Contracts in the Wild: a Study of Java Programs

Jens Dietrich[1] David J. Pearce[2] Kamil Jezek[3]
Premek Brada[3]

[1]Massey University, New Zealand
[2]Victoria University of Wellington, New Zealand
[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.”*

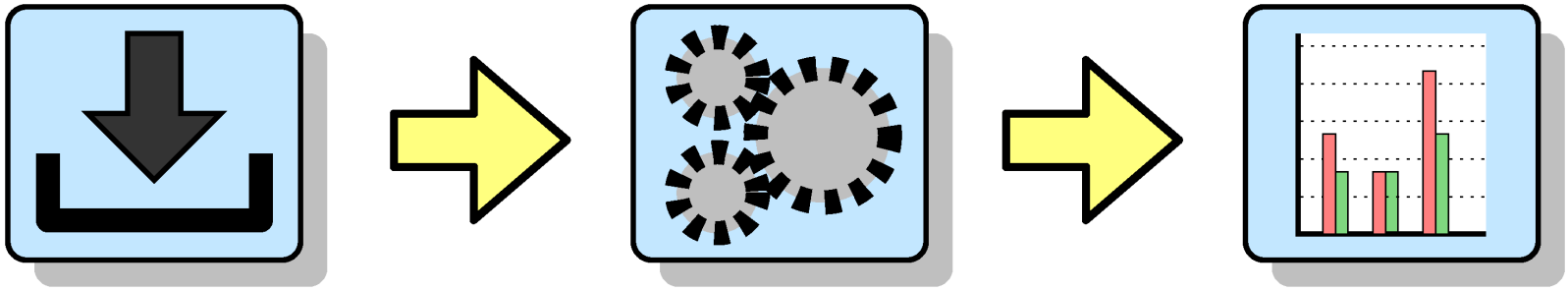
Alan Turing, 1949

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

Overview



Download & Filter ***Classification via Static Analysis*** ***Data Aggregation & Analysis***

- 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)**.

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

- **Runtime Exceptions (Unconditional)**.

```
throw new UnsuppportedOperationException();
```

- **Contract APIs**.

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

- **Assertions**.

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

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

```
void f(@NonNull String str) { ... }
```

Contract Classification

Contract Type	Classification
Runtime Exception (Conditional)	`precondition`
Runtime Exception (Unconditional)	`precondition`
Contract API	`precondition`
Assert	`unclassified`
Annotation (parameter)	`precondition`
Annotation (method)	`postcondition`
Annotation (field,class)	`class invariant`

Static Analysis

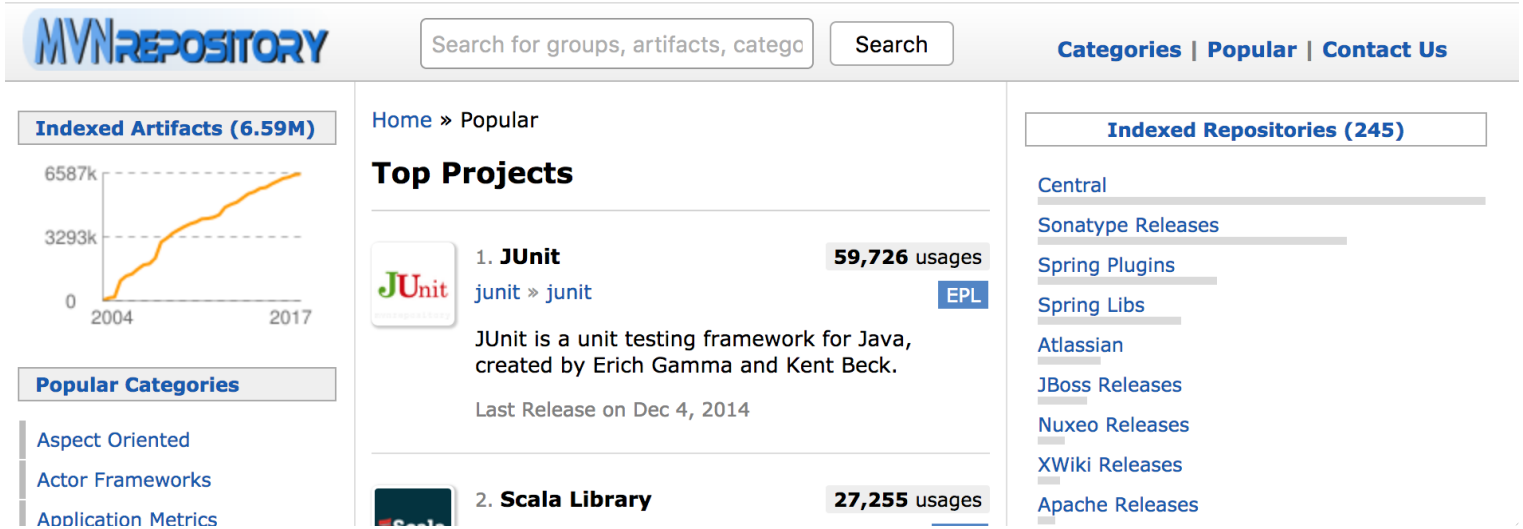
```
@Override
public void visit(ThrowStmt n, Object arg) {

  // 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?

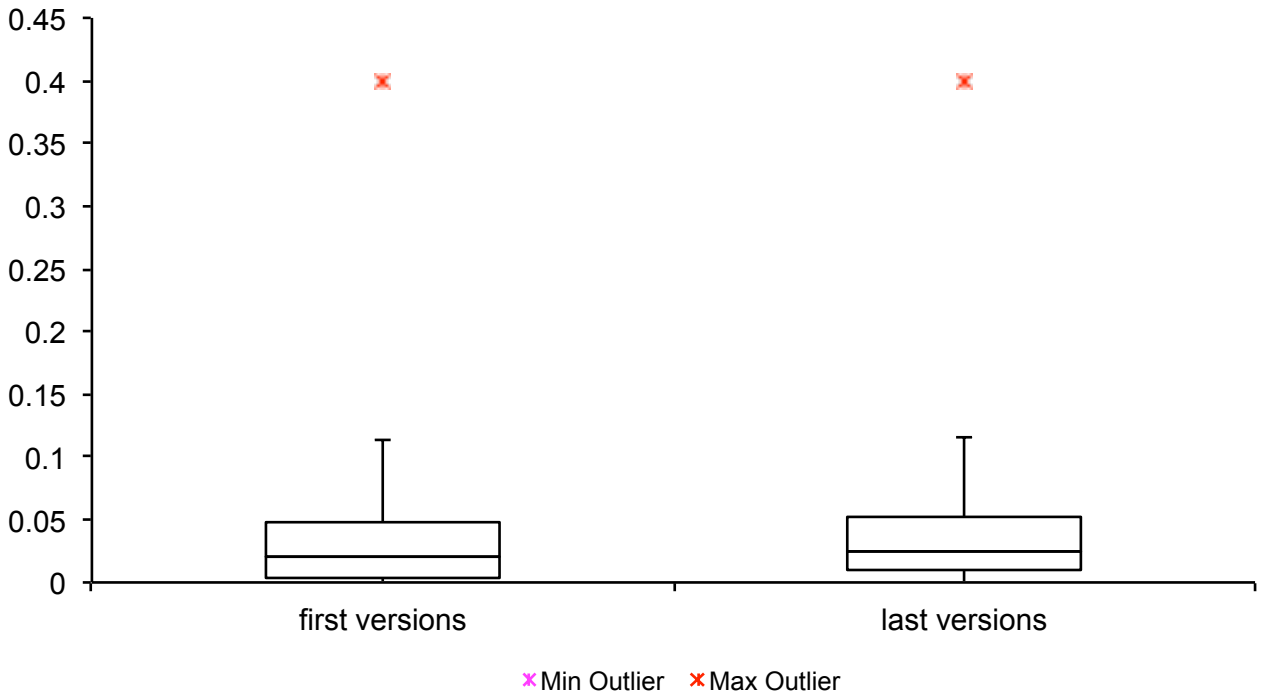
Category	**Notes**	**Constructs** (all versions)	**Constructs** (latest versions)	**programs**
Runtime Exception				
	(Conditional)	484,964	15,720	155
	(Unconditional)	123,966	3,084	122
Contract API				
	(Guava)	49,021	1,188	6
	(Spring)	100,232	2,148	13
	(Commons)	879	110	6
Assert		131,340	3,284	52
Annotation				
	(JSR303/349)	586	20	1
	(JSR305)	33,281	911	6
Precondition		786,723	22,969	160
Postcondition		2,413	112	6
Class Invariant		3,793	100	5
Unclassified		131,340	3,284	52

RQ1: Which Language Features used for Contracts?

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

- 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**

RQ2: How does Contract Use Change in Time?



- Generally speaking, if projects use contracts they **keep** using them
- Contract usage **increases proportionally** with program size

RQ3: Are Contracts used Correctly in Time?

Change		Critical	Count
Unchanged		no	652,395
Minor change		no	1,512
Preconditions			
	(weakened)	no	12,675
	(strengthened)	yes	2,777
Postconditions			
	(strengthened)	no	18
	(weakened)	yes	7
Unclassified		?	5,028

- **Precondition Strengthening**: e.g. making method throw `UnsupportedOperationException`
- **Postcondition Weakening**: e.g. removing `@NonNull` method annotation

RQ4: Are Contracts used Correctly over Inheritance?

Change	**Critical**	**Count**
Unchanged	no	351
Minor Change	no	193
Preconditions		
(weaker)	no	40
(stronger)	yes	1,242
Postconditions		
(stronger)	no	0
(weaker)	yes	0
unclassified	?	556

- **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");
      ..
```