○ Some minor post processing of the code
  ○ Total of 1175 version pairs in which 556 of them contained meaningful changes
  ○ 110 of the 556 version pairs had worsening tests
  ○ Failure-inducing changes could be manually identified for 98 of these 110 version pairs
  ○ 61 of the 98 version pairs have more than one change
  ○ These 61 version pairs have total of 1295 atomic changes

<table>
<thead>
<tr>
<th>number of version pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>written by students</td>
</tr>
<tr>
<td>that contain meaningful changes</td>
</tr>
<tr>
<td>with associated worsening tests</td>
</tr>
<tr>
<td>with identifiable failure-inducing changes</td>
</tr>
<tr>
<td>where versions pairs differ by &gt;1 change</td>
</tr>
</tbody>
</table>
Student Projects

- **Red Changes**
  - *Recall* is the fraction of failure-inducing changes colored red
  - *Precision* is the ratio of the actual failure-inducing red changes to all red changes
    - False-Positives: 1 - precision, False-Negatives: 1 - recall

- **Green Changes**
  - *Recall* is the fraction of all non-failure inducing changes colored Green
  - *Precision* is the ratio of actual non-failure inducing changes to all green changes
    - Precision is always 1 since green changes never affect worsening tests
  - */Gr Recall: 0.19, */Gs Recall: 0.15
Student Projects

- Total of 444 worsening tests in the 61 version pairs.
- $R_{r/*}$ classifiers colored all the failure-inducing changes in 211/444 (47.5%) of the tests.
- $R_{s/*}$ 113/444 (25.5%), Simple 67/444 (15.1%).
- Conclusion: $R_{r/Gr}$ classifiers work best.
Daikon Study

- Authors could not find any worsening tests in several versions of Daikon
  - Several unit tests changed among versions
  - Old tests with the edited version produced 2 test failures
- For the first test:
- Rr/* failed because they classified all 35 effecting changes as Red, because there were no improving tests
- Rs/* classified 2 failure-inducing changes Red as well as 2 of 33 remaining changes. Rest is Yellow. Therefore, 4 of 35 catch the attention of the user
- Rs/* outperformed Rr/* significantly
Discussion

- What do you think about using different classifiers in respect to project type or development phase?
- Do you think the results are satisfactory. Would you ever use JUnit/CIA?