The Unsafety in Java Regression Test Selection and its Occurrence in the Wild

Lukas Döllerer

Seminar: Software Quality

Informatics 4 - Software and Systems Engineering

June 12, 2022

What is the problem?

Over time, projects accumulate a lot of regression tests.

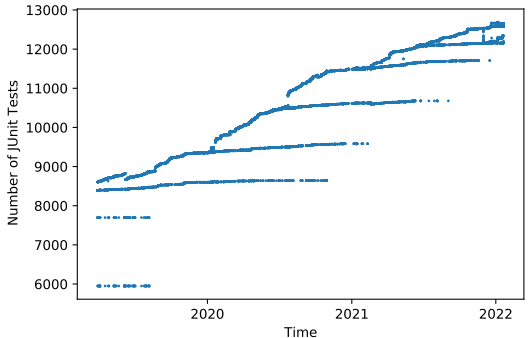


Figure: Spring Boot JUnit tests in the “master” branch.

Too many regression tests.

→ *RetestAll* not feasible

→ ⚠ Bugs / regression may be introduced.

How can we solve this problem?

Regression Test Minimization

Eliminate redundant test cases.

Regression Test Selection

Only run affected test cases.

Regression Test Prioritization

Fail fast — Maximize early fault detection.

[regression_testing_reduction]

Regression Test Selection (RTS)

1. **Dependency collection:** Which parts of the source code affect the test's result?
2. **Identification of affected tests:** Were any of these parts changed?

[**regression_testing_reduction**]

Safe Regression Test Selection

"[...] they select every test from the original test suite that can expose faults in the modified program." [**safety_paper**].

Additionally, we require that **safe** RTS does not otherwise interfere with testing.

Since 1997, a lot of RTS tools claimed to be **safe**. However, this *safety* is often only semi-formally proven for code-only changes.

RTS Safety — Not so easy?

Zhu and others presented RTSCHECK, a tool for testing RTS-Tools, in 2019.

They found **9 sources of unsafety** in 3 RTS-Tools that were expected to act safe in most conditions.

Supposedly **safe** RTS-Tools act unsafe in certain conditions.

[**unsafety_eval**]

Research Questions & Study Objects

Study Objects

Name	*Ekstazi*	*GIBstazi*	*OpenClover*	*STARTS*	*HyRTS*
Venue	ISSTA	ISSRE	—	ASE	ICSE
Dependency Collection Technique	Dynamic Runtime Instrumentation	Dynamic with Static Analysis	Static Instrumentation	Static Analysis	Dynamic Runtime Instrumentation

Table: Study Objects

Research Questions

RQ1 What sources of unsafety exist for the examined RTS-tools?

RQ2 What are the differences in safety of the examined tools, in the context of the previously identified sources of unsafety?

RQ3 How can potential for unsafety be automatically identified in code changes without dynamic program analysis?

RQ4 In which quantities do the identified sources of unsafety occur in real world software projects?

Evaluation of Sources of Unsafety

RQ1 & RQ2

Color Code

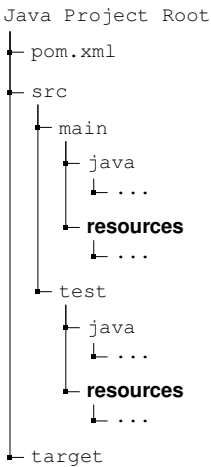
Conflicted with the proposed source of unsafety, the RTS-tool...

✓	...acts **safe**.
×	...acts **unsafe**.
⚠	...was unable to execute the tests.

Example:

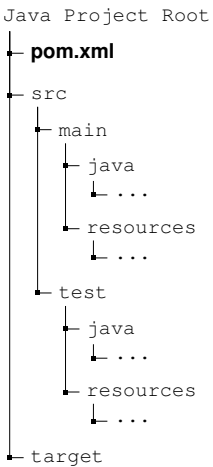
Ekstazi	✓
GIBstazi	✓
OpenClover	×
STARTS	×
HyRTS	⚠

External Files



Ekstazi	×
GIBstazi	✓
OpenClover	×
STARTS	×
HyRTS	×

Configuration Files




```
<version>2.11.4</version>
```

⇓

```
<version>2.12.0</version>
```

Ekstazi	×
GIBstazi	✓
OpenClover	×
STARTS	✓
HyRTS	×

Reflections

```
Class.forName("ModifiedClass");
```

Ekstazi	✓
GIBstazi	✓
OpenClover	✓
STARTS	×
HyRTS	×

Runtime Instrumentation

Ekstazi & *GIBstazi*:
Dynamic Bytecode Instrumentation incompatible with modern language features.

```
MyClass.class.getAnnotatedInterfaces();
MyClass.class.toGenericString();
```

OpenClover:
Source Code Instrumentation alters results of reflective method calls.

```
MyClass.class.getDeclaredClasses();
MyClass.class.getFields();
```

Ekstazi	×
GIBstazi	×
OpenClover	×
STARTS	✓
HyRTS	✓

Dependency Injection (1/3)

Dependency Source Code Changes - Spring Framework

```
@Bean
public BeanInterface beanName() {
    // change this implementation
    return new Implementation();
}
```

Ekstazi	✓
GIBstazi	⚠
OpenClover	⚠
STARTS	×
HyRTS	✓

Dependency Injection (1/3)

Dependency Source Code Changes - Guice Framework

```
@ImplementedBy(A.class)
public interface InterfaceA {
      void method();
}

class A {
   // change this source code
}
```

Ekstazi	✓
GIBstazi	✓
OpenClover	✓
STARTS	×
HyRTS	✓

Dependency Injection (2/3)

Collection Injection - Spring Framework

```
@Autowired
Collection<BeanInterface> injected;
```

⇒ Multiple Beans packed into a collection.

Ekstazi	×
GIBstazi	⚠
OpenClover	⚠
STARTS	×
HyRTS	×

Dependency Injection (2/3)

Collection Injection - Guice Framework

```
@ProvidesIntoSet
InjectedType impl1() {
   return Implementation();
}

@ProvidesIntoSet
InjectedType impl2() {
   return OtherImplementation();
}
```

⇒ Set of multiple implementations.

Ekstazi	✓
GIBstazi	✓
OpenClover	✓
STARTS	✓
HyRTS	×

Dependency Injection (3/3)

Classpath Scanning - Spring Framework only

```
@SpringBootApplication
public class MainSpringAppClass { }

@Configuration
@ComponentScan("example.beanConfig")
public class MainConfiguration { }
```

Ekstazi	×
GIBstazi	× / ⚠
OpenClover	× / ⚠
STARTS	×
HyRTS	×

Occurence of Unsafety in the Wild

RQ3 & RQ4

Occurence in the Wild

But, are these scenarios actually relevant?

Do these sources of unsafety occur in real-world software projects?

⇒ Search through the last 100 commits of 100 open source projects.

Occurence in the Wild

Study Object Selection

Choose the 100 public projects with the most “stars” from GitHub that...

- ...have the majority of their code written in Java
- ...use Maven as their build system
- ...were recently updated (at least once after the 06/01/2020)
- ...have at least 100 JUnit test cases (approximated)
- ...

Occurence in the Wild

Study Object Selection

A selection of the contained projects. (Collected between 11/11/2021 and 11/13/2021)

- Google Guava (42835 Stars)
- Apache Dubbo (36435 Stars)
- Jenkins (18057 Stars)
- Apache Flink (17519 Stars)
- Apache Hadoop (12081 Stars)
- Google Guice (10532 Stars)
- Apache Zookeeper (9962 Stars)
- JUnit 4 (8213 Stars)
- Signalapp Signal-Server (7210 Stars)
- Apache HBase (4260 Stars)

Occurence in the Wild

Commit Scanners

Scan of the last 100 commits of each project. Each source of unsafety was evaluated based on these questions.

External Files	**Dependency Injection**	**Runtime Instrumentation**	**Reflections**
Did the commiter change any external files?	Was any keyword related to dependency injection added / removed?	Does the source code contain any problematic methods?	1. Were reflective accesses introduced? 2. Were reflectively accessed classes changed?

Table: Commit Scanner Objectives

Commit Scanners — Pseudocode

Main Repository Scanner:

```
for commit in repo.traverse_commits(to=100):
   for scanner in commit_scanners:
         scanner.scan(commit)
```

Commit Scanner Module:

```
def scan(self, commit):
   for change in commit:
      if change_is_source_of_unsafety(change):
         self.unsafety_storage.add(commit)
```

Results

Results

Sources of Unsafety for specific RTS-Tools: RQ1 & RQ2

Source of Unsafety	*Ekstazi*	*GIBstazi*	*OpenClover*	*STARTS*	*HyRTS*
Dynamic Dispatch	✓	✓	×	✓	✓
External Files	×	✓	×	×	×
Configuration Files	×	✓	×	✓	×
Reflections	✓	✓	✓	×	×
Static Initializers	×	×	×	×	×
Dependency Injection (Spring)	✓	⚠	⚠	×	✓
Dependency Injection (Guice)	✓	✓	✓	×	✓
Collection Injection (Spring)	×	⚠	⚠	×	×
Collection Injection (Guice)	✓	✓	✓	✓	×
Classpath Scanning (Spring)	×	×	×	×	×
Runtime Instrumentation	×	×	×	✓	✓

Table: Test results from the PoC Repositories.

Results

Sources of Unsafety for specific RTS-Tools: RQ1 & RQ2

Source of Unsafety	*Ekstazi*	*GIBstazi*	*OpenClover*	*STARTS*	*HyRTS*
Dynamic Dispatch	✓	✓	×	✓	✓
External Files	×	✓	×	×	×
Configuration Files	×	✓	×	✓	×
Reflections	✓	✓	✓	×	×
Static Initializers	×	×	×	×	×
Dependency Injection (Spring)	✓	⚠	⚠	×	✓
Dependency Injection (Guice)	✓	✓	✓	×	✓
Collection Injection (Spring)	×	⚠	⚠	×	×
Collection Injection (Guice)	✓	✓	✓	✓	×
Classpath Scanning (Spring)	×	×	×	×	×
Runtime Instrumentation	×	×	×	✓	✓

Table: Test results from the PoC Repositories.

Results

Occurence of Sources of Unsafety in the Wild: RQ3 & RQ4

External Files Scanner

- 10% of the scanned commits change external files.
- They alter, on average, 1993.59 lines of text in an average of 11.85 files.
- These changes were only counted in the `resources` and `filters` folders.

Results

Occurence of Sources of Unsafety in the Wild: RQ3 & RQ4

Dependency Injection

- 53% of the projects use the Spring framework, 17% use Guice libraries.
- Detected 484 commits with Spring-related changes
- . . . and 297 commits with Guice-related changes.

Results

Occurence of Sources of Unsafety in the Wild: RQ3 & RQ4

Runtime Instrumentation

- 63% of the programs are possibly affected by dynamic or source code instrumentation.
- *Ekstazi* cannot execute code from 14 projects.
- *OpenClover* changes the behavior of code in 52 projects.

Results

Occurence of Sources of Unsafety in the Wild: RQ3 & RQ4

Reflections

- 56 projects use the `Class.forName(...)` method.
- A total of 333 occurrences of the `Class.forName(...)` method call.
- 72 changes on reflectively accessed classes were found.
  (⚠ High false negative rate.)

Results

Occurence of Sources of Unsafety in the Wild

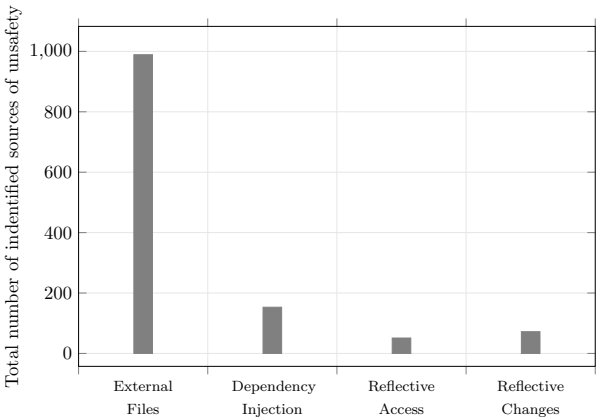


Figure: Number of detected sources of unsafety.

Conclusions

Conclusion

The Alarming State of RTS for Java

- All examined RTS-tools act in an unsafe manner.
- Latest versions of the tools cannot be used as **safe** RTS-tools.
- **Safe** RTS for Java is possible though, but it is not easy.

Conclusion

Maintenance of Research Projects

- Tools developed for research purposes are not maintained properly.
- Further research has to focus on improving existing tools.

Thank you for your attention

Time for Questions

Appendix

Bibliography I

Backup Slides

Sources of Unsafety

Approach to Simulating Szenarios

Each scenario that could lead to unsafe behavior of one of the tools is simulated.

Proof of Concept (PoC) Repositories

A **PoC Repository** contains a mini Java project that documents the steps required to reproduce the unsafe behavior.

Through Maven profiles, I can comfortably switch between tools without altering the configuration.

Dynamic Dispatch

Runtime selection of the most specific method in the inheritance chain.

Ekstazi	✓
GIBstazi	✓
OpenClover	×
STARTS	✓
HyRTS	✓

Static Initializers

```
class A {
   static {
      // Code with side-effects
   }
}

/* [...] */

Class.forName("A");
```

Ekstazi	×
GIBstazi	×
OpenClover	×
STARTS	×
HyRTS	×

Runtime Instrumentation

Keywords that cause unsafety.

Ekstazi	*OpenClover*
`getAnnotatedInterfaces()`	`getDeclaredClasses()`
`getAnnotatedSuperclass()`	`getDeclaredFields()`
`toGenericString()`	`getClasses()`
	`getFields()`

Table: Methods whose behavior changes with the corresponding RTS-tools runtime instrumentation. All methods are called on the meta class object of type `Class`.

Dependency Injection

Examined Frameworks:

- Spring
- Guice

Scenarios:

- Configuration in External File
- Configuration in Code
  - Dependency Source Code Changes
  - Collection Injection
  - Implicit Configuration
    ⇒ Class Path Scanning (Spring-only)

Dependency Injection

Configuration in External Files

When placing the configuration of dependencies into an external file, the primary cause of unsafety is the tool's inability to detect changes to this external file.

Ekstazi	×
GIBstazi	✓
OpenClover	×
STARTS	×
HyRTS	×

Keywords for Dependency Injections

Occurence of Sources of Unsafety in the Wild: RQ3 & RQ4

Spring Framework		Guice Framework	
Keyword	Regular expression	Keyword	Regular expression
`@Bean`	`@Bean`	`@AutoBindSingleton`	`@AutoBindSingleton`
`@Component`	`@Component`	`@Provides`	`@Provides`
		`@CheckedProvides`	`@CheckedProvides`
		`@ProvidesIntoSet`	`@ProvidesIntoSet`
		`@ProvidesIntoMap`	`@ProvidesIntoMap`
		`@ProvidesIntoOptional`	`@ProvidesIntoOptional`
		`bind(...)`	`bind(.*)`
		`LifecycleInjector`	`LifecycleInjector`

Table: Keywords used for the Dependency Injection Scanner

Results

Occurence of Sources of Unsafety in the Wild

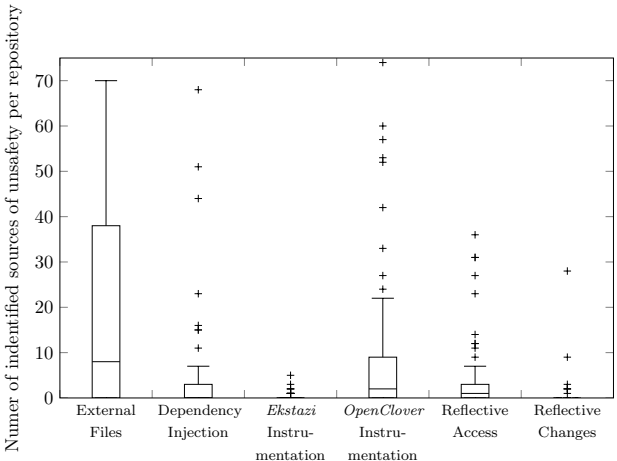


Figure: Sources of unsafety per repository (Removed outliers above 75 sources of unsafety. The full diagram is shown on the next slide.).

Results

Occurence of Sources of Unsafety in the Wild

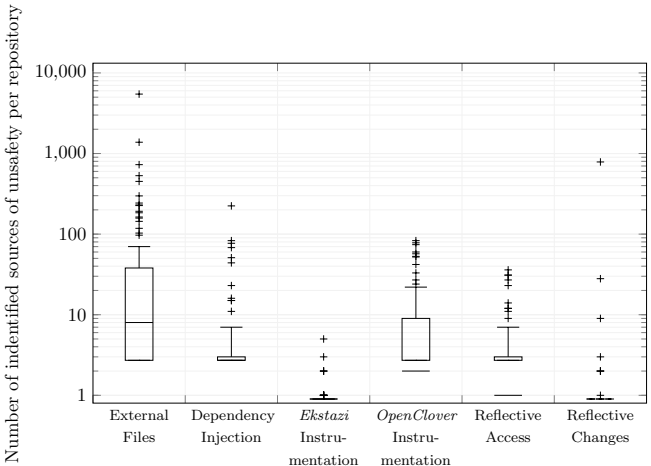


Figure: Sources of unsafety per scanned repository (Including all outliers).