An Empirical Study on the Use of Defect Prediction for Test Case Prioritization

International Conference on Software Testing, Verification and Validation
Xi'an, China
April 22-27 2019

DAVID PATERSON,
UNIVERSITY OF SHEFFIELD

JOSE CAMPOS,
UNIVERSITY OF WASHINGTON

RUI ABREU,
UNIVERSITY OF LISBON

GREGORY M. KAPFHAMMER,
ALLEGHENY COLLEGE

GORDON FRASER,
UNIVERSITY OF PASSAU

PHIL MCMINN,
UNIVERSITY OF SHEFFIELD

DPATERSON1@SHEFFIELD.AC.UK
In software development, our goal is to minimize the impact of faults. If we know that a fault exists, we can use *fault localization* to pinpoint the code unit responsible. If we don’t know that a fault exists, we can use *defect prediction* to estimate which code units are likely to be faulty.
Defect Prediction

Class A: 33%
Class B: 10%
Class C: 72%
Class D: 3%
## Defect Prediction

### Code Smells
- Feature Envy
- God Class
- Inappropriate Intimacy

### Code Features
- Cyclomatic Complexity
- Method Length
- Class Length

### Version Control Information
- Number of Changes
- Number of Authors
- Number of Fixes

---

**DPATerson1@sheffield.ac.uk**
Why Do We Prioritize Test Cases?

Regression testing can account for up to 80% of the total testing budget, and up to 50% of the cost of software maintenance.

In some situations, it may not be possible to re-run all test cases on a system.

By prioritizing test cases, we aim to ensure faults are detected in the smallest amount of time irrespective of program changes.

DPATERSON1@SHEFFIELD.AC.UK
How Do We Prioritize Test Cases?

<table>
<thead>
<tr>
<th></th>
<th>t₁</th>
<th>t₂</th>
<th>t₃</th>
<th>t₄</th>
<th>t₅</th>
<th>t₆</th>
<th>t₇</th>
<th>t₈</th>
<th>t₉</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version 1</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Version 2</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Version 3</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Version 4</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Version 5</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Version 6</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Version 7</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Version 8</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Version 9</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>tₙ₋₃</th>
<th>tₙ₋₂</th>
<th>tₙ₋₁</th>
<th>tₙ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version 1</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Version 2</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Version 3</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Version 4</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Version 5</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Version 6</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Version 7</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Version 8</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Version 9</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Version n</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Version n+1</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
How Do We Prioritize Test Cases?

Code Coverage
"How many lines of code are executed by this test case?"

Test History
"Has this test case failed recently?"

Defect Prediction:
"What is the likelihood that this code is faulty?"

```java
public int abs(int x){
    if (x >= 0) {
        return x;
    } else {
        return -x;
    }
}
```
### Defect Prediction for Test Case Prioritization

<table>
<thead>
<tr>
<th>Class</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClassA</td>
<td>33%</td>
</tr>
<tr>
<td>ClassB</td>
<td>10%</td>
</tr>
<tr>
<td>ClassC</td>
<td>72%</td>
</tr>
<tr>
<td>ClassD</td>
<td>3%</td>
</tr>
</tbody>
</table>

DPATerson1@sheffield.ac.uk
Defect Prediction for Test Case Prioritization

ClassC

72%
Defect Prediction for Test Case Prioritization

ClassC

72%

Test Cases that execute code in ClassC:
- TestClass.testOne
- TestClass.testSeventy
- OtherTestClass.testFive
- OtherTestClass.testThirteen
- TestClassThree.test165

How do we order these test cases before placing them in the prioritized suite?
Secondary Objectives

Test Cases that execute code in ClassC:

- TestClass.testOne
- TestClass.testSeventy
- OtherTestClass.testFive
- OtherTestClass.testThirteen
- TestClassThree.test165

We can use one of the features described earlier (e.g. code coverage) as a way of ordering the *subset* of test cases
### Secondary Objectives

<table>
<thead>
<tr>
<th>Test Cases that execute code in ClassC:</th>
<th>Lines Covered:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- TestClass.testOne</td>
<td>25</td>
</tr>
<tr>
<td>- TestClass.testSeventy</td>
<td>32</td>
</tr>
<tr>
<td>- OtherTestClass.testFive</td>
<td>144</td>
</tr>
<tr>
<td>- OtherTestClass.testThirteen</td>
<td>8</td>
</tr>
<tr>
<td>- TestClassThree.test165</td>
<td>39</td>
</tr>
</tbody>
</table>

We can use one of the features described earlier (e.g. code coverage) as a way of ordering the *subset* of test cases.
Secondary Objectives

Test Cases that execute code in ClassC:  

- OtherTestClass.testFive 144
- TestClassThree.test165 39
- TestClass.testSeventy 32
- TestClass.testOne 25
- OtherTestClass.testThirteen 8

We can use one of the features described earlier (e.g. code coverage) as a way of ordering the subset of test cases
Defect Prediction for Test Case Prioritization

ClassC

72%

Test Cases that execute code in ClassC:

- OtherTestClass.testFive
- TestClassThree.test165
- TestClass.testSeventy
- TestClass.testOne
- OtherTestClass.testThirteen

Prioritized Test Suite:
Defect Prediction for Test Case Prioritization

ClassC

72%

Test Cases that execute code in ClassC:

Prioritized Test Suite:

- OtherTestClass.testFive
- TestClassThree.test165
- TestClass.testSeventy
- TestClass.testOne
- OtherTestClass.testThirteen
Defect Prediction for Test Case Prioritization

ClassA

33%

Test Cases that execute code in ClassA: Lines Covered:
- ClassATest.testA 14
- ClassATest.testB 27
- ClassATest.testC 9

Prioritized Test Suite:
- OtherTestClass.testFive
- TestClassThree.test165
- TestClass.testSeventy
- TestClass.testOne
- OtherTestClass.testThirteen
Defect Prediction for Test Case Prioritization

Prioritized Test Suite:
- OtherTestClass.testFive
- TestClassThree.test165
- TestClass.testSeventy
- TestClass.testOne
- OtherTestClass.testThirteen

ClassA

Test Cases that execute code in ClassA: Lines Covered:
- ClassATest.testB 27
- ClassATest.testA 14
- ClassATest.testC 9

33%
Defect Prediction for Test Case Prioritization

ClassA

33%

Test Cases that execute code in ClassA:

Prioritized Test Suite:
- OtherTestClass.testFive
- TestClassThree.test165
- TestClass.testSeventy
- TestClass.testOne
- OtherTestClass.testThirteen
- ClassATest.testB
- ClassATest.testA
- ClassATest.testC
By repeating this process for all classes in the system, we generate a fully prioritized test suite based on defect prediction.
Empirical Evaluation
Empirical Evaluation

Defect Prediction: Schwa [1]

Uses version control information to produce defect prediction scores comprised of weighted number of commits, authors, and fixes related to a file

Empirical Evaluation

**Defect Prediction:** Schwa [1]

Uses version control information to produce defect prediction scores comprised of weighted number of commits, authors, and fixes related to a file

**Faults:** DEFECTS4J [2]

Repository containing 395 real faults collected across 6 open-source Java projects

Empirical Evaluation

Defect Prediction: Schwa [1]
Uses version control information to produce defect prediction scores comprised of weighted number of commits, authors, and fixes related to a file

Faults: DEFECTS4J [2]
Repository containing 395 real faults collected across 6 open-source Java projects

Test Prioritization: KANONIZO [3]
Test Case Prioritization tool built for Java Applications

Research Objectives

1. Discover the best parameters for defect prediction in order to predict faulty classes as soon as possible

2. Compare our approach against existing coverage-based approaches

3. Compare our approach against existing history-based approaches
Parameter Tuning

1. Revisions Weight
2. Authors Weight
3. Fixes Weight
4. Time Weight

\[ \sum RevisionsWeight + AuthorsWeight + FixesWeight = 1 \]
### Parameter Tuning

The sum of the revisions weight, authors weight, and fixes weight is equal to 1:

\[ \sum \text{RevisionsWeight} + \text{AuthorsWeight} + \text{FixesWeight} = 1 \]

<table>
<thead>
<tr>
<th>Revisions Weight</th>
<th>Authors Weight</th>
<th>Fixes Weight</th>
<th>Time Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.9</td>
<td>0.1</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.8</td>
<td>0.2</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.9</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

- **726 Valid Configurations**
Parameter Tuning

- Select 5 bugs from each project at random
- For each bug/valid configuration
  - Initialize Schwa with configuration and run
  - Collect “true” faulty class from DEFECTS4J
  - Calculate index of “true” faulty class according to prediction
## Parameter Tuning

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.jfree.chart.plot.XYPlot</td>
<td>99.98</td>
</tr>
<tr>
<td>org.jfree.chart.ChartPanel</td>
<td>99.92</td>
</tr>
<tr>
<td>org.jfree.chart.renderer.xy.AbstractXYItemRenderer</td>
<td>99.30</td>
</tr>
<tr>
<td>org.jfree.chart.plot.CategoryPlot</td>
<td>99.20</td>
</tr>
<tr>
<td>org.jfree.chart.renderer.AbstractRenderer</td>
<td>98.58</td>
</tr>
<tr>
<td>org.jfree.chart.renderer.category.AbstractCategoryItemRenderer</td>
<td>98.02</td>
</tr>
<tr>
<td>org.jfree.chart.renderer.category.BarRenderer</td>
<td>95.82</td>
</tr>
<tr>
<td>org.jfree.chart.renderer.xy.XYBarRenderer</td>
<td>95.22</td>
</tr>
<tr>
<td>org.jfree.chart.plot.Plot</td>
<td>94.75</td>
</tr>
<tr>
<td>org.jfree.data.time.TimeSeriesCollection</td>
<td>94.53</td>
</tr>
<tr>
<td>org.jfree.data.xy.XYSeriesCollection</td>
<td>94.48</td>
</tr>
<tr>
<td>org.jfree.chart.plot.junit.XYPlotTests</td>
<td>94.35</td>
</tr>
<tr>
<td>org.jfree.chart.renderer.category.StatisticalLineAndShapeRenderer</td>
<td>93.80</td>
</tr>
<tr>
<td>org.jfree.chart.renderer.xy.XYItemRenderer</td>
<td>92.43</td>
</tr>
<tr>
<td>org.jfree.chart.panel.RegionSelectionHandler</td>
<td>92.24</td>
</tr>
<tr>
<td>org.jfree.data.general.DatasetUtilities</td>
<td>92.11</td>
</tr>
<tr>
<td>org.jfree.chart.axis.CategoryAxis</td>
<td>90.82</td>
</tr>
</tbody>
</table>

+1091 more…

org.jfree.data.time.junit.TimePeriodValuesTests.MySeriesChangeListener 8.30

DPATerson1@SHEffield.AC.UK
**Parameter Tuning**

### DEFECTS4J

- **“True” Faulty Class**
- **org.jfree.data.general.DatasetUtilities**

### Table: Prediction of Class Name

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.jfree.chart.plot.XYPlot</td>
<td>99.98</td>
</tr>
<tr>
<td>org.jfree.chart.ChartPanel</td>
<td>99.92</td>
</tr>
<tr>
<td>org.jfree.chart.renderer.xy.AbstractXYItemRenderer</td>
<td>99.30</td>
</tr>
<tr>
<td>org.jfree.chart.plot.CategoryPlot</td>
<td>99.20</td>
</tr>
<tr>
<td>org.jfree.chart.renderer.AbstractRenderer</td>
<td>98.58</td>
</tr>
<tr>
<td>org.jfree.chart.renderer.category.AbstractCategoryItemRenderer</td>
<td>98.02</td>
</tr>
<tr>
<td>org.jfree.chart.renderer.category.BarRenderer</td>
<td>95.82</td>
</tr>
<tr>
<td>org.jfree.chart.renderer.xy.XYBarRenderer</td>
<td>95.22</td>
</tr>
<tr>
<td>org.jfree.chart.plot.Plot</td>
<td>94.75</td>
</tr>
<tr>
<td>org.jfree.data.time.TimeSeriesCollection</td>
<td>94.53</td>
</tr>
<tr>
<td>org.jfree.data.xy.XYSeriesCollection</td>
<td>94.48</td>
</tr>
<tr>
<td>org.jfree.chart.plot.junit.XYPlotTests</td>
<td>94.35</td>
</tr>
<tr>
<td>org.jfree.chart.renderer.category.StatisticalLineAndShapeRenderer</td>
<td>93.80</td>
</tr>
<tr>
<td>org.jfree.chart.renderer.xy.XYItemRenderer</td>
<td>92.43</td>
</tr>
<tr>
<td>org.jfree.chart.panel.RegionSelectionHandler</td>
<td>92.24</td>
</tr>
<tr>
<td>org.jfree.data.general.DatasetUtilities</td>
<td>92.11</td>
</tr>
<tr>
<td>org.jfree.chart.axis.CategoryAxis</td>
<td>98.82</td>
</tr>
</tbody>
</table>

**Position: 16**

- **org.jfree.data.time.junit.TimePeriodValuesTests.MySeriesChangeListener** 0.30

+1091 more…

DPATerson1@sheffield.ac.uk
<table>
<thead>
<tr>
<th>Revisions Weight</th>
<th>Authors Weight</th>
<th>Fixes Weight</th>
<th>Time Range</th>
<th>Average Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.7</td>
<td>0.1</td>
<td>0.2</td>
<td>0.4</td>
<td>49.49</td>
</tr>
<tr>
<td>0.1</td>
<td>0.3</td>
<td>0.4</td>
<td></td>
<td>49.26</td>
</tr>
</tbody>
</table>

**Revisions are important** – best results were observed when revisions weight was high.

**TOP 3:**

**No single configuration significantly outperformed all others**

**Author Weight should be low** – this indicates that the number of authors has little impact.

**Fixes weight is similar in both**

**BOTTOM:**

The 3 worst results all occurred when the time range was 1 – this indicates that newer commits are more important to analyze.
## Parameter Tuning

<table>
<thead>
<tr>
<th>Project</th>
<th>Top 1</th>
<th>Top 1%</th>
<th>Top 5%</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chart</td>
<td>1</td>
<td>7</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>Closure</td>
<td>1</td>
<td>31</td>
<td>77</td>
<td>107</td>
</tr>
<tr>
<td>Lang</td>
<td>9</td>
<td>11</td>
<td>26</td>
<td>39</td>
</tr>
<tr>
<td>Math</td>
<td>1</td>
<td>15</td>
<td>40</td>
<td>55</td>
</tr>
<tr>
<td>Mockito</td>
<td>3</td>
<td>14</td>
<td>29</td>
<td>33</td>
</tr>
<tr>
<td>Total</td>
<td>17</td>
<td>87</td>
<td>200</td>
<td>267</td>
</tr>
</tbody>
</table>

For 67.5% of the bugs, the faulty class was inside the top 10% of classes.

For 17 faults, Schwa predicted the correct faulty class.

Schwa can effectively predict the location of real faults in DEFECTS4J.
Parameter Tuning

1. Greedy
2. Additional Greedy
3. Random
4. Constraint Solver
For real bug prediction data, the constraint solver is the best secondary objective.
For real bug prediction data, the constraint solver is the best secondary objective.

For perfect bug prediction data, most secondary objectives are able to almost perfectly prioritize test cases.
Research Objectives

1. Discover the best parameters for defect prediction in order to predict faulty classes as soon as possible.

2. Compare our approach against existing coverage-based approaches.

3. Compare our approach against existing history-based approaches.
Our Approach vs Coverage-Based

365 faults from DEFECTS4J

5 coverage-based strategies

Total 1,825 combinations of fault/strategy

Our approach is best for 1,165 combinations

Significantly outperforms 4 of the 5 strategies
In most cases, our approach requires the fewest test cases to find faults.
Research Objectives

1. Discover the best parameters for defect prediction in order to predict faulty classes as soon as possible.

2. Compare our approach against existing coverage-based approaches.

3. Compare our approach against existing history-based approaches.
Our Approach vs History-Based

82 faults from DEFECTS4J

4 history-based strategies

Total 328 combinations of fault/strategy

Our approach is best for 209 combinations

Significantly outperforms 3 of the 4 strategies
Our Approach vs History-Based
## Our Approach vs History-Based

<table>
<thead>
<tr>
<th>Project</th>
<th>Avg. Commits</th>
<th>% Occurrences</th>
<th>Num Failures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chart</td>
<td>24</td>
<td>73%</td>
<td>67%</td>
</tr>
<tr>
<td>Closure</td>
<td>178</td>
<td>82%</td>
<td>0%</td>
</tr>
<tr>
<td>Lang</td>
<td>159</td>
<td>87%</td>
<td>5%</td>
</tr>
<tr>
<td>Math</td>
<td>383</td>
<td>77%</td>
<td>6%</td>
</tr>
<tr>
<td>Mockito</td>
<td>105</td>
<td>65%</td>
<td>19%</td>
</tr>
<tr>
<td>Time</td>
<td>36</td>
<td>100%</td>
<td>0%</td>
</tr>
</tbody>
</table>

DPATerson1@sheffield.ac.uk
In software development, our goal is to minimize the impact of faults.

- If we know that a fault exists, we can use fault localization to pinpoint the code unit responsible.
- If we don’t know that a fault exists, we can use defect prediction to estimate which code units are likely to be faulty.

Why Do We Prioritize Test Cases?

- Regression testing can account for up to 80% of the total testing budget, and up to 50% of the cost of software maintenance.
- In some situations, it may not be possible to re-run all test cases on a system.
- By prioritizing test cases, we aim to ensure faults are detected in the smallest amount of time irrespective of program changes.

Our Approach vs History-Based

1. Discover the best parameters for defect prediction in order to predict faulty classes as soon as possible.
2. Compare our approach against existing coverage-based approaches.
3. Compare our approach against existing history-based approaches.

82 faults from DEFECTS4J

- 4 history-based strategies
- Total 328 combinations of fault/strategy
- Our approach is best for 209 combinations
- Significantly outperforms 3 of the 4 strategies

DPATERSON1@SHEFFIELD.AC.UK

**Tool:** https://github.com/kanonizo/kanonizo

**Data:** https://bitbucket.org/josecampos/history-based-test-prioritization-data
In order to cover $L_1$, we must select either $TC_1$ or $TC_3$

$$(TC_1 \lor TC_3) \land (TC_2 \lor TC_3) \land (TC_1)$$

Minimal set:

$$(TC_1 \land TC_2)$$

$$(TC_1 \land TC_3)$$
Statistical Tests

For each of our experiments, we calculated:

- The Mann-Whitney U Test \( p \)-value in order to calculate the likelihood that our results were observed as a result of chance

- The Vargha-Delaney effect size, to measure the magnitude of difference between results

- The ranking position of each configuration