Contracts in the Wild: a Study of Java Programs

Jens Dietrich\textsuperscript{1} David J. Pearce\textsuperscript{2} Kamil Jezek\textsuperscript{3} Premek Brada\textsuperscript{3}

\textsuperscript{1}Massey University, New Zealand  
\textsuperscript{2}Victoria University of Wellington, New Zealand  
\textsuperscript{3}University of West Bohemia Pilsen, Czech Republic
Assertions

“In order that the man who checks may not have too difficult a task the programmer should make a number of definite assertions which can be checked individually, and from which the correctness of the whole program easily follows.”

Alan Turing, 1949
Closet Contract Conjecture

“It’s this question that leads to the Closet Contract Conjecture: are the contracts of Eiffel libraries a figment of the Eiffel programmer’s obsession with this mechanism? Or are they present anyway, hidden, in non-Eiffel libraries as well?”

Arnout & Meyer, 2002
- Projects **Downloaded & Filtered** from Maven Central

- Contract usage classified using **static analysis**

- Data is aggregated and **analysed** using “stats”
Research Questions

- **RQ1.** *Which language features are used to represent contracts in real-world Java programs?*

- **RQ2.** *How does the use of contracts change throughout the evolution of a program?*

- **RQ3.** *Are contracts used correctly in the context of program evolution in real-world Java programs?*

- **RQ4.** *Are contracts used correctly in the context of subtyping in real-world Java programs?*
Contract Patterns

- Runtime Exceptions (Conditional).

```java
if (...) { throw new IllegalArgumentException(); }
```

- Runtime Exceptions (Unconditional).

```java
throw new UnsuppportedException();
```

- Contract APIs.

```java
Preconditions.checkState(index >= 0, "error");
```

- Assertions.

```java
assert index >= 0;
```

- Annotations (e.g. JSR303, JSR349, FindBugs, JML, Lombok, etc).

```java
void f(@NonNull String str) { ... }
```
## Contract Classification

<table>
<thead>
<tr>
<th>Contract Type</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime Exception (Conditional)</td>
<td>precondition</td>
</tr>
<tr>
<td>Runtime Exception (Unconditional)</td>
<td>precondition</td>
</tr>
<tr>
<td>Contract API</td>
<td>precondition</td>
</tr>
<tr>
<td>Assert</td>
<td>unclassified</td>
</tr>
<tr>
<td>Annotation (parameter)</td>
<td>precondition</td>
</tr>
<tr>
<td>Annotation (method)</td>
<td>postcondition</td>
</tr>
<tr>
<td>Annotation (field, class)</td>
<td>class invariant</td>
</tr>
</tbody>
</table>
@Override
public void visit(ThrowStmt n, Object arg) {

// look for the following pattern:
// if (<condition>) throw new <exception>(<args>);

if (n.getExpr() instanceof ObjectCreationExpr
    && (n.getParentNode() instanceof IfStmt
        || (n.getParentNode() instanceof BlockStmt
            && n.getParentNode().getParentNode() instanceof IfStmt))) {
    ...

- Used javaparser (great!) to parse Java source code

- Code available here:
  https://bitbucket.org/jensdietrich/contractstudy
Corpus

- **Top 200 artefacts** on mvnrepository.com/popular
- **Removed** projects without Java source code (e.g. scala)
- For each artefact **all versions** downloaded
- **Obtained** 176 projects, 6,934 versions, 4.6GB
### RQ1: Which Language Features used for Contracts?

<table>
<thead>
<tr>
<th>Category</th>
<th>Notes</th>
<th>Constructs (all versions)</th>
<th>Constructs (latest versions)</th>
<th>programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime Exception</td>
<td>(Conditional)</td>
<td>484,964</td>
<td>15,720</td>
<td>155</td>
</tr>
<tr>
<td></td>
<td>(Unconditional)</td>
<td>123,966</td>
<td>3,084</td>
<td>122</td>
</tr>
<tr>
<td>Contract API</td>
<td>(Guava)</td>
<td>49,021</td>
<td>1,188</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>(Spring)</td>
<td>100,232</td>
<td>2,148</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>(Commons)</td>
<td>879</td>
<td>110</td>
<td>6</td>
</tr>
<tr>
<td>Assert</td>
<td></td>
<td>131,340</td>
<td>3,284</td>
<td>52</td>
</tr>
<tr>
<td>Annotation</td>
<td>(JSR303/349)</td>
<td>586</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>(JSR305)</td>
<td>33,281</td>
<td>911</td>
<td>6</td>
</tr>
<tr>
<td>Precondition</td>
<td></td>
<td>786,723</td>
<td>22,969</td>
<td>160</td>
</tr>
<tr>
<td>Postcondition</td>
<td></td>
<td>2,413</td>
<td>112</td>
<td>6</td>
</tr>
<tr>
<td>Class Invariant</td>
<td></td>
<td>3,793</td>
<td>100</td>
<td>5</td>
</tr>
<tr>
<td>Unclassified</td>
<td></td>
<td>131,340</td>
<td>3,284</td>
<td>52</td>
</tr>
</tbody>
</table>
RQ1: Which Language Features used for Contracts?

<table>
<thead>
<tr>
<th>Category</th>
<th>Programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime Exception</td>
<td>open-jdk (3,695), elasticsearch (1,348), lucene-core (612), netty (553), hadoop-common (550)</td>
</tr>
<tr>
<td>Contract API</td>
<td>guava (948), spring (661), spring-test (262), spring-web (218), spring-core (208)</td>
</tr>
<tr>
<td>Assertion</td>
<td>lucene-core (1,000), elasticsearch (656), open-jdk (390), gwt-user (371), gwt-servlet (371)</td>
</tr>
<tr>
<td>Annotation</td>
<td>guava (859), reflections (46), hibernate-validator (20), annotations (4), jsr305 (2)</td>
</tr>
</tbody>
</table>

- Few programs use zero contract types ($\frac{16}{176}$, 9%)
- Some programs use only one contract type ($\frac{32}{176}$, 18%)
- Most programs use multiple contract types ($\frac{63}{176}$ use 2, $\frac{59}{176}$ use 3, $\frac{6}{176}$ use 4 or more)
- High Gini of contract usage (0.74) means large inequality
Generally speaking, if projects use contracts they **keep** using them.

- Contract usage **increases proportionally** with program size.
**RQ3**: Are Contracts used Correctly in Time?

<table>
<thead>
<tr>
<th>Change</th>
<th>Critical</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unchanged</td>
<td>no</td>
<td>652,395</td>
</tr>
<tr>
<td>Minor change</td>
<td>no</td>
<td>1,512</td>
</tr>
<tr>
<td>Preconditions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(weakened)</td>
<td>no</td>
<td>12,675</td>
</tr>
<tr>
<td>(strengthened)</td>
<td>yes</td>
<td>2,777</td>
</tr>
<tr>
<td>Postconditions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(strengthened)</td>
<td>no</td>
<td>18</td>
</tr>
<tr>
<td>(weakened)</td>
<td>yes</td>
<td>7</td>
</tr>
<tr>
<td>Unclassified</td>
<td>?</td>
<td>5,028</td>
</tr>
</tbody>
</table>

- **Precondition Strengthening**: e.g. making method throw `UnsupportedOperationException`
- **Postcondition Weakening**: e.g. removing `@NonNull` method annotation
## RQ4: Are Contracts used Correctly over Inheritance?

<table>
<thead>
<tr>
<th>Change</th>
<th>Critical</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unchanged</td>
<td>no</td>
<td>351</td>
</tr>
<tr>
<td>Minor Change</td>
<td>no</td>
<td>193</td>
</tr>
<tr>
<td>Preconditions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(weaker)</td>
<td>no</td>
<td>40</td>
</tr>
<tr>
<td>(stronger)</td>
<td>yes</td>
<td>1,242</td>
</tr>
<tr>
<td>Postconditions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(stronger)</td>
<td>no</td>
<td>0</td>
</tr>
<tr>
<td>(weaker)</td>
<td>yes</td>
<td>0</td>
</tr>
<tr>
<td>unclassified</td>
<td>?</td>
<td>556</td>
</tr>
</tbody>
</table>

- **Stronger Precondition** in subclass violates LSP
- **Weaker Postcondition** in subclass violates LSP
Conclusion

Closet Contract Conjecture (Arnout & Meyer). *Programmers will encode contracts by whatever means available.*

- No evidence of *widespread* contract use
- If *Closet Contract Conjecture* holds, contracts are *hidden deeper*
- Projects which use contracts *continue* to do so and *expand* their use
- Found some cases of *incorrect* contract usage in context of evolution and inheritance
@WhileyDave
Example: Versioning Violation

// slf4j-api v 1.7.8
org.slf4j.LoggerFactory {
  ..
  @javax.validation.constraints.Nonnull
  public static ILoggerFactory getILoggerFactory()
  ..
}

// slf4j-api v 1.7.9
org.slf4j.LoggerFactory {
  ..
  public static ILoggerFactory getILoggerFactory()
  ..
}
Example: Versioning Violation

// commons-cli-1.0
org.apache.commons.cli.Option {
  ..
  public boolean addValue(String value) { .. }
  ..
}

// commons-cli-1.1
org.apache.commons.cli.Option {
  ..
  public boolean addValue(String value) {
    throw new UnsupportedOperationException( ..);
  }
  ..
}
Example: LSP violation

// from openjdk8-b132
java.beans.PropertyEditorSupport {
   ..
   public void setAsText(String text) {
      if (value instanceof String) {setValue(text); return;}
      throw new java.lang.IllegalArgumentException(text);
   }
   ..

// spring-beans-4.2.5.RELEASE
org.springframework.beans.propertyeditors.ResourceBundleEditor {
  ..
   public void setAsText(String text) {
      Assert.hasText(text, "'text' must not be empty");
   ..