Corpus Studies
Corpus Studies

• How often does X occur in the real world? X could be:
  • A bug
    • If it occurs often, it's worth preventing or detecting
  • A use of a feature or tool
  • Is my tool applicable to real-world code?
• What is the world like?
  • e.g., what kinds of questions do people ask about Rust?
Corpus Study Techniques

• Get a corpus
  • Replicable studies require a fixed corpus
    • Snapshot of GitHub?
      • Why this corpus? Consider external validity.
  • Write a tool or analyze manually.
• Sample?
True and False Positives and Negatives

- **Context:** finding needles in a haystack. (or: finding bugs in programs. or: proving that a program lacks a certain bug)

- **Procedure:** investigate each item in the haystack. Check if each item is a needle.

- **True positive:** This needle is a needle. 😊

- **False positive:** This pine cone is a needle. 😞

- **True negative:** This pine cone is *not* a needle. 😊

- **False negative:** This needle is *not* a needle. 😞

*Complete analyses never make this mistake*

*Sound analyses never make this mistake*
Research Questions

• RQ1: how many bugs of type X are there in this corpus? Possible answers:
  • At least n.
  • Exactly n.
  • No more than n.

• RQ2: Can my tool find bugs?
  • Yes, lots!
  • Yes, but only a small fraction of the ones that are present.
  • No, but that's because there weren't any bugs of that type.
  • No, but it's unsound, and I have no idea how many bugs there are of that type.
Analysis Techniques

• Sound analysis: find everything in category X
  • Might also find things NOT in category X
  • Mitigate with manual analysis
• Complete analysis: only find things in category X
  • But we have no idea how many we missed
• Sound and complete: nice if you can get it…
  • Rice's theorem: "all non-trivial, semantic properties of programs are undecidable."
Commits Typically Specify Which Bugs They Fix

Authors mined commit logs to find bug numbers
Corpus Study 1: When Do Changes Induce Fixes?

• Jacek Śliwerski, Thomas Zimmermann, Andreas Zeller

• RQ: When do developers insert bugs?
  • Imagine if your IDE could tell you: "your change is likely buggy!"

• Problem: how do you tell when a change inserted a bug?

• Approach: if a change's code needs to be changed AGAIN, the change induced a fix.
Fix-Inducing Changes

• "A fix-inducing change is a change that later gets undone by a fix."

• Suppose change δ changes line 42 to fix bug B.

• Aha, revisions 675d7f and 56879a changed line 42.

• But 675d7f was committed AFTER bug B was reported, so that wasn't the cause.

• 56879a is suspect.
Fix-Inducing Commits Are Large

<table>
<thead>
<tr>
<th></th>
<th>fix-inducing</th>
<th>¬fix-inducing</th>
<th>all</th>
</tr>
</thead>
<tbody>
<tr>
<td>fix</td>
<td>3.82±26.32</td>
<td>2.08± 7.42</td>
<td>2.73± 7.87</td>
</tr>
<tr>
<td>¬ fix</td>
<td>11.30±63.02</td>
<td>2.77±14.94</td>
<td>3.81±26.32</td>
</tr>
<tr>
<td>all</td>
<td>7.49±44.37</td>
<td>2.61±13.66</td>
<td>3.52±22.81</td>
</tr>
</tbody>
</table>

Table 3: Average sizes of fix and fix-inducing transactions for ECLIPSE
Table 5: Distribution of fixes and fix-inducing changes across day of week in ECLIPSE

<table>
<thead>
<tr>
<th>% of revisions</th>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
<th>Sat</th>
<th>Sun</th>
<th>avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P(\text{fix})$</td>
<td>18.4</td>
<td>20.9</td>
<td>20.0</td>
<td>22.3</td>
<td>24.0</td>
<td>14.7</td>
<td>16.9</td>
<td>20.8</td>
</tr>
<tr>
<td>$P(\text{bug})$</td>
<td>11.3</td>
<td>10.4</td>
<td>11.1</td>
<td>12.1</td>
<td>12.2</td>
<td>11.7</td>
<td>11.6</td>
<td>11.4</td>
</tr>
<tr>
<td>$P(\text{bug} \cap \text{fix})$</td>
<td>4.6</td>
<td>4.8</td>
<td>4.6</td>
<td>5.2</td>
<td>5.6</td>
<td>4.5</td>
<td>4.5</td>
<td>4.9</td>
</tr>
<tr>
<td>$P(\neg \text{bug} \cap \neg \text{fix})$</td>
<td>74.9</td>
<td>73.5</td>
<td>73.5</td>
<td>70.8</td>
<td>63.4</td>
<td>78.1</td>
<td>76.0</td>
<td>72.7</td>
</tr>
</tbody>
</table>

$P(\text{bug} \mid \text{fix})$ | 25.1 | 22.9 | 23.3 | 23.5 | 23.2 | 30.3 | 26.4 | 23.7 |
$P(\text{bug} \mid \neg \text{fix})$ | 8.2  | 7.1  | 8.1  | 8.8  | 8.7  | 8.4  | 8.6  | 8.1  |
Explanation?

• Maybe developers commit without as much testing or review on Fridays because they don't want to leave tasks pending over the weekend.
Another Example: Object Protocols

• Example: a File is either **Open** or **Closed**.

• On **Open** file: `close()` ok. `open()` illegal.

• On **Closed** file: `open()` ok. `close()` illegal.

• "Definition: A type defines an object protocol if the concrete state of objects of that type can be abstracted into a finite number of abstract states of which clients must be aware in order to use that type correctly, and among which object instances will dynamically transition."
Finding Protocols

```java
// from java.util.concurrent.ArrayBlockingQueue.Itr
public void remove() {
    final ReentrantLock lock = ArrayBlockingQueue.this.lock;
    lock.lock();
    try {
        int i = this.lastRet;
        if (i == -1)
            throw new IllegalStateException();
        lastRet = -1;
        // ... method continues
    } // ... method continues
}

// from javax.swing.undo.AbstractUndoableEdit
public void undo() throws CannotUndoException {
    if (!canUndo()) {
        throw new CannotUndoException();
    }
    hasBeenDone = false;
}

public boolean canUndo() { return alive && hasBeenDone; }
```

Strategy: find exceptions that are thrown conditionally, depending on field data.

Sound? Complete? What is an object protocol, anyway?
Method

• Phase 1: find protocols
  • Static analysis to find candidate protocols (unsound, incomplete)
  • Manual investigation of reports
• Phase 2: find how often protocols are used
  • Find calls to protocol methods
An Empirical Study of Object Protocols in the Wild

ProtocolFinder

Classes, Methods, Overriden Methods

Candidate Locations

Examination

Protocol Methods, Protocol Types

Protocol Methods

Protocol Evidence

Protocol Categories

Classes Calling Protocol Methods

Classes with Protocol Fields

Random Sample

Examination

ProtocolUsage

Classes Calling Protocol Methods

Est. of Classes w/ Field Wrap Protocols

Legend:
- Automated analysis
- Manual analysis
- Program under analysis
- Result Data
- Temporary Data
- Information Flow
- Correlation

Fig. 1. As explaining the experimental procedure appropriate test cases must be found to exercise all of the possible protocols in an application. For the same reasons, a dynamic approach would require examining only programs that were accompanied by sufficient test cases, and thus, was ruled out. By comparison, a static analysis can be run on any open-source program. In the end, we decided to develop a conservative static analysis that would eliminate many (although not all) false negatives while reducing manual effort. A subsequent manual examination is used to eliminate false positives.

ProtocolFinder is a static analysis created for this study that attempts to find object protocols by searching for locations in code where protocol violations are detected. Specifically, it looks for locations in code where instance methods throw exceptions as a result of reading instance fields. The intuition behind the analysis is simple: In our preliminary investigations we noticed that many protocol methods throw exceptions when object protocols are violated. Because our definition of object protocol depends on the abstract state of the method receiver, we expect that any exceptions thrown for protocol violation will be thrown in instance methods and as a result of reading an
Analysis

• Dynamic analysis?
  • False negatives (missing protocols due to lack of test cases)

• Manual analysis?
  • Too slow/expensive

• Static analysis?
  • False positives
ProtocolFinder

- Sound (does it find every protocol)?
  - No (not even with respect to the paper's definition of protocol)

- Complete (does it ONLY find protocols)?
  - That depends on what a protocol is.

  ```java
  if (f()) {
    throw new Error();
  }
  ```

- What about a broader definition of protocol?
Corpus (Phase 1)

- Four open source programs from Qualitas corpus
  - Large, popular programs; mix of libraries and applications
- 1.9 MLOC (Java)

To what do you expect the results to generalize?

What corpus would you have picked?
Number of Protocols

Table 2. The results of running the ProtocolFinder on the four phase one code bases

<table>
<thead>
<tr>
<th>Program</th>
<th>Protocol Candidates</th>
<th>Evidence Candidates</th>
<th>E.C.</th>
<th>P.T.</th>
<th>T.S.E.C.</th>
<th>Precision</th>
<th>%E.C.</th>
<th>%P.T.</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSL</td>
<td>2,690</td>
<td>613</td>
<td>195</td>
<td>842</td>
<td>54</td>
<td>22.8%</td>
<td>2.3%</td>
<td>8.2%</td>
</tr>
<tr>
<td>PMD</td>
<td>32</td>
<td>7</td>
<td>3</td>
<td>10</td>
<td>0</td>
<td>21.9%</td>
<td>0.8%</td>
<td>2.4%</td>
</tr>
<tr>
<td>Azureus</td>
<td>136</td>
<td>24</td>
<td>19</td>
<td>32</td>
<td>4</td>
<td>17.6%</td>
<td>2.1%</td>
<td>2.6%</td>
</tr>
<tr>
<td>JDT</td>
<td>62</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>0</td>
<td>6.5%</td>
<td>1.3%</td>
<td>1.5%</td>
</tr>
<tr>
<td>Total</td>
<td>2,920</td>
<td>648</td>
<td>221</td>
<td>889</td>
<td>58</td>
<td>22.2%</td>
<td>2.2%</td>
<td>7.2%</td>
</tr>
</tbody>
</table>

T.S.E.C.=Thread-Safe Evidence Classes E.C.= Evidence Classes
P.T.= Protocol Types
Types of Protocols

Table 3. Categorization of each of the 648 reports issued by the ProtocolFinder that were evidence for actual protocols.

<table>
<thead>
<tr>
<th>Category</th>
<th>Protocol Evidence</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>182</td>
<td>28.1%</td>
</tr>
<tr>
<td>Deactivation</td>
<td>167</td>
<td>25.8%</td>
</tr>
<tr>
<td>Type Qualifier</td>
<td>106</td>
<td>16.4%</td>
</tr>
<tr>
<td>Dynamic Preparation</td>
<td>52</td>
<td>8.0%</td>
</tr>
<tr>
<td>Boundary</td>
<td>51</td>
<td>7.9%</td>
</tr>
<tr>
<td>Redundant Operation</td>
<td>47</td>
<td>7.3%</td>
</tr>
<tr>
<td>Domain Mode</td>
<td>31</td>
<td>4.8%</td>
</tr>
<tr>
<td>Others</td>
<td>12</td>
<td>1.9%</td>
</tr>
</tbody>
</table>
Phase 2: Protocol Usage

Table 4. The results of running the ProtocolUsage analysis on the sixteen candidate code bases.

<table>
<thead>
<tr>
<th>Program</th>
<th>Classes Calling</th>
<th>% Classes w/ Protocol Methods</th>
<th>% Classes w/ Prot. Fields</th>
<th>% Exposes Protocol Rate</th>
<th>Est. Classes From Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSL</td>
<td>1012</td>
<td>12%</td>
<td>1082</td>
<td>13%</td>
<td>157</td>
</tr>
<tr>
<td>PMD</td>
<td>85</td>
<td>22%</td>
<td>29</td>
<td>7%</td>
<td>0</td>
</tr>
<tr>
<td>Azureus</td>
<td>198</td>
<td>22%</td>
<td>763</td>
<td>8%</td>
<td>31%</td>
</tr>
<tr>
<td>JDT</td>
<td>13</td>
<td>4%</td>
<td>18</td>
<td>6%</td>
<td>0</td>
</tr>
<tr>
<td>ant</td>
<td>209</td>
<td>28%</td>
<td>187</td>
<td>19%</td>
<td>20%</td>
</tr>
<tr>
<td>antlr</td>
<td>20</td>
<td>11%</td>
<td>16</td>
<td>9%</td>
<td>0</td>
</tr>
<tr>
<td>aoi</td>
<td>25</td>
<td>6%</td>
<td>37</td>
<td>8%</td>
<td>0</td>
</tr>
<tr>
<td>columba</td>
<td>120</td>
<td>12%</td>
<td>246</td>
<td>25%</td>
<td>8%</td>
</tr>
<tr>
<td>crystal</td>
<td>9</td>
<td>5%</td>
<td>2</td>
<td>1%</td>
<td>0</td>
</tr>
<tr>
<td>drjava</td>
<td>49</td>
<td>8%</td>
<td>107</td>
<td>17%</td>
<td>0</td>
</tr>
<tr>
<td>freecol</td>
<td>94</td>
<td>22%</td>
<td>117</td>
<td>27%</td>
<td>0</td>
</tr>
<tr>
<td>log4j</td>
<td>39</td>
<td>22%</td>
<td>32</td>
<td>18%</td>
<td>0</td>
</tr>
<tr>
<td>lucene</td>
<td>30</td>
<td>11%</td>
<td>27</td>
<td>10%</td>
<td>0</td>
</tr>
<tr>
<td>poi</td>
<td>41</td>
<td>10%</td>
<td>13</td>
<td>3%</td>
<td>100%</td>
</tr>
<tr>
<td>quartz</td>
<td>16</td>
<td>13%</td>
<td>10</td>
<td>8%</td>
<td>0</td>
</tr>
<tr>
<td>xalan</td>
<td>91</td>
<td>9%</td>
<td>142</td>
<td>14%</td>
<td>13%</td>
</tr>
<tr>
<td>Total</td>
<td>2111</td>
<td>13%</td>
<td>2141</td>
<td>13%</td>
<td>17%</td>
</tr>
<tr>
<td>W/O JSL</td>
<td>1099</td>
<td>15%</td>
<td>1059</td>
<td>14%</td>
<td>18%</td>
</tr>
</tbody>
</table>
Implications on Language Design

• Do you wish you had typestate?

• Do these results mean we should?