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

Test Oracles

testcases - : fx 5.0 smoketest - functionality

Submit All Results

1: [awesombar] address field and go button »

Steps to Perform:

1. Load a random page in the currently selected tab.
2. Type "www.google.com" into the location bar.
3. Click the Go button (it is right facing triangle) on the right side of the location bar.

Expected Results:

1. With step 2, when you type something in the loca Go button should appear.
2. Clicking the Go button should load Google in the c

This test is covered by Mozmill:
testawesomebar/testgobutton.js</p>

Result:

- Not Run
- Pass
- Fail
- Test unclear/broken

Notes/Comments (optional):

Associated Bug #s:

(bug #,bug #,...)

Software Testing

- A unit test

Test Inputs

```
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);
}
```

Test Oracles

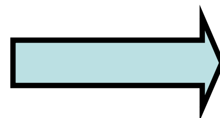
Software Testing

- 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

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

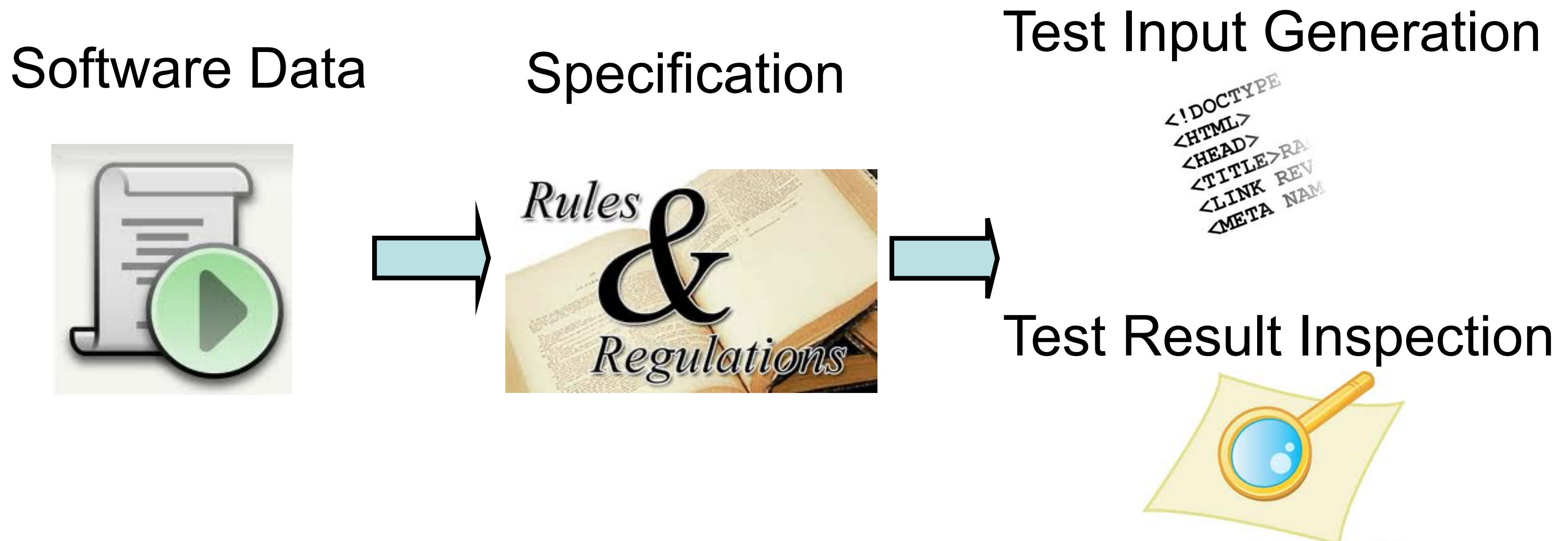
Test Input Generation

<!DOCTYPE
<HTML>
<HEAD>
<TITLE>
<LINK
<META

Test Result Inspection

My Thesis

- Mining specifications from software data to guide test input generation and test result inspection



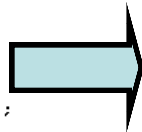
My Thesis

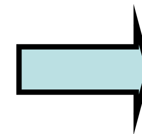
- Part 1: unit-test generation via mining relevant APIs
  - A unit-test is a method call sequence

Source Code

```
public class HelloWorld {
    /*
     * a demo
     */
    public static void main(String[] args)
    {
        System.out.println("Hello World!");
    }
}
```

Relevant APIs





Test Input Generation

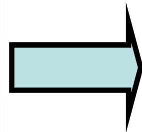
<!DOCTYPE
<HTML>
<HEAD>
<TITLE>
<LINK
<META

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

My Thesis

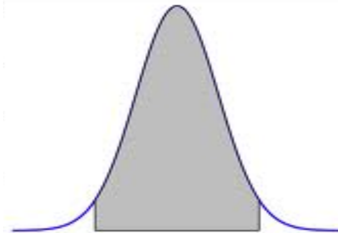
- Part 2: test selection via mining operational models
  - Control rules, data rules

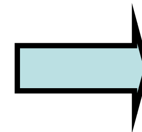
Execution Traces



Operational Models

Br1 => Br2





Test Result Inspection

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

My Thesis

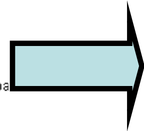
- Part 3: mining test oracles of Web search engines

Program Outputs

Software testing - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/Software_testing - Cached
Software testing is an investigation conducted to provide stakeholders
the quality of the product or service under test. ...
Graphical user interface testing - Portal - Category:Software testing

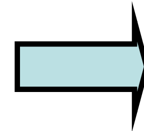
Software Testing
www.ece.cmu.edu/~koopman/des_s99/sw_testing/ - Cached
Software testing is any activity aimed at evaluating an attribute or capa
system and determining that it meets its required results. ...

Software Testing Tutorials
www.guru99.com/software-testing.html - Cached
Fundamentals of **Software** (Manual)**Testing** explained used Real Life S
Tutorials that cover entire ISTQB and CSTE syllabus.



Output Rules and Classification Models

$P1 \Rightarrow P2$



Test Result Inspection

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

My Thesis

- Overview

	Software Data	Mined/Learned Specifications	Testing Tasks
Part 1	Source Code	Relevant APIs (Specifications about Program Inputs)	Test Input Generation
Part 2	Execution Traces	Operational Models (Specifications about Program States)	Test Result Inspection (Test Selection)
Part 3	Program Inputs and Outputs	Output Rules (Specifications about Program Outputs)	Test Result Inspection (Test Selection)

Part 1: Unit-Test Generation via Mining Relevant APIs

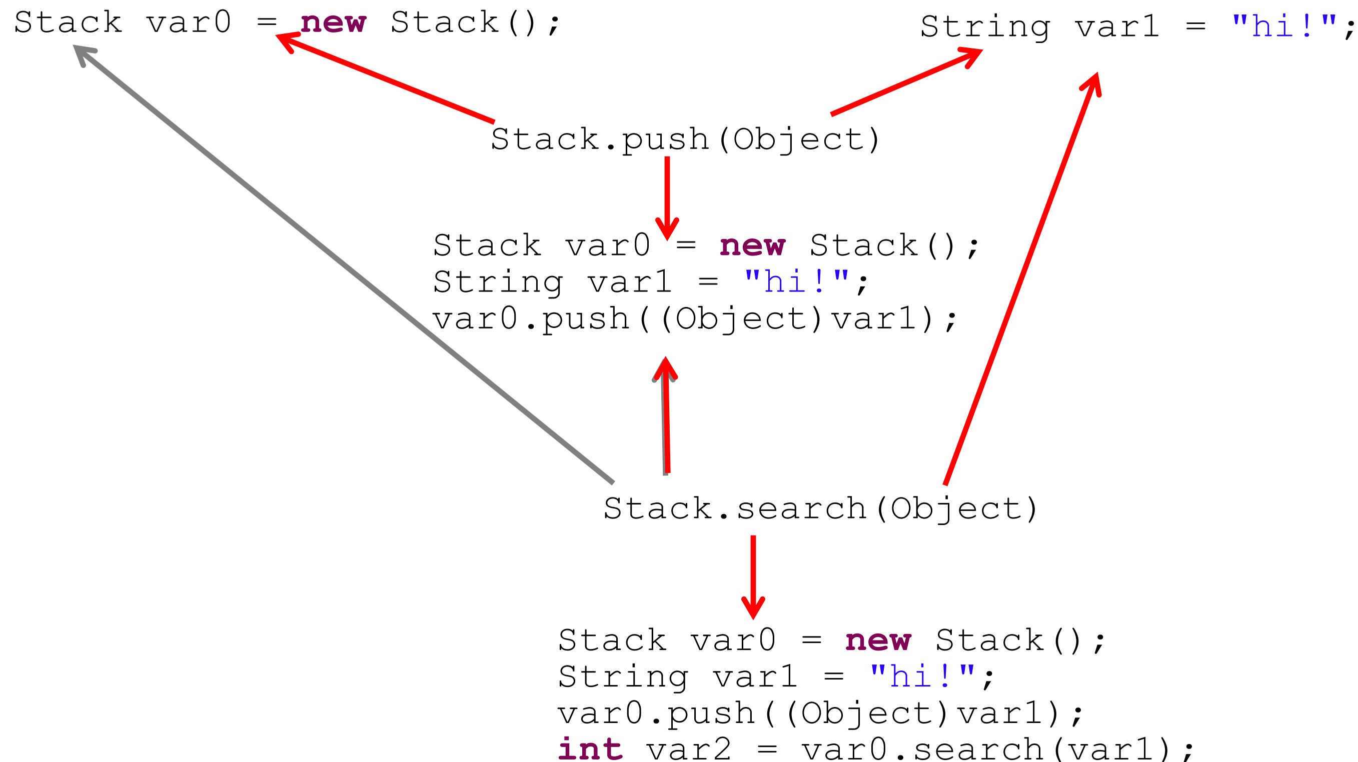
Problem

- 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

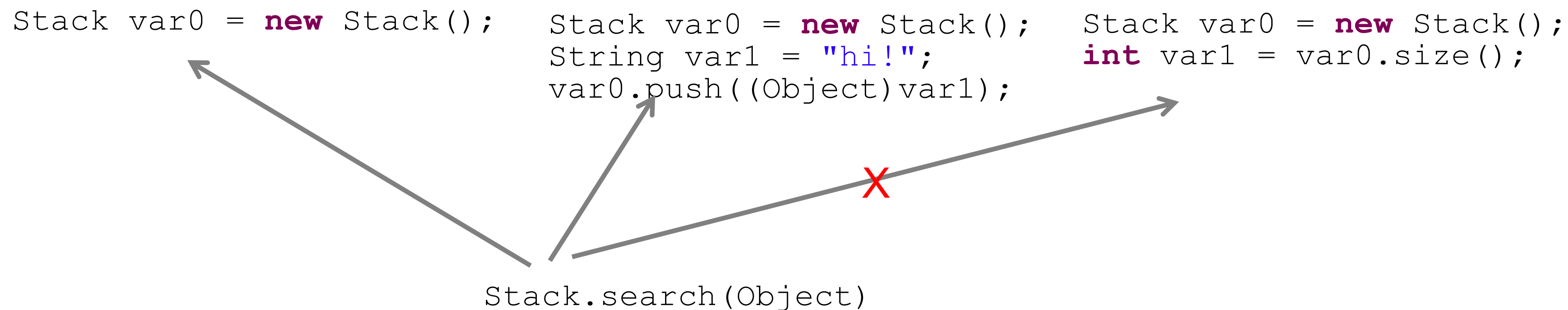


Existing Approaches

- 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.



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

Example

```
// Environment.java
public synchronized Database openDatabase(
                        Transaction txn,
                        String databaseName,
                        DatabaseConfig dbConfig)
throws DatabaseException {
    checkHandleIsValid();
    checkEnv();
    try {
        if (dbConfig == null) {
            dbConfig = DatabaseConfig.DEFAULT;
        }
        Database db = new Database(this);
        setupDatabase(txn, db, databaseName,
                        dbConfig,
                        false,
                        false,
                        envImpl.isReplicated());
        return db;
    } catch (Error E) {
        envImpl.invalidate(E);
        throw E;
    }
}
```

```
private void setupDatabase(..., DatabaseConfig
                              dbConfig, ...)
throws DatabaseException {
    ...
    if (databaseExists) {
        ...
    } else {
        ...
        /* No database.
           Create if we're allowed to. */
        if (dbConfig.getAllowCreate()) {
            ...
        }
    }
    ...
}

// DatabaseConfig.java
public boolean getAllowCreate() {
    return allowCreate;
}

public void setAllowCreate(boolean allowCreate) {
    this.allowCreate = allowCreate;
}
```

- openDatabase() calls setupDatabase() calls getAllowCreate() accesses allowCreate
- setAllowCreate() accesses allowCreate
- To test openDatabse(), 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

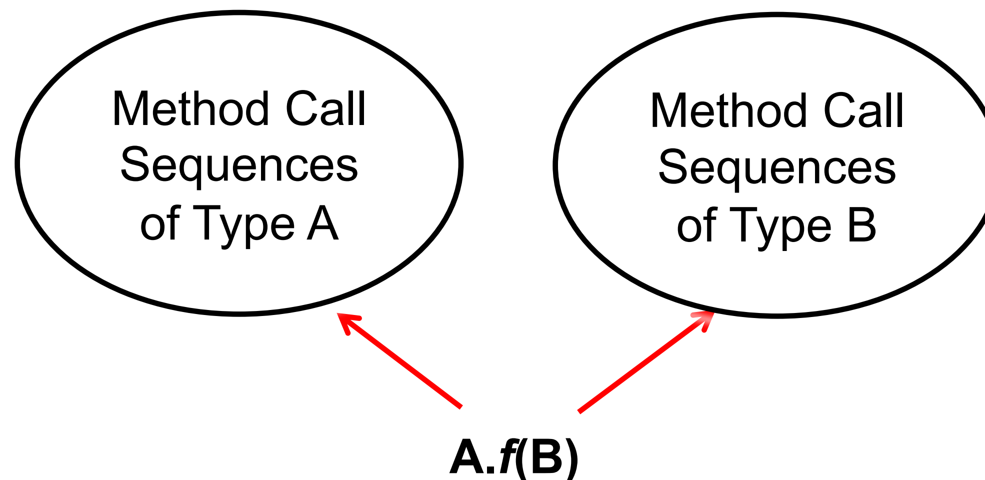
openDatabase() : Environment.envImpl, DatabaseConfig.allowCreate, ...

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



Experiments

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

Experiments

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

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

- With feedback is better
- With sequence recommendation is better

Experiments

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

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

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

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

- With feedback is better
- With sequence recommendation is better

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

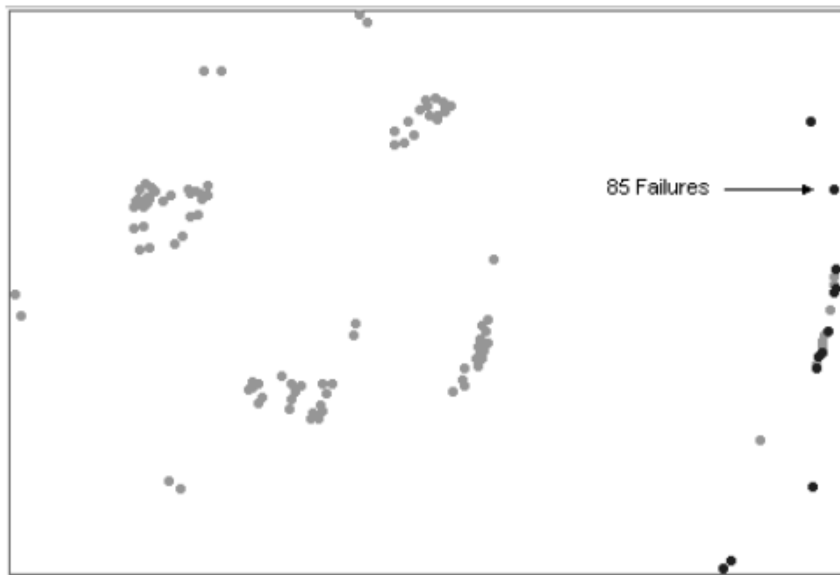
- Select a new test if it increases some coverage criteria, otherwise discard it
  - Method, line, branch coverage

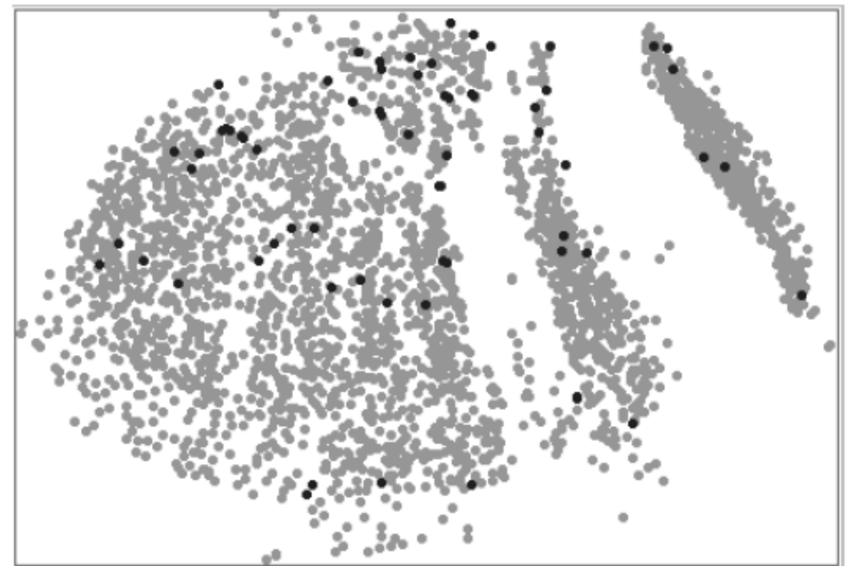
	Br1	Br2	Br3	Br4	…
Test1	1	0	1	1	…
Test2	1	0	1	1	…
Test3	0	1	0	0	…
Test4	1	0	1	0	…

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: $n \geq 0$
Postcondition: $s = (\sum j : 0 \leq j < n : b[j])$
Loop invariant: $0 \leq i \leq n$ and $s = (\sum j : 0 \leq j < i : b[j])$

- 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	…

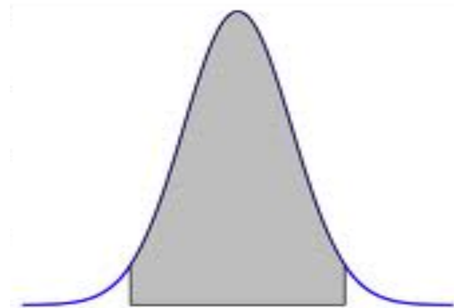
Br1 => !Br2

Br1 => Br3

Common Operational Models

- Data rules: implicit data value distributions

	min(Var1)	*max*(Var1)	*min*(Var2)	*max*(Var2)	…
Test1	0	10	0	11	…
Test2	0	32	-1	1	…
Test3	0	1	1	3	…
Test4	0	23	2	6	…



The distribution of *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

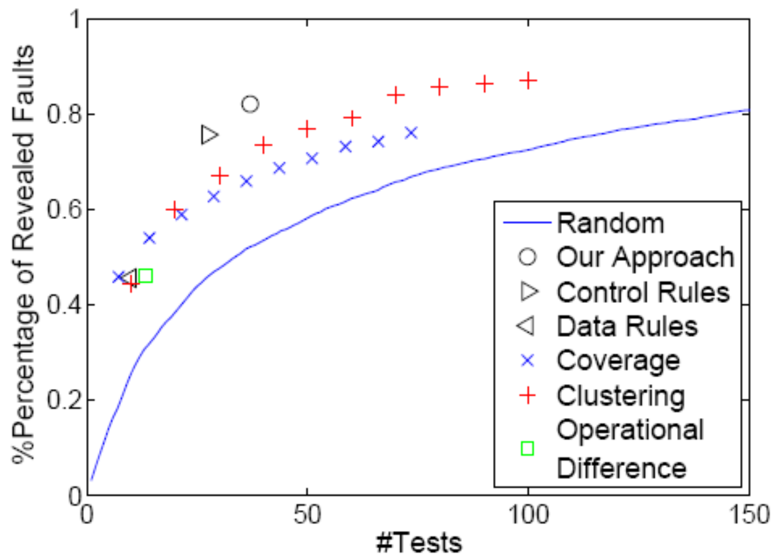
Experiments

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

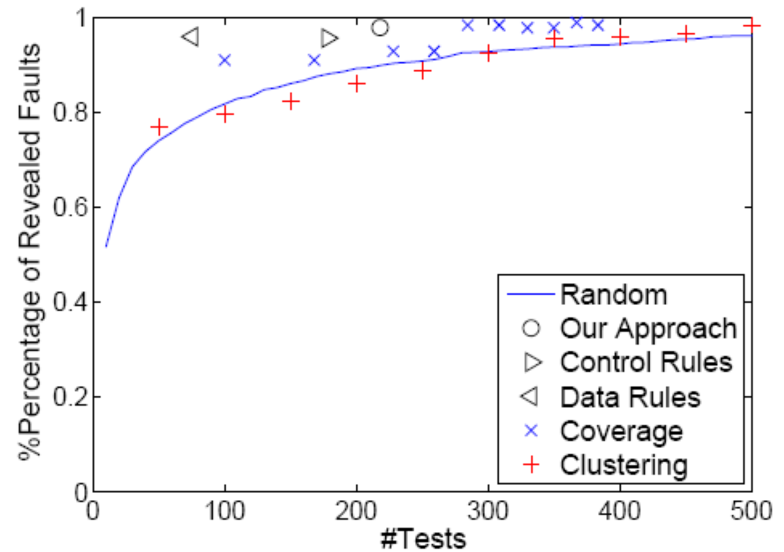
Program	LOC	Test Cases	Faulty Versions	Failed Tests (Avg.)	Program Description
print_tokens	539	4130	7	69	lexical analyzer
print_tokens2	489	4115	10	224	lexical analyzer
replace	507	5542	31	106	pattern replacement
schedule	397	2650	9	88	priority scheduler
schedule2	299	2710	9	33	priority scheduler
tcas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

Experiments

- Effectiveness
  - The number of the selected tests
  - The percentage of revealed faults



a) the *Siemens* suite



b) the *grep* program

Experiments

- Our approach is more effective

Program	Manual Test Suite		Our Approach		Random		Coverage(k=1)		Clustering		OD	
	#T	#F	#T	%F	#T	%F	#T	%F	#T	%F	#T	%F
print_tokens	4130	7	25	89	37	39	6	61	40	84	9	37
print_tokens2	4115	10	41	100	37	78	4	90	40	100	6	51
replace	5542	31	75	80	37	45	12	33	40	57	18	45
schedule	2650	9	31	86	37	48	7	26	40	60	10	33
schedule2	2710	9	32	62	37	34	5	26	40	47	13	30
tcas	1608	41	26	74	37	46	11	31	40	84	26	55
tot_info	1052	23	29	84	37	75	5	53	40	82	9	72
Siemens	3115	130	**37**	**82**	**37**	**52**	**7**	**46**	**40**	**73**	**13**	**46**
grep	809	20	**218**	**98**	**219**	**90**	**100**	**91**	**250**	**89**	-	-

Control Rules vs. Data Rules

- Control rules reveal more faults

Program	Original Test Suite		Our Approach		Control Rules		Data Rules	
	#Tests	#Faults	#Tests	%Faults	#Tests	%Faults	#Tests	%Faults
print_tokens	4130	7	25	89	17	88	8	50
print_tokens2	4115	10	41	100	30	100	10	61
replace	5542	31	75	80	60	73	16	37
schedule	2650	9	31	86	24	70	7	49
schedule2	2710	9	32	62	24	61	9	25
tcas	1608	41	26	74	15	68	12	23
tot_info	1052	23	29	84	21	71	9	74
Siemens suite	3115	130	**37**	**82**	27	76	10	46
grep	809	20	**218**	**98**	178	96	75	96

Random Test Suites

- Our approach works well on automatically generated test suites

Program	Automated Test Suite		Our Approach	
	#Tests	#Faults	#Tests	%Faults
print_tokens	1000	2	21	100
print_tokens2	1000	7	32	86
replace	1000	10	31	100
schedule	1000	3	26	33
schedule2	1000	4	17	100
tcas	1000	23	18	83
tot_info	1000	15	18	93
Siemens suite	1000	64	23	85
grep	1000	12	116	100

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

Background

- 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

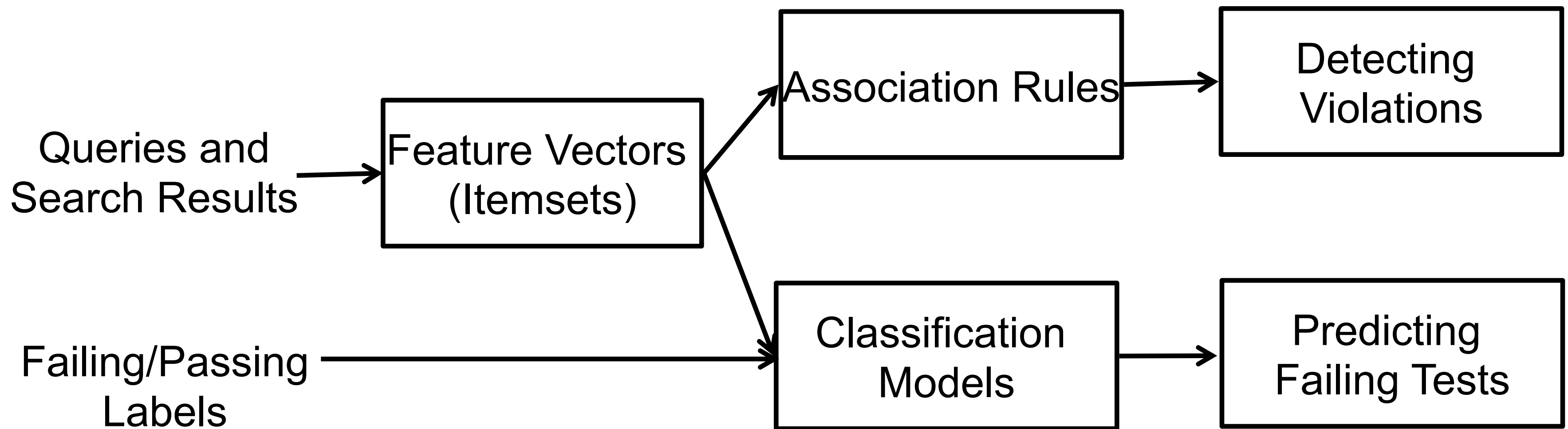
- Click through data as implicit feedback
  - Clicked results are relevant
- The idea
  - Let users inspect all search results of all queries
- Limitations
  - Position bias, summary bias
    - E.g., cannot find relevant pages that are not in the search results

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:boston colleges, QW:boston, QW:colleges, TwoWords, CommonQ, top10:searchboston.com, top1:searchboston.com, top10:en.wikipedia.org, …, SOMEGE100K, SOMELE1K
- 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 $Y\text{-}x_i \Rightarrow x_i$

    SE:bing=>top10:en.wikipedia.org

Learning Classification Models

- Feature Vectors
  - Can describe more general types of properties

	wordLength	*queryType*	*max*(domainRank)	google.com	facebook.com	…
Search Result List 1	2	common	900	1	0	…
Search Result List 2	3	hot	100000	0	1	…
Search Result List 3	1	hot	9782	1	1	…

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
7.HotQ,top10:movies.yahoo.com, => top10:imdb.com, : 20/20=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' results in multiple days
  - Google's search results from December 25, 2010 to March 31, 2011.
  - minsup = 20, minconf = 0.95, and maxL = 2

```
13.Q:hulu, => top1:hulu.com, : 91/91=1.0
14.Q:facebook, => top1:facebook.com, : 91/91=1.0
15.Q:youtube, => top1:youtube.com, : 91/91=1.0
16.Q:rosenbluth, => top1:rvacations.com, : 91/91=1.0
17.Q:espn picks, => top1:espn.go.com, : 91/91=1.0
18.Q:stock futures, => top1:bloomberg.com, : 91/91=1.0
```

  - Rules 13-18 show the rules about the top 1 results for the queries

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

Models	Data	Abnormal Data	Accuray	Precision	Recall
Decision Tree	3429	921	0.72	0.47	0.42
Naive Bayes	3429	921	0.66	0.36	0.38

  - 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

- **Wujie Zheng**, Hao Ma, Michael R. Lyu, Tao Xie, and Irwin King, “Mining Test Oracles of Web Search Engines”, *To appear in the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, Short Paper, Lawrence, Kan., US, November 6-10, 2011.
- **Wujie Zheng**, Qirun Zhang, and Michael R. Lyu, “Cross-library API Recommendation using Web Search Engines”, T*o appear in ESEC/FSE 2011, New Ideas Track*, Szeged, Hungary, September 5-9, 2011.
- Qirun Zhang, **Wujie Zheng**, and Michael R. Lyu, “Flow-Augmented Call Graph: A New Foundation for Taming API Complexity“, *Fundamental Approaches to Software Engineering (FASE'11)*, Saarbrücken, Germany, 26 March - 3 April, 2011.
- **Wujie Zheng**, Qirun Zhang, Michael Lyu, and Tao Xie, “Random Unit-Test Generation with MUT-aware Sequence Recommendation”, In Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering (*ASE 2010*), Short Paper, Antwerp, Belgium, September 2010.
- **Wujie Zheng,** Michael R. Lyu, and Tao Xie, “Test Selection for Result Inspection via Mining Predicate Rules”, In Proceedings of *ICSE Companion 2009*, pp. 219-222, Vancouver, Canada, May 2009.
- Xiaoqi Li, **Wujie Zheng**, Michael R. Lyu, “A Coalitional Game Model for Heat Diffusion Based Incentive Routing and Forwarding Scheme”, In Proceedings of *IFIP Networking 2009*, pp. 664-675, Aachen, Germany, May, 2009.

Q&A

Thanks!