Fixing broken robots - Android Mutation Testing

DroidConSG 2019

MERCARI

About me

- Matthew Vern
- Twitter @panini_ja
- Github panpanini
- Mercari, Inc
- Software Engineer (Android)

MERCARI

My job

- Client Engineer
- Solving problems for our customers
- Shipping features
- Improve existing functionality

MERCARI

My job

- A non-shipped feature doesn't provide benefit
- Ship features as quick as possible

MERCARI

My job

- A shipped, broken feature doesn't provide benefit
- Ship *quality* features as quick as possible

Maintaining quality

MERCARI

Maintaining quality

- QA

MERCARI

Maintaining quality

- QA
- Code Review

MERCARI

Maintaining quality

- QA
- Code Review
- Tests

MERCARI

Maintaining quality

- How do we know our tests are providing quality

Maintaining quality

- How do we know our tests are providing quality
- Use coverage to make sure that our tests are calling production code

Maintaining quality

- How do we know our tests are providing quality
- Use coverage to make sure that our tests are calling production code
  - changes introduced will not break existing code

Maintaining quality

- How do we know our tests are providing quality
- Use coverage to make sure that our tests are calling production code
  - changes introduced will not break existing code
  - new code does what it says on the tin

MERCARI

Who watches the watchmen?

MERCARI

Maintaining quality

How do we know that our *tests* are quality?

MERCARI

What are tests

MERCARI

What are tests

- asserting that our assumptions about a piece of code are correct
- binary assertions of code correctness

Lets fail some tests

MERCARI

Lets fail some tests

- Unit tests assert code behaviour
- change code behaviour
- tests fail
- ????
- profit

Mutation testing

MERCARI

Mutation testing

- proposed by Richard Lipton in 1971
- computationally expensive, not a viable testing solution until recently

Mutation testing steps

1. Create a mutant
2. Run test suite
3. Confirm if mutant was detected or not
4. Repeat

What is a mutant?

- A mutant is a biological entity which has undergone a change in its genetic structure.

What is a mutant?

- A mutant is a code block which has undergone a change in its structure.

Creating mutations

```
class SessionController(
    private val sessions: MutableList<Session>
) : EpoxyController() {

    fun setSessions(sessions: List<Session>) {}

    override fun buildModels() {}

    fun generateModels(sessions: List<Session>): List<SessionModel> {}
}
```

Creating mutations

```
fun setSessions(sessions: List<Session>) {
    this.sessions.clear()
    this.sessions.addAll(sessions)
    requestModelBuild()
}
```

MERCARI

Creating mutations

```
override fun buildModels() {
    generateModels(sessions)
        .forEach { it.addTo(this) }
}
```

MERCARI

Creating mutations

```
fun generateModels(sessions: List<Session>): List<SessionModel> {
    return sessions
        .map { session →
            SessionModel_()
                .title(session.title)
                .imageUrl(
                    if (session.speaker.profileImage ≠ "") {
                        session.speaker.profileImage
                    } else {
                        null
                    }
                )
        }
}
```

MERCARI

Creating mutations

MERCARI

Creating mutations

- Competent Programmer Hypothesis

MERCARI

Creating mutations

- Competent Programmer Hypothesis
- Coupling Effect

MERCARI

Conditionals boundary

Replaces relational operators with boundary counterpart

Original	Mutated
<	<=
<=	<
>	>=
>=	>

MERCARI

Conditionals boundary

```
// original
if (currentTime < startTime) {
  // do something
}

// mutated
if (currentTime ⩽ startTime) {
  // do something
}
```

MERCARI

Negate Conditionals

Negates conditional checks

Original	Mutated
==	!=
!=	==
<=	>
>	<=

MERCARI

```
// original
fun buildModels() {
  SessionModel_()
      .title(session.title)
      .imageUrl(
          if (session.speaker.profileImage ≠ "") {
              session.speaker.profileImage
          } else {
              null
          }
      )
}

// mutated
fun buildModels() {
  SessionModel_()
      .title(session.title)
      .imageUrl(
          if (session.speaker.profileImage == "") {
              session.speaker.profileImage
          } else {
              null
          }
      )
}
```

Remove void calls 😈

removes void method calls

MERCARI

Remove void calls 😈

```
// original
fun setSessions(sessions: List<Session>) {
    this.sessions.clear()
    this.sessions.addAll(sessions)
    requestModelBuild()
}

// mutated
fun setSessions(sessions: List<Session>) {
    this.sessions.clear()
    this.sessions.addAll(sessions)

}
```

MERCARI

So what?

MERCARI

Why mutation testing

- Code coverage, but better

MERCARI

Mutation testing

The better way

1. Introduce a fault into production code
2. Use code coverage to determine which tests to run
3. Run tests
4. Confirm if fault was detected or not
5. Repeat

That's a lot of work you expect us to do there bud

MERCARI

Pitest

pitest.org

Real world mutation testing

PIT is a state of the art mutation testing system, providing gold standard test coverage for Java and the jvm. It's fast, scalable and integrates with modern test and build tooling.

Get Started

User Group Issues Source Maven Central

MERCARI

Pitest

- pitest.org
- mutation testing system
- mutants stored in memory
- outputs pretty reports
- Gradle plugin 😻

Gradle plugin

- szpak/gradle-pitest-plugin
- `apply plugin: pitest`
- generates `pitest<Variant>` tasks

MERCARI

9. Can I use gradle-pitest-plugin with my Android application?

Short answer is: not directly. Due to some incompatibilities between "standard" Java applications and Android Java applications in Gradle the plugin does not support the later. Luckily, there is an Android fork of the plugin maintained by Karol Wrótniak which provides a modified version supporting Android applications (but on the other hand it doesn't work with standard Java applications).

Android Gradle plugin

- koral--/gradle-pitest-plugin
- written by Karol Wrótniak, forked from szpak/gradle-pitest-plugin
- works with Android projects
- has some Android specific helpers (eg: generating mockable Android jar)

Android Gradle plugin

```
plugins {
  id("pl.droidsonroids.pitest")
}

pitest {
    excludeMockableAndroidJar = false
    targetClasses = setOf("jp.co.panpanini.mypackage.*")
    outputFormats = setOf("XML", "HTML")
}
```

MERCARI

Android Gradle plugin

```
plugins {
  id("pl.droidsonroids.pitest")
}

pitest {
    excludeMockableAndroidJar = false
    targetClasses = setOf("jp.co.panpanini.mypackage.*")
    outputFormats = setOf("XML", "HTML")
}
```

MERCARI

Android Gradle plugin

```
plugins {
  id("pl.droidsonroids.pitest")
}

pitest {
    excludeMockableAndroidJar = false
    targetClasses = setOf("jp.co.panpanini.mypackage.*")
    outputFormats = setOf("XML", "HTML")
}
```

MERCARI

Android Gradle plugin

```
plugins {
  id("pl.droidsonroids.pitest")
}

pitest {
    excludeMockableAndroidJar = false
    targetClasses = setOf("jp.co.panpanini.mypackage.*")
    outputFormats = setOf("XML", "HTML")
}
```

MERCARI

Android Gradle plugin

```
plugins {
  id("pl.droidsonroids.pitest")
}

pitest {
    excludeMockableAndroidJar = false
    targetClasses = setOf("jp.co.panpanini.mypackage.*")
    outputFormats = setOf("XML", "HTML")
}
```

MERCARI

Android Gradle plugin

```
plugins {
  id("pl.droidsonroids.pitest")
}

pitest {
    excludeMockableAndroidJar = false
    targetClasses = setOf("jp.co.panpanini.mypackage.*")
    outputFormats = setOf("XML", "HTML")
}
```

Demo

MERCARI

Pitest tips & tricks

MERCARI

Pitest kotlin

- pitest/pitest-kotlin
- MutationInterceptor
- Removes mutants for Kotlin generated code

Run PITest on Unit tests only

MERCARI

Run PITest on CI

MERCARI

Mutation Testing

- panpanini/mutation_testing
- Github: panpanini
- Twitter: panini_ja

MERCARI