Automatic Software Testing Via Mining Software Data

Wujie Zheng

Supervisor: Prof. Michael R. Lyu

Department of Computer Science & Engineering
The Chinese University of Hong Kong

August 17, 2011
Outline

• Introduction
• Part 1: Unit-Test Generation via Mining Relevant APIs
• Part 2: Test Selection via Mining Operational Models
• Part 3: Mining Test Oracles of Web Search Engines
• Conclusions
Introduction

- Software bugs annoy users or even cause great losses!

  Google harms your computer

  Destruction of NASA Mariner 1

- Software failures cost the US economy about $60 billion every year [NIST Report 2002]
Software Testing

• The primary way for removing bugs
• Three steps
  – Generate test inputs
  – Run test inputs
  – Inspect test results (check actual outputs or properties against test oracles)
Software Testing

- A system test

![Test Inputs](image1.png)

![Test Oracles](image2.png)
Software Testing

• A unit test

```java
public void testSearch() {
    // test input
    Stack var0 = new Stack();
    String var1 = "hi!";
    var0.push((Object)var1);
    int var2 = var0.search(var1);

    // test oracle
    assertTrue(var2==1);
}
```
Software Testing

• Manual software testing
  – Difficult to create a good set of test inputs
    • Software systems become large-sized and complex
  – Tedious to inspect a large set of test results
Automatic Software Testing

- Test input generation
  - Random testing, combinatorial testing, model-based testing, grammar-based testing
- Test result inspection
  - Model-based testing

Specification

Test Input Generation

Test Result Inspection

Rules & Regulations
Automatic Software Testing

• Specification: a complete description of the behavior of a software to be developed
  – Constraints on test inputs
    • socket->bind->listen->accept
    • For a method $f(int x, int y), x>0, y>0$
  – Constraints on program states
    • From state $s$ and action $x$, the next state should be $t$.
    • There should be no memory errors, e.g., double free
  – Constraints on test outputs
    • For a method $sort(x)$, the output is sorted
Challenges

• The specification is often unavailable or incomplete

Specification → Test Input Generation → Test Result Inspection
My Thesis

• Mining specifications from software data to guide test input generation and test result inspection
My Thesis

- Part 1: unit-test generation via mining relevant APIs
  - A unit-test is a method call sequence

Source Code | Relevant APIs | Test Input Generation
---|---|---
```java
public class HelloWorld {
    public static void main(String[] args)
    {}
System.out.println("Hello World!");
```

- Contribution
  - Reduce the search space of possible method call sequences by exploiting the relevance of methods
My Thesis

- Part 2: test selection via mining operational models
  - Control rules, data rules

**Execution Traces**   **Operational Models**   **Test Result Inspection**

\[ Br1 \rightarrow Br2 \]

- Contribution
  - Propose two kinds of operational models that can detect failing tests effectively and can be mined efficiently
My Thesis

• Part 3: mining test oracles of Web search engines

Program Outputs

Output Rules and Classification Models

Test Result Inspection

\[ P1 \Rightarrow P2 \]

• Contribution
  – Apply test selection techniques to Web Search Engines
  – Select failing tests by exploiting application-level knowledge
# My Thesis

## Overview

<table>
<thead>
<tr>
<th>Part</th>
<th>Software Data</th>
<th>Mined/Learned Specifications</th>
<th>Testing Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part 1</td>
<td>Source Code</td>
<td>Relevant APIs (Specifications about Program Inputs)</td>
<td>Test Input Generation</td>
</tr>
<tr>
<td>Part 2</td>
<td>Execution Traces</td>
<td>Operational Models (Specifications about Program States)</td>
<td>Test Result Inspection (Test Selection)</td>
</tr>
<tr>
<td>Part 3</td>
<td>Program Inputs and Outputs</td>
<td>Output Rules (Specifications about Program Outputs)</td>
<td>Test Result Inspection (Test Selection)</td>
</tr>
</tbody>
</table>
Part 1: Unit-Test Generation via Mining Relevant APIs
Problem

• Given a set of methods under test (MUTs), generate inputs (method-call sequences) that explore different behaviors of each method.
Existing Approaches

• Random
  – Select parameters of methods randomly

  A. $f(B)$ means $f$ is a method class A and it has an argument of class B

```java
Stack var0 = new Stack();
String var1 = "hi!";
Stack.push(Object)

Stack var0 = new Stack();
String var1 = "hi!";
var0.push((Object)var1);
Stack.search(Object)

Stack var0 = new Stack();
String var1 = "hi!";
var0.push((Object)var1);
int var2 = var0.search(var1);
```
Existing Approaches

• Feedback-directed generation
  – Discard sequences whose execution throw exceptions

• Adaptive random generation
  – Select sequences that are most different from previous selected ones

• *They do not consider how the specific method under test is implemented*
The Idea

- A method cannot affect the execution of the method under test (MUT) if it does not mutate an input’s fields accessed by the MUT.

```java
Stack var0 = new Stack();
String var1 = "hi!";
var0.push((Object)var1);
```

- the `size()` method has no effect because it does not change any fields that `search()` access.
Example

- `openDatabase()` calls `setupDatabase()` calls `getAllowCreate()` accesses `allowCreate`
- `setAllowCreate()` accesses `allowCreate`
- To test `openDatabase()`, for sequences of `DatabaseConfig` objects, we prefer the sequences that call `setAllowCreate()`
Our Approach

• Mining relevant APIs
  – Use Eclipse JDT Compiler to analyze the object fields accessed by each method
    • Each method is represented as an itemset of the object fields that it accesses
      openDatabase() : Environment.envImpl, DatabaseConfig.allowCreate, ...
      setAllowCreate() : DatabaseConfig.allowCreate
  – Find relevant APIs that access the same object fields
    • openDatabase() is relevant to setAllowCreate()
Our Approach

• RecGen: recommendation-based test generation
  – For each parameter, recommend a method call sequence from the existing sequences
    • Assign more weights to short sequences with more relevant APIs

\[ A.f(B) \]
Experiments

- Three subjects
  - Berkeley DB Java Edition (BDB)
  - Java Data Structure Library (JDSL)
  - Science Computing Library (JScience)
- Compared with three representative tools
  - JCrasher
  - Randoop
  - ARTGen
- Metrics
  - Code Coverage
Experiments

Table 3.2: Statement coverage (%) on Berkeley DB (LOC: lines of code)

<table>
<thead>
<tr>
<th>Package</th>
<th>#LOC</th>
<th>JCrasher</th>
<th>Randoop</th>
<th>ARTGen</th>
<th>RecGen</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.sleepycat.je</td>
<td>4755</td>
<td>9.8</td>
<td>36.6</td>
<td>32.5</td>
<td>44.3</td>
</tr>
<tr>
<td>com.sleepycat.je.cleaner</td>
<td>2850</td>
<td>1.6</td>
<td>30.6</td>
<td>8.5</td>
<td>52.8</td>
</tr>
<tr>
<td>com.sleepycat.je.config</td>
<td>764</td>
<td>89.1</td>
<td>95.9</td>
<td>95.5</td>
<td>95.2</td>
</tr>
<tr>
<td>com.sleepycat.je.dbi</td>
<td>4401</td>
<td>10.4</td>
<td>40.0</td>
<td>27.9</td>
<td>53.4</td>
</tr>
<tr>
<td>com.sleepycat.je.evictor</td>
<td>456</td>
<td>0.0</td>
<td>11.2</td>
<td>0.2</td>
<td>8.6</td>
</tr>
<tr>
<td>com.sleepycat.je.incomp</td>
<td>318</td>
<td>0.3</td>
<td>23.3</td>
<td>0.3</td>
<td>16.0</td>
</tr>
<tr>
<td>com.sleepycat.je.jca.ra</td>
<td>278</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>com.sleepycat.je.jmx</td>
<td>441</td>
<td>49.2</td>
<td>58.3</td>
<td>57.8</td>
<td>64.6</td>
</tr>
<tr>
<td>com.sleepycat.je.latch</td>
<td>215</td>
<td>27.0</td>
<td>74.9</td>
<td>67.4</td>
<td>76.7</td>
</tr>
<tr>
<td>com.sleepycat.je.log</td>
<td>3789</td>
<td>9.6</td>
<td>36.3</td>
<td>15.1</td>
<td>49.6</td>
</tr>
<tr>
<td>com.sleepycat.je.log.entry</td>
<td>366</td>
<td>15.0</td>
<td>47.5</td>
<td>29.8</td>
<td>65.6</td>
</tr>
<tr>
<td>com.sleepycat.je.recovery</td>
<td>1954</td>
<td>7.0</td>
<td>33.9</td>
<td>7.8</td>
<td>34.4</td>
</tr>
<tr>
<td>com.sleepycat.je.tree</td>
<td>4398</td>
<td>9.3</td>
<td>34.8</td>
<td>22.0</td>
<td>47.4</td>
</tr>
<tr>
<td>com.sleepycat.je.txn</td>
<td>2608</td>
<td>6.6</td>
<td>37.6</td>
<td>22.1</td>
<td>52.5</td>
</tr>
<tr>
<td>com.sleepycat.je.util</td>
<td>1564</td>
<td>5.9</td>
<td>22.9</td>
<td>22.5</td>
<td>34.6</td>
</tr>
<tr>
<td>com.sleepycat.je.utilint</td>
<td>678</td>
<td>19.3</td>
<td>63.7</td>
<td>50.7</td>
<td>64.5</td>
</tr>
<tr>
<td>Total</td>
<td>29835</td>
<td>11.0</td>
<td>37.4</td>
<td>24.2</td>
<td>48.4</td>
</tr>
</tbody>
</table>

- With feedback is better
- With sequence recommendation is better
Experiments

With feedback is better
With sequence recommendation is better

Table 3.3: Statement coverage (%) on JDSL (LOC: lines of code)

<table>
<thead>
<tr>
<th>Package</th>
<th>LOC</th>
<th>JCrasher</th>
<th>Randoo</th>
<th>ARTGen</th>
<th>RecGen</th>
</tr>
</thead>
<tbody>
<tr>
<td>jdsl.core.algo.sorts</td>
<td>91</td>
<td>24.2</td>
<td></td>
<td>48.4</td>
<td></td>
</tr>
<tr>
<td>jdsl.core.algo.traversals</td>
<td>26</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>jdsl.core.api</td>
<td>62</td>
<td>69.4</td>
<td>93.5</td>
<td>90.3</td>
<td>25.8</td>
</tr>
<tr>
<td>jdsl.core.ref</td>
<td>2497</td>
<td>26.1</td>
<td>49.4</td>
<td>39.4</td>
<td>67.4</td>
</tr>
<tr>
<td>jdsl.core.util</td>
<td>60</td>
<td>30.0</td>
<td>6.7</td>
<td>6.7</td>
<td>1.7</td>
</tr>
<tr>
<td>jdsl.graph.algo</td>
<td>602</td>
<td>8.7</td>
<td>40.0</td>
<td>20.1</td>
<td>41.4</td>
</tr>
<tr>
<td>jdsl.graph.api</td>
<td>46</td>
<td>47.8</td>
<td>89.1</td>
<td>82.6</td>
<td>37.0</td>
</tr>
<tr>
<td>jdsl.graph.ref</td>
<td>541</td>
<td>15.7</td>
<td>29.6</td>
<td>25.9</td>
<td>51.9</td>
</tr>
<tr>
<td>Total</td>
<td>3925</td>
<td>23.2</td>
<td>45.5</td>
<td>35.2</td>
<td>58.9</td>
</tr>
</tbody>
</table>

Table 3.4: Statement coverage (%) on JScience (LOC: lines of code; GEO.COOR: geography.coordinates, MATH: mathematics)

<table>
<thead>
<tr>
<th>Package</th>
<th>LOC</th>
<th>JCrasher</th>
<th>Randoo</th>
<th>ARTGen</th>
<th>RecGen</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.jscience</td>
<td>396</td>
<td>3.0</td>
<td>4.5</td>
<td>4.8</td>
<td>4.8</td>
</tr>
<tr>
<td>org.jscience.economics.money</td>
<td>55</td>
<td>43.6</td>
<td>87.3</td>
<td>85.5</td>
<td>96.4</td>
</tr>
<tr>
<td>org.jscience.GEO.COOR</td>
<td>667</td>
<td>17.4</td>
<td>61.9</td>
<td>60.9</td>
<td>21.9</td>
</tr>
<tr>
<td>org.jscience.GEO.COOR.crs</td>
<td>198</td>
<td>52.5</td>
<td>64.1</td>
<td>61.6</td>
<td>61.1</td>
</tr>
<tr>
<td>org.jscience.MATH.function</td>
<td>692</td>
<td>32.8</td>
<td>32.7</td>
<td>37.3</td>
<td>39.6</td>
</tr>
<tr>
<td>org.jscience.MATH.number</td>
<td>1683</td>
<td>68.1</td>
<td>83.1</td>
<td>79.3</td>
<td>86.1</td>
</tr>
<tr>
<td>org.jscience.MATH.vector</td>
<td>1551</td>
<td>22.0</td>
<td>39.8</td>
<td>46.1</td>
<td>82.8</td>
</tr>
<tr>
<td>org.jscience.physics.amount</td>
<td>614</td>
<td>36.5</td>
<td>67.4</td>
<td>57.8</td>
<td>70.5</td>
</tr>
<tr>
<td>org.jscience.physics.model</td>
<td>60</td>
<td>58.3</td>
<td>96.7</td>
<td>96.7</td>
<td>100</td>
</tr>
<tr>
<td>Total</td>
<td>5916</td>
<td>37.7</td>
<td>56.1</td>
<td>56.0</td>
<td>64.9</td>
</tr>
</tbody>
</table>
Summary of Part 1

• Problem
  – Unit-Test input generation (method call sequence)

• Our approach
  – Mine relevant APIs that access common fields
  – For each parameter, select short method call sequences that have more relevant APIs

• Contribution
  – Reduce the search space of possible method call sequences by exploiting the relevance of methods
Part 2: Test Selection via Mining Operational Models
Problem

• Given a large set of test results, find the failing tests from them
  – Without executable test oracles
  – Manual test result inspection could be labor-intensive
Solution

• Test selection for result inspection
  – Select a small subset of tests that are likely to reveal faults

Hey! Check only these tests!
Existing Approaches

- Code coverage based selection
- Clustering based selection
- Operational model based selection
Code Coverage Based Selection

- Select a new test if it increases some coverage criteria, otherwise discard it
  - Method, line, branch coverage

<table>
<thead>
<tr>
<th></th>
<th>Br1</th>
<th>Br2</th>
<th>Br3</th>
<th>Br4</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>Test2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>Test3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>Test4</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>...</td>
</tr>
</tbody>
</table>

Test1, Test3
Clustering Based Selection

• Use hierarchical clustering of execution profiles and perform one-per-cluster sampling
  – Failing tests are often grouped into small clusters
Operational Model Based Selection

• Mine *invariants* from *passing* tests (Daikon, DIDUCE)

```
i, s := 0, 0;
do i ≠ n →
i, s := i + 1, s + b[i]
od
```

Precondition: \( n ≥ 0 \)
Postcondition: \( s = \sum_{j : 0 ≤ j < n} b[j] \)
Loop invariant: \( 0 ≤ i ≤ n \) and \( s = \sum_{j : 0 ≤ j < i} b[j] \)

• Select tests that violate the existing invariants (Jov, Eclat, DIDUCE)
Our Approach

• Mine **common** operational models from **unverified** tests
  – The models are often but not always true in the observed traces
Our Approach

• Why is it difficult?
  – The previous templates of operational models generate too much candidates
  – Examine all the candidates at runtime may incur high runtime overhead
    • For passing tests, we can discard any violation
    • For unverified tests, we cannot!
Our Approach

• Effective mining of operational models
  – Collect simple traces at runtime
    • Branch coverage
    • Data value bounds
  – Generate and evaluate potential operational models after running all the tests
    • Control rules: implication relationships between branches
    • Data rules: implicit data value distributions
Common Operational Models

- Control rules: implication relationships between branches

|       | Br1 | Br2 | Br3 | Br4 | ...
|-------|-----|-----|-----|-----|-----
| Test1 | 1   | 0   | 1   | 1   | ... |
| Test2 | 1   | 0   | 1   | 1   | ... |
| Test3 | 0   | 1   | 0   | 0   | ... |
| Test4 | 1   | 0   | 1   | 0   | ... |

$Br1 \Rightarrow \neg Br2$

$Br1 \Rightarrow Br3$
Data rules: implicit data value distributions

<table>
<thead>
<tr>
<th></th>
<th>min(Var1)</th>
<th>max(Var1)</th>
<th>min(Var2)</th>
<th>max(Var2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test1</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>Test2</td>
<td>0</td>
<td>32</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>Test3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Test4</td>
<td>0</td>
<td>23</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

The distribution of $\text{max(Var1)}$

Too large or too small values are suspicious
Test Selection

- Select tests for result inspection
  - Sort the mined rules in the descending order of confidence
  - Select tests that violate the rules from the top to bottom
Experiments

- **Subject programs**
  - *Siemens* suite: 130 faulty versions of 7 programs
  - *grep* program: 20 faulty versions

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>Test Cases</th>
<th>Faulty Versions</th>
<th>Failed Tests (Avg.)</th>
<th>Program Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>print_tokens</td>
<td>539</td>
<td>4130</td>
<td>7</td>
<td>69</td>
<td>lexical analyzer</td>
</tr>
<tr>
<td>print_tokens2</td>
<td>489</td>
<td>4115</td>
<td>10</td>
<td>224</td>
<td>lexical analyzer</td>
</tr>
<tr>
<td>replace</td>
<td>507</td>
<td>5542</td>
<td>31</td>
<td>106</td>
<td>pattern replacement</td>
</tr>
<tr>
<td>schedule</td>
<td>397</td>
<td>2650</td>
<td>9</td>
<td>88</td>
<td>priority scheduler</td>
</tr>
<tr>
<td>schedule2</td>
<td>299</td>
<td>2710</td>
<td>9</td>
<td>33</td>
<td>priority scheduler</td>
</tr>
<tr>
<td>teas</td>
<td>174</td>
<td>1608</td>
<td>41</td>
<td>39</td>
<td>altitude separation</td>
</tr>
<tr>
<td>tot_info</td>
<td>398</td>
<td>1052</td>
<td>23</td>
<td>83</td>
<td>information measure</td>
</tr>
<tr>
<td>Siemens suite</td>
<td>404</td>
<td>3115</td>
<td>130</td>
<td>92</td>
<td>–</td>
</tr>
<tr>
<td>grep</td>
<td>13358</td>
<td>809</td>
<td>20</td>
<td>177</td>
<td>pattern matching</td>
</tr>
</tbody>
</table>
Experiments

• Effectiveness
  – The number of the selected tests
  – The percentage of revealed faults

![Graph a) the Siemens suite](image)

![Graph b) the grep program](image)
Experiments

- Our approach is more effective

<table>
<thead>
<tr>
<th>Program</th>
<th>Manual Test Suite</th>
<th>Our Approach</th>
<th>Random</th>
<th>Coverage(k=1)</th>
<th>Clustering</th>
<th>OD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#T</td>
<td>#F</td>
<td>#T</td>
<td>%F</td>
<td>#T</td>
<td>%F</td>
</tr>
<tr>
<td>print_tokens</td>
<td>4130</td>
<td>7</td>
<td>25</td>
<td>89</td>
<td>37</td>
<td>39</td>
</tr>
<tr>
<td>print_tokens2</td>
<td>4115</td>
<td>10</td>
<td>41</td>
<td>100</td>
<td>37</td>
<td>78</td>
</tr>
<tr>
<td>replace</td>
<td>5542</td>
<td>31</td>
<td>75</td>
<td>80</td>
<td>37</td>
<td>45</td>
</tr>
<tr>
<td>schedule</td>
<td>2650</td>
<td>9</td>
<td>31</td>
<td>86</td>
<td>37</td>
<td>48</td>
</tr>
<tr>
<td>schedule2</td>
<td>2710</td>
<td>9</td>
<td>32</td>
<td>62</td>
<td>37</td>
<td>34</td>
</tr>
<tr>
<td>tcas</td>
<td>1608</td>
<td>41</td>
<td>26</td>
<td>74</td>
<td>37</td>
<td>46</td>
</tr>
<tr>
<td>tot_info</td>
<td>1052</td>
<td>23</td>
<td>29</td>
<td>84</td>
<td>37</td>
<td>75</td>
</tr>
<tr>
<td>Siemens</td>
<td>3115</td>
<td>130</td>
<td>37</td>
<td>82</td>
<td>37</td>
<td>52</td>
</tr>
<tr>
<td>grep</td>
<td>809</td>
<td>20</td>
<td>218</td>
<td>98</td>
<td>219</td>
<td>90</td>
</tr>
</tbody>
</table>
Control Rules vs. Data Rules

- Control rules reveal more faults

<table>
<thead>
<tr>
<th>Program</th>
<th>Original Test Suite</th>
<th>Our Approach</th>
<th>Control Rules</th>
<th>Data Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Tests</td>
<td>#Faults</td>
<td>#Tests</td>
<td>%Faults</td>
</tr>
<tr>
<td>print_tokens</td>
<td>4130</td>
<td>7</td>
<td>25</td>
<td>89</td>
</tr>
<tr>
<td>print_tokens2</td>
<td>4115</td>
<td>10</td>
<td>41</td>
<td>100</td>
</tr>
<tr>
<td>replace</td>
<td>5542</td>
<td>31</td>
<td>75</td>
<td>80</td>
</tr>
<tr>
<td>schedule</td>
<td>2650</td>
<td>9</td>
<td>31</td>
<td>86</td>
</tr>
<tr>
<td>schedule2</td>
<td>2710</td>
<td>9</td>
<td>32</td>
<td>62</td>
</tr>
<tr>
<td>tcas</td>
<td>1608</td>
<td>41</td>
<td>26</td>
<td>74</td>
</tr>
<tr>
<td>tot_info</td>
<td>1052</td>
<td>23</td>
<td>29</td>
<td>84</td>
</tr>
<tr>
<td>Siemens suite</td>
<td>3115</td>
<td>130</td>
<td>37</td>
<td>82</td>
</tr>
<tr>
<td>grep</td>
<td>809</td>
<td>20</td>
<td>218</td>
<td>98</td>
</tr>
</tbody>
</table>
Random Test Suites

- Our approach works well on automatically generated test suites

<table>
<thead>
<tr>
<th>Program</th>
<th>Automated Test Suite</th>
<th>Our Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Tests</td>
<td>#Faults</td>
</tr>
<tr>
<td>print_tokens</td>
<td>1000</td>
<td>2</td>
</tr>
<tr>
<td>print_tokens2</td>
<td>1000</td>
<td>7</td>
</tr>
<tr>
<td>replace</td>
<td>1000</td>
<td>10</td>
</tr>
<tr>
<td>schedule</td>
<td>1000</td>
<td>3</td>
</tr>
<tr>
<td>schedule2</td>
<td>1000</td>
<td>4</td>
</tr>
<tr>
<td>tcas</td>
<td>1000</td>
<td>23</td>
</tr>
<tr>
<td>tot_info</td>
<td>1000</td>
<td>15</td>
</tr>
<tr>
<td>Siemens suite</td>
<td>1000</td>
<td>64</td>
</tr>
<tr>
<td>grep</td>
<td>1000</td>
<td>12</td>
</tr>
</tbody>
</table>
Summary of Part 2

• Problem
  – Test selection for result inspection

• Our approach
  – Mining common operational models (control rules, data rules) from execution traces of unverified tests

• Contribution
  – Propose two kinds of operational models that can detect failing tests effectively and can be mined efficiently
Part 3: Mining Test Oracles of Web Search Engines
• Find defects of *Web search engines* with respect to *retrieval effectiveness*.
  – Web search engines have major impact in people’s everyday life.
  – Retrieval effectiveness is one of the major concerns of search engine users
    • How well a search engine satisfies users’ information need
    • Relevance, authority, and freshness
Background

• An example
  – Declaration from the PuTTY Website for Google’s search result change

  2010-05-17 Google listing confusion

  Several users have pointed out to us recently that the top Google hit for "putty" is now not the official PuTTY site but a mirror that used to be listed on our Mirrors page.

  The official PuTTY web page is still where it has always been:

  http://www.chiark.greenend.org.uk/~sgtatham/putty/

  – This declaration suggests that Google’s search results for “putty” at some time may not be satisfactory and may cause confusions of the users.
Problem

• Given a large set of search results, find the failing tests from them
  – Test oracles: relevance judgments
Problem

• It is labor-intensive to collect the relevance judgments of search results
  – For a large number of queries

• Previous relevance judgments may not be reusable
  – The desired search results may change over time
Existing Approaches

• The *pooling* process
  – Different information retrieval systems submit the top $K$ results per query
  – The assessors judge for relevance manually

• The idea
  – Inspect *parts* of search results for all queries

• Limitations
  – Too costly, hardly reusable
Existing Approaches

• Click through data as implicit feedback
  – Clicked results are relevant

• The idea
  – Let users inspect all search results of all queries

• Limitations
  – Position bias, summary bias
    • E.g., cannot find relevant pages that are not in the search results
Our Approach

• Test selection
  – Inspect parts of search results for some queries by mining search results of all queries
  – Exploit application-level knowledge
    • Execution traces may not help
  – Utilize the existing labels of testers
    • The process needs to be repeated
Our Approach

- Mining and learning output rules
Mining Output Rules

• Query items
  – Query words, query types, query length, etc.

• Search result items
  – Domain, domain’s Alexa rank, etc.

• Query-result matching items
  – Whether the domain name has the query, whether the title has the query, etc.

• Search engine items
  – Search engine names
Example Itemsets

- SE: bing, Q: day after day, QW: day, QW: after, ManyWords, CommonQ, top10: en.wikipedia.org, top1: en.wikipedia.org, top10: dayafterday.org, ..., SOMEGE100K, SOMELE1K
Mining Association Rules

• Mining frequent itemsets with length constraint
  – An itemset is frequent if its support is larger than the min_support
    
    \{SE:bing, top10:en.wikipedia.org\}

• Generating rules with only one item in the right hand side
  – For each item \( x_i \) in \( Y \), generate a rule \( Y-x_i \Rightarrow x_i \)
    
    SE:bing=>top10:en.wikipedia.org
Learning Classification Models

- Feature Vectors
  - Can describe more general types of properties

<table>
<thead>
<tr>
<th>Search Result List 1</th>
<th>wordLength</th>
<th>queryType</th>
<th>max(domainRank)</th>
<th>google.com</th>
<th>facebook.com</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>common</td>
<td>900</td>
<td>1</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>Search Result List 2</td>
<td>3</td>
<td>hot</td>
<td>100000</td>
<td>0</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>Search Result List 3</td>
<td>1</td>
<td>hot</td>
<td>9782</td>
<td>1</td>
<td>1</td>
<td>...</td>
</tr>
</tbody>
</table>
Learning Classification Models

• Learn classification models of the failing tests based on the training data

• Given new search results, use the learned model to predict whether they fail.
Experiments

• Search engines
  – Google
  – Bing
  These two search engines, together with many other search engines powered by them (e.g., Yahoo! Search is now powered by Bing and AOL Search is powered by Google), possess more than 90 percent search market share in U.S.
Experiments

• Queries
  – Common queries
    • Queries in KDDCUP 2005, 800 queries
  – Hot queries
    • 3432 unique hot queries from Google Trends and Yahoo! Buzz from November 25, 2010 to April 21, 2011
Experiments

• Search results
  – Use the Web services of Google and Bing to collect the top 10 search results of each query from December 25, 2010 to April 21, 2011
  – 390797 ranked lists of search results (each list contains the top 10 search results)
The Mined Rules

• Mining from one search engine' results in one day
  – Google's search results on Dec. 25, 2010
  – minsup = 20, minconf = 0.95, and maxL = 3

1. top10:starpulse.com,HotQ, => top10:imdb.com, : 22/22=1.0
2. top10:starpulse.com,TwoWords, => top10:imdb.com, : 22/23=0.96
The Mined Rules

• Mining from multiple search engines' results in one day
  – Google and Bing's search results on Dec. 25, 2010
  – minsup = 20, minconf = 0.95, and maxL = 3

  6. top10:starpulse.com, HotQ, => top10:imdb.com, : 24/24=1.0
  8. TwoWords, top10:tvguide.com, => top10:imdb.com, : 23/24=0.96
  9. top10:absoluteastronomy.com, => SE:bing, : 63/63=1.0
  10. top10:thirdage.com, => SE:bing, : 40/40=1.0
  11. TwoWords, top10:youtube.com, => SE:google, : 137/143=0.95
  12. OneWord, top10:twitter.com, => SE:google, : 28/29=0.97

  – Rules 9-12 show the different opinions of search engines to certain Websites
The Mined Rules

- Mining from one search engine's results in multiple days
  - Google's search results from December 25, 2010 to March 31, 2011.
  - \text{minsup} = 20, \text{minconf} = 0.95, \text{and maxL} = 2

- Rules 13-18 show the rules about the top 1 results for the queries:
  
  13. \text{Q:hulu, } \Rightarrow \text{top1:hulu.com, : 91/91=1.0}
  14. \text{Q:facebook, } \Rightarrow \text{top1:facebook.com, : 91/91=1.0}
  15. \text{Q:youtube, } \Rightarrow \text{top1:youtube.com, : 91/91=1.0}
  16. \text{Q:rosenbluth, } \Rightarrow \text{top1:rvacations.com, : 91/91=1.0}
  17. \text{Q:espn picks, } \Rightarrow \text{top1:espn.go.com, : 91/91=1.0}
  18. \text{Q:stock futures, } \Rightarrow \text{top1:bloomberg.com, : 91/91=1.0}
Example Violations

• Search results of Bing on April 1st, 2011 violate the following rule

Q: where to login to john carroll university email, =>
top1: mirapoint.jcu.edu, : 172/180=0.96

• The actual result of Bing
  http://www.jcu.edu/index.php
  points to the homepage of the John Carroll University, not easy to get the answer of the query
Learning Classification Models

• Conduct experiments with the following classes
  – Unexpected top 1 change
    • the other search engines oppose the change (they returned the same top 1 result and do not change)
  – Normal top 1 change
    • the other search engines do not oppose the change

• Task
  – Given a top 1 change of the search engine under test, predict whether it is an unexpected change
Learning Classification Models

• Data
  – Training data: December 26, 2010 to March 31, 2011
  – Testing data: April 1, 2011 to April 22, 2011

• Results of predicting unexpected top 1 changes

<table>
<thead>
<tr>
<th>Models</th>
<th>Data</th>
<th>Abnormal Data</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decision Tree</td>
<td>3429</td>
<td>921</td>
<td>0.72</td>
<td>0.47</td>
<td>0.42</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>3429</td>
<td>921</td>
<td>0.66</td>
<td>0.36</td>
<td>0.38</td>
</tr>
</tbody>
</table>

– Decision Tree is more accurate, but Naive Bayes is faster
Summary of Part 3

• Problem
  – Search engine testing

• Our Approach
  – Mine and learn output rules to find suspicious search results automatically

• Contribution
  – Apply test selection techniques to Web Search Engines
  – Select failing tests by exploiting application-level knowledge
Conclusions
Conclusions

• Mining specifications from software data to guide test input generation and test result inspection
  – Part 1: unit-test generation via mining relevant APIs
    • Reduce the search space of possible method call sequences by exploiting the relevance of methods
Conclusions

– Part 2: test selection via mining operational models
  • Propose two kinds of operational models that can detect failing tests effectively and can be mined efficiently

– Part 3: mining test oracles of web search engines
  • Apply test selection techniques to Web Search Engines
  • Select failing tests by exploiting application-level knowledge
Publications


Q&A

Thanks!