

User Study of Tools to Assist Java Programmers in Finding Bugs

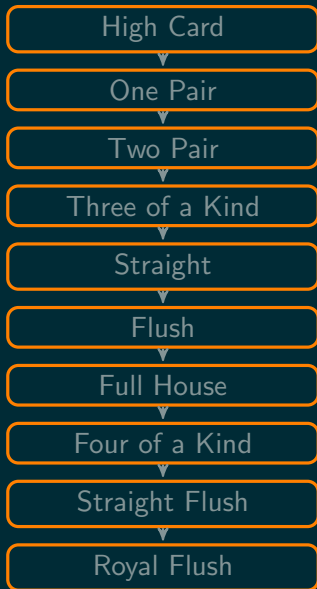
Andreas Bach Landgrebe

Department of Computer Science
Allegheny College

November 24, 2015

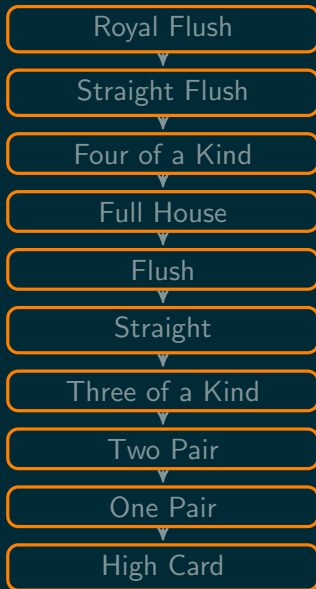
Motivation

Wrong Poker Representation



Motivation

Correct Poker Representation



Motivation

Definitions

- ▶ Logic Bug: is a bug in a program that causes it to operate incorrectly but not to terminate or crash.

Motivation

Definitions

- ▶ **Logic Bug:** is a bug in a program that causes it to operate incorrectly but not to terminate or crash.
- ▶ **Static Program Analysis:** analysis of computer software that is performed without actually executing programs.
- ▶ **Dynamic Program Analysis:** analysis of computer software that is performed by executing programs on a real or virtual processor.

Motivation

Why I decided this project

- ▶ Logic errors are the most difficult to find and fix.
- ▶ There is not one tool that will do everything for the programmer.

Motivation

Why I decided this project

- ▶ Logic errors are the most difficult to find and fix.
- ▶ There is not one tool that will do everything for the programmer.
- ▶ These three tools are the most popular pieces of software when looking for logic-based bugs in Java.

Overview of Project

- ▶ Three Static Code Analysis Tools
- ▶ Evaluate each of these three tools and provide an evaluation on how each tool operates.

Overview of Project

- ▶ Three Static Code Analysis Tools
- ▶ Evaluate each of these three tools and provide an evaluation on how each tool operates.
- ▶ Eclipse Integrated Development Environment

Overview of Project

- ▶ Three Static Code Analysis Tools
- ▶ Evaluate each of these three tools and provide an evaluation on how each tool operates.
- ▶ Eclipse Integrated Development Environment
 - ▶ Findbugs

Overview of Project

- ▶ Three Static Code Analysis Tools
- ▶ Evaluate each of these three tools and provide an evaluation on how each tool operates.
- ▶ Eclipse Integrated Development Environment
 - ▶ Findbugs
 - ▶ PMD

Overview of Project

- ▶ Three Static Code Analysis Tools
- ▶ Evaluate each of these three tools and provide an evaluation on how each tool operates.
- ▶ Eclipse Integrated Development Environment
 - ▶ Findbugs
 - ▶ PMD
 - ▶ checkstyle

Overview of Project

- ▶ Three Static Code Analysis Tools
- ▶ Evaluate each of these three tools and provide an evaluation on how each tool operates.
- ▶ Eclipse Integrated Development Environment
 - ▶ Findbugs
 - ▶ PMD
 - ▶ checkstyle
- ▶ Implementation of five Java files with buggy code.

Overview of Project

- ▶ Three Static Code Analysis Tools
- ▶ Evaluate each of these three tools and provide an evaluation on how each tool operates.
- ▶ Eclipse Integrated Development Environment
 - ▶ Findbugs
 - ▶ PMD
 - ▶ checkstyle
- ▶ Implementation of five Java files with buggy code.
- ▶ Human Study starting with students of Computer Science 112.

Tools

Findbugs

FindBugs - org.eclipse.equinox.p2.ui/src/org/eclipse/equinox/internal/provisional/p2/ui/ResolutionResult.java - Eclipse SDK

File Edit Source Refactor Navigate Search Project Run Window Help

Bug Explorer

- org.eclipse.equinox.p2.core (29) [dev.eclipse.org]
 - Class defines compareTo(...) and uses Object.equals() (3)
 - Class doesn't override equals in superclass (1)
 - Class implements Cloneable but does not define or use clone method (1)
 - equals() method does not check for null argument (1)
 - Inconsistent synchronization (1)
 - Inefficient use of keySet iterator instead of entrySet iterator (1)
 - Method ignores exceptional return value (12)
 - Method invokes inefficient new String(String) constructor (1)
 - Method uses the same code for two branches (1)
 - Nullcheck of value previously dereferenced (2)
 - private readResolve method not inherited by subclasses (1)
 - Should be a static inner class (4)
- org.eclipse.equinox.p2.ui (74) v20090203 [dev.eclipse.org]
 - Ambiguous invocation of either an inherited or outer method (1)
 - Class defines field that masks a superclass field (1)
 - Class doesn't override equals in superclass (1)
 - Class names shouldn't shadow simple name of superclass (1)
 - Dead store to local variable (3)
 - Dead store to iuSummaryStatus**
 - Dead store to mon
 - Dead store to mon
 - Method call passes null for nonnull parameter (1)
 - Method might ignore exception (1)
 - Possible null pointer dereference in method on exception path
 - Read of unwritten field (4)
 - Should be a static inner class (1)
 - Unread field (7)
 - Unused field (45)
 - Unusual equals method (2)
 - Unwritten field (3)
 - Write to static field from instance method (1)

ResolutionResult

```
47 }
48
49 public void addStatus(IInstallableUnit iu, IStatus status) {
50     MultiStatus iuSummaryStatus = (MultiStatus) iuToStatusMap.get(iu);
51     if (iuSummaryStatus == null) {
52         iuSummaryStatus = new MultiStatus(ProvUIActivator.PLUGIN_ID,
53     } else {
54         iuSummaryStatus.add(status);
55     }
56
57 private String getIUString(IInstallableUnit iu) {
58     if (iu == null)
59         return ProvUIMessages.PlanStatusHelper_Items;
60     // Get the iu name in the default locale
61     String name = IUPropertyUtils.getProperty(iu, IInstallableUnit
```

Properties

Properties Problems

Bug: Dead store to iuSummaryStatus

Bug: Dead store to iuSummaryStatus
Pattern id: DLS_DEAD_LOCAL_STORE, **type:** DLS, **category:** STYLE

This instruction assigns a value to a local variable, but the value is not read or used in any subsequent instruction. Often, this indicates an error, because the value computed is never used.

Note that Sun's javac compiler often generates dead stores for final local variables. Because FindBugs is a bytecode-based tool, there is no easy way to eliminate these false positives.

Dead store to iuSummaryStatus 54M of 106M

Tools

Features of FindBugs

- ▶ Operates on Java bytecode
- ▶ Additional rule sets can be plugged in.

Tools

Features of FindBugs

- ▶ Operates on Java bytecode
- ▶ Additional rule sets can be plugged in.
- ▶ Bugs Rank 1-20

Tools

Features of FindBugs

- ▶ Operates on Java bytecode
- ▶ Additional rule sets can be plugged in.
- ▶ Bugs Rank 1-20
 - ▶ Rank 1-4: Scariest

Tools

Features of FindBugs

- ▶ Operates on Java bytecode
- ▶ Additional rule sets can be plugged in.
- ▶ Bugs Rank 1-20
 - ▶ Rank 1-4: Scariest
 - ▶ Rank 5-9: Scary

Tools

Features of FindBugs

- ▶ Operates on Java bytecode
- ▶ Additional rule sets can be plugged in.
- ▶ Bugs Rank 1-20
 - ▶ Rank 1-4: Scariest
 - ▶ Rank 5-9: Scary
 - ▶ Rank 10-14: Troubling

Tools

Features of FindBugs

- ▶ Operates on Java bytecode
- ▶ Additional rule sets can be plugged in.
- ▶ Bugs Rank 1-20
 - ▶ Rank 1-4: Scariest
 - ▶ Rank 5-9: Scary
 - ▶ Rank 10-14: Troubling
 - ▶ Rank 15-20: Concern

Tools

PMD

The screenshot shows the Eclipse IDE with the PMD plugin. The Package Explorer on the left shows the project structure. The source code editor in the center displays the `HotelModel.java` file. The Violations Outline at the bottom left lists 12 violations, including messages like "Each class should declare at least one...", "The String literal 'Paris' appears 4 times", and "Parameter 'city' is not assigned and cc...". The Violations Overview table at the bottom right provides a summary of violations across the project.

Element	# Violations	# Violations/LOC	# Violations/Method	Project
com.wakaleo.tutorials.hotelworld.businessobjects	24	0.19	0.83	HotelWorld
com.wakaleo.tutorials.hotelworld.model	17	0.27	2.83	HotelWorld
HotelModel.java	17	0.27	2.83	HotelWorld
com.wakaleo.tutorials.hotelworld.tapestry	9	0.3	0.69	HotelWorld



Tools

PMD

- ▶ Java source code analyzer
- ▶ Built-in rules and supports the ability to write custom rules
- ▶ Not true errors, but rather inefficient code
 - ▶ PMD possible flaws to detect
 - ▶ Possible bugs - Empty try/catch/switch blocks.
 - ▶ Dead code - Unused variables and private methods.

Tools

PMD

- ▶ Java source code analyzer
- ▶ Built-in rules and supports the ability to write custom rules
- ▶ Not true errors, but rather inefficient code
 - ▶ PMD possible flaws to detect
 - ▶ Possible bugs - Empty try/catch/switch blocks.
 - ▶ Dead code - Unused variables and private methods.
 - ▶ Empty if/while statements

Tools

PMD

- ▶ Java source code analyzer
- ▶ Built-in rules and supports the ability to write custom rules
- ▶ Not true errors, but rather inefficient code
 - ▶ PMD possible flaws to detect
 - ▶ Possible bugs - Empty try/catch/switch blocks.
 - ▶ Dead code - Unused variables and private methods.
 - ▶ Empty if/while statements
 - ▶ Overcomplicated expressions - for loops that could be while loops

Tools

PMD

- ▶ Java source code analyzer
- ▶ Built-in rules and supports the ability to write custom rules
- ▶ Not true errors, but rather inefficient code
 - ▶ PMD possible flaws to detect
 - ▶ Possible bugs - Empty try/catch/switch blocks.
 - ▶ Dead code - Unused variables and private methods.
 - ▶ Empty if/while statements
 - ▶ Overcomplicated expressions - for loops that could be while loops
 - ▶ Suboptimal code - Wasteful String usage

Tools

PMD

- ▶ Java source code analyzer
- ▶ Built-in rules and supports the ability to write custom rules
- ▶ Not true errors, but rather inefficient code
 - ▶ PMD possible flaws to detect
 - ▶ Possible bugs - Empty try/catch/switch blocks.
 - ▶ Dead code - Unused variables and private methods.
 - ▶ Empty if/while statements
 - ▶ Overcomplicated expressions - for loops that could be while loops
 - ▶ Suboptimal code - Wasteful String usage
 - ▶ Classes with high Cyclomatic Complexity measurements (complexity of a program).

Tools

PMD

- ▶ Java source code analyzer
- ▶ Built-in rules and supports the ability to write custom rules
- ▶ Not true errors, but rather inefficient code
 - ▶ PMD possible flaws to detect
 - ▶ Possible bugs - Empty try/catch/switch blocks.
 - ▶ Dead code - Unused variables and private methods.
 - ▶ Empty if/while statements
 - ▶ Overcomplicated expressions - for loops that could be while loops
 - ▶ Suboptimal code - Wasteful String usage
 - ▶ Classes with high Cyclomatic Complexity measurements (complexity of a program).
 - ▶ Duplicated code - copied-pasted code could mean copied/pasted bugs.

Tools

Checkstyle

The screenshot shows the Eclipse IDE with the following components:

- Editor:** Displays the source code of `ObjectCheckerImpl.java`. The code includes package declarations, imports, and a public class `ObjectCheckerImpl` that implements `ObjectChecker`. It features a `MatcherAssertEdge` class and methods for `isEqual` and `isNotEqual`.
- Problems View:** Shows two Checkstyle violations:
 - Line 44: Line contains a tab character. (Marker count: 2)
 - Line 49: Name %0 must match pattern %1. (Marker count: 1)
- Outline View:** Shows the class structure, including `ObjectCheckerImpl` and its nested class `MatcherAssertEdge`.

Tools

Checkstyle

- ▶ Java source code analyzer
- ▶ Checks if Java source code adheres to a coding standard
- ▶ Highly configurable to support any coding standard
- ▶ Designed to check for programming practices.

Evaluation Strategy

- ▶ Five Java programs with logic-based bugs

Evaluation Strategy

- ▶ Five Java programs with logic-based bugs
 - ▶ Misplaced semi-colon

Evaluation Strategy

- ▶ Five Java programs with logic-based bugs
 - ▶ Misplaced semi-colon
 - ▶ Velocity logic-bug

Evaluation Strategy

- ▶ Five Java programs with logic-based bugs
 - ▶ Misplaced semi-colon
 - ▶ Velocity logic-bug
 - ▶ `String ==` vs. `String equals()` comparison bug

Evaluation Strategy

- ▶ Five Java programs with logic-based bugs
 - ▶ Misplaced semi-colon
 - ▶ Velocity logic-bug
 - ▶ `String ==` vs. `String equals()` comparison bug
 - ▶ Memory leak bug

Evaluation Strategy

- ▶ Five Java programs with logic-based bugs
 - ▶ Misplaced semi-colon
 - ▶ Velocity logic-bug
 - ▶ `String ==` vs. `String equals()` comparison bug
 - ▶ Memory leak bug
 - ▶ Switch case bug

Evaluation Strategy

Human Study Survey

- ▶ Did you find the bugs?
- ▶ If yes, did Findbugs assist you in finding these bugs?
- ▶ If yes, did PMD assist you in finding these bugs?
- ▶ If yes, did Checkstyle assist you in finding these bugs?

Evaluation Strategy

Human Study Survey

- ▶ Did you find the bugs?
- ▶ If yes, did Findbugs assist you in finding these bugs?
- ▶ If yes, did PMD assist you in finding these bugs?
- ▶ If yes, did Checkstyle assist you in finding these bugs?
- ▶ Would you use any of these tools in the future for finding logic-based bugs?

Evaluation Strategy

Human Study Survey

- ▶ Did you find the bugs?
- ▶ If yes, did Findbugs assist you in finding these bugs?
- ▶ If yes, did PMD assist you in finding these bugs?
- ▶ If yes, did Checkstyle assist you in finding these bugs?
- ▶ Would you use any of these tools in the future for finding logic-based bugs?
- ▶ Suggestion to further improve these tools.

Evaluation Strategy

How To Evaluate Human Study

- ▶ Computer Science 112 Students
- ▶ Knowledge past Computer Science 111

Evaluation Strategy

How To Evaluate Human Study

- ▶ Computer Science 112 Students
- ▶ Knowledge past Computer Science 111
- ▶ 24 hours to evaluate and find the logic bugs

Evaluation Strategy

How To Evaluate Human Study

- ▶ Computer Science 112 Students
- ▶ Knowledge past Computer Science 111
- ▶ 24 hours to evaluate and find the logic bugs
- ▶ IRB approval and CITI Training

Deliverables

- ▶ Evaluation of how each of these three tools operates.

Deliverables

- ▶ Evaluation of how each of these three tools operates.
- ▶ User study of whether these three tools assisted in finding these bugs.

Deliverables

- ▶ Evaluation of how each of these three tools operates.
- ▶ User study of whether these three tools assisted in finding these bugs.
- ▶ Human Study of whether these three tools were able to help Java programmers finding logic bugs.

Deliverables

- ▶ Evaluation of how each of these three tools operates.
- ▶ User study of whether these three tools assisted in finding these bugs.
- ▶ Human Study of whether these three tools were able to help Java programmers finding logic bugs.
- ▶ Java source code with logic-based bugs

Conclusion

Thank You!

Comments, Questions or Concerns?