On the Distribution of Test Smells in Open Source Android Applications: An Exploratory Study

1. Introduction
Introduction

▸ A high-quality system need not be necessarily maintenance-friendly

▸ Systems built using poor design/coding practices can meet functional requirements

▸ In the long run, such events impact software maintenance - and maintenance is not cheap!
  ▸ Maintenance consumes 50% to 80% of resources
Towards maintenance-friendly code

- Researchers and industry have defined and created approaches and tools to detect code in need of refactoring
  - Design/code smells - Cohesion, Coupling, God Class, etc.
  - Tools - FindBugs, PMD, Checkstyle, etc.

- Smells make code harder to understand and make it more prone to bugs and changes

- Research and tools have been primarily on production code
Test Smells

- **Test code**, like production code, is subject to smells
- Formally introduced in 2001 with 11 smell types
- Inclusion of additional smell types through the years, analysis of their evolution and longevity, and elimination patterns
- **Tools** to detect specific smell types
- Studies on *traditional Java* applications
2.6 million apps available on Google Play as of Q4 2018
Objective

Insight into the *unit testing practices of Android app developers* with the aim of providing developers a mechanism to *improve unit testing code*
Contribution

- Expansion of Test Smell Types
- Open-Source Test Smell Detection Tool
- Understanding of Test Smells in Android apps
- Replication Package Availability
Research Questions

RQ 01
How likely are Android apps to contain unit test smells?
- Are apps, that contain a test suite, prone to test smells?
- What is the frequency and distribution of test smells in apps?
- How does the distribution of smell types in Android apps compare against traditional Java applications?

RQ 02
What is the general trend of test smells in Android apps over time?
- When are test smells first introduced into the project?
- How do test smells exhibited by the apps evolve over time?
2. Test Smells
Proposed Test Smells

- Conditional Test Logic
- Constructor Initialization
- Default Test
- Duplicate Assert
- Empty Test
- Exception Handling
- Ignored Test
- Magic Number Test
- Redundant Print
- Redundant Assertion
- Sleepy Test
- Unknown Test
Practicability

Are our proposed smells indicative of problems?

- 120 smelly unit test files
- 100 software systems
- 120 software developers
- 41.7% response rate
Conditional Test Logic

- Conditions within the test method will alter the behavior of the test and its expected output.
- Developers agree on the negative impact on code comprehension.
- However, outright removal may not always be applicable – decide on a “case by case basis”.

```
/* Test method contains multiple control statements */

@Test
public void testSpinner() {
    /* Control statement #1 */
    for (Map.Entry<String, String> entry : sourcesMap.entrySet()) {
        String id = entry.getKey();
        Object resultObject = resultsMap.get(id);
        /* Control statement #2 */
        if (resultObject instanceof EventsModel) {
            EventsModel result = (EventsModel) resultObject;
            /* Control statement #3 */
            if (result.testSpinner.runTest) {
                System.out.println("Testing " + id + " (testSpinner)");
                AnswerObject answer = new AnswerObject(entry.getValue(), "", new CookieManager(), "");
                EventsScraper scraper = new EventsScraper(RuntimeEnvironment.
                        application, answer);
                SpinnerAdapter spinnerAdapter = scraper.spinnerAdapter();
                assertEquals(spinnerAdapter.getCount(), result.testSpinner.dataSize());
                /* Control statement #4 */
                for (int i = 0; i < spinnerAdapter.getCount(); ++i) {
                    assertEquals(spinnerAdapter.getItem(i), result.testSpinner.
                            data.get(i));
                }
            }
        }
    }
}
```

“I actually have no idea why that for loop is there. It doesn’t do anything but run the test 1000 times, and there’s no point in that. I’ll remove it.”
Constructor Initialization

- Initialization of fields should be in the `setUp()` method (i.e., test fixtures)
- Most developers are aware of test fixtures
- Developers unanimously agree on using test fixtures
- Reasons for not using test fixtures include “laziness” and being “sloppy”

```
public class TagEncodingTest extends BrambleTestCase {
    private final CryptoComponent crypto;
    private final SecretKey tagKey;
    // ** Constructor initializing field variable ** */
    public TagEncodingTest() {
        crypto = new CryptoComponentImpl(new TestSecureRandomProvider());
        tagKey = TestUtils.getSecretKey();
    }
    @Test
    public void testKeyAffectsTag() throws Exception {
        for (int i = 0; i < 100; i++) {
            // Field variable utilized in test method */
            crypto encodeTag(tag, tagKey, PROTOCOL_VERSION, streamNumber);
            assertTrue(set.add(new Bytes(tag)));
        }
    }
```

“I have already made this change since you pointed it out so the code is clearer now”
Default Test

- Default test class meant to serve as an example
- Should either be removed
- A test-first approach will force developers to remove the file
- Unanimous agreement among developers that the file “serves no concrete purpose” and that it may lead to confusion

```java
/**
 * Default test class created by Android Studio **
 */
public class ExampleUnitTest {

/**
 * Default test method created by Android Studio **
 */
@Test
public void addition_isCorrect() throws Exception {
    assertEquals(4, 2 + 2);
}

/**
 * Actual test method **
 */
@Test
public void shareProblem() throws InterruptedException {
    Observable.just(200)
        .subscribeOn(Schedulers.newThread())
        .subscribe(begin.asAction());
    begin.set(200);
    Thread.sleep(1000);
    assertEquals(beginTime.get(), "200");
    .....}
}
```

“Removed useless example unit test”
The same condition is tested multiple times within the same test method.

The name of the test method should be an indication of the test.

Mixed responses - some developers preferred to split the assertion statement into separate methods.

"I might enforce it on some bigger projects."
Empty Test

- When a test method has no executable statements
- JUnit will indicate that the test passes even if there are no executable statements present in the method body
- Unanimous agreement among developers that such test methods should be removed from the test suite

```java
/** Test method without executable statements */
public void testCredGetFullSampleV1() throws Throwable {
    // ScrapedCredentials credentials = innerCredTest(FULL_SAMPLE_v1);
    // assertEquals("p4ssw0rd", credentials.pass);
    // assertEquals("user@example.com", credentials.user);
}
```

"Yes definitely should be removed"
Exception Handling

- Passing or failing of a test method is explicitly dependent on the production method throwing an exception
- Developers should utilize JUnit’s exception handling features to automatically pass/fail

```java
@Test
public void realCase() {
    ....
    /** ** Fails the test when an exception occurs ** **
    try {
        a.compute();
    } catch (CalculationException e) {
        Assert.fail(e.getMessage());
    }
    Assert.assertEquals("233.2405", this.df4.format(a.getResults().get(0).getUnknownOrientation()));
    ....
```
Ignored Test

- Ignored test methods result in overhead with regards to compilation time and an increase in code complexity and comprehension.
- Mixed responses - investigate problems or serve as a means for new developers “to understand behavior.”

```java
@Test
/** This test will not be executed due to the @Ignore annotation ** */
@Ignore("disabled for now as this test is too flaky")
public void peerPriority() throws Exception {
    final List<InetSocketAddress> addresses = Lists.newArrayList(
        new InetSocketAddress("localhost", 2000),
        new InetSocketAddress("localhost", 2001),
        new InetSocketAddress("localhost", 2002)
    );
    peerGroup.addConnectedEventListener(connectedListener);
    ...
}
```

“would not tolerate to have ignored tests in the code”
Magic Number Test

- Test method contains unexplained and undocumented numeric literals
- Developers agree that the use of constants over magic numbers improve code readability/understandability
- Not a blanket rule - a constant should only be used so that its “name adds useful information”

```
@Test
public void testGetLocalTimeAsCalendar() {
    Calendar localTime = calc.getLocalTimeAsCalendar(BigDecimal.valueOf(15.5D),
        Calendar.getInstance());
    /* ** Numeric literals are used within the assertion statement ** */
    assertEquals(15, localTime.get(Calendar.HOUR_OF_DAY));
    assertEquals(30, localTime.get(Calendar.MINUTE));
}
```

“ If the numerical value has a deeper meaning (e.g. flag, physical constant, enum value) then a constant should be used.”
Redundant Assertion

- Assertion statements that are either **always true or false**
- Common reason for the existence of this smell is due to **developer mistakes**
- Developers confirmed that such code "is not needed", "bad style" and "should probably be removed"
- Might exist to support edge cases
Redundant Print

- Unit tests are executed as part of an automated script
- They can consume computing resources or increase execution time
- Unanimous agreement that print statements do not belong in test suites
- A common reason for the existence of this smell is due to developer debugging

"a waste of resources (cpu+disk space)"
Explicitly causing a thread to sleep can lead to unexpected results as the processing time for a task differs when executed in various environments and configurations.

Developers confirmed that there are risks (i.e., inconsistent results) involved with causing a thread to sleep.

```
public void testEditExternalSearch() throws Exception {
    ....
    assertEquals("Searching", entry.english);
    // ** Forcing the thread to sleep ** */
    Thread.sleep(500);
    final Intent i2 = getStartedActivityIntent();
    ....
}
```

“the alternative requires more code”
The assertion statement helps to indicate the purpose of the test

JUnit will show the test method as passing

Majority of the developers are in favor of having assertion statements in a test method

Missing assertions were due to mistakes

“...It looks like just sloppy coding there. I'll look to fix that test..."
TS Detect

- **Open-source**, Java-based, static analysis
- Available as a **standalone jar** and requires a list of file paths as input
- Utilizes an **abstract syntax tree** to parse and detect test smells
- Detects **19** test smells (12 proposed + 7 existing)
- Average **F-Score of 96.5%**
3.
Experiment Methodology
Data Collection Phase

- **2,011 cloned apps**
- **1,037,236 commits**
- **6,379,006 java files affected by commits**
- **+3.5 GB java files collected**
Detection Phase

656 analyzed apps
206,598 detected test files
1,187,055 analyzed test methods
175,866 test files with 1 or more smells
4. Analysis & Discussion
RQ1 – Test Smell Occurrence

Test Smell Occurrence & Distribution

- 97% of the analyzed apps contained test smells
- *Assertion Roulette* occurred the most (in over 50% of the analyzed apps and test files)
- All smell types had a high co-occurrence with *Assertion Roulette*
- *Similar distribution* of test smells between Android and non-Android applications
RQ1 – Test Smell Occurrence

**Smell Type Distribution**

**Smell Type Occurrence**

<table>
<thead>
<tr>
<th>Smell Type</th>
<th>Smell Exhibition In Apps</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assertion Roulette</td>
<td>52.28%</td>
<td>58.46%</td>
</tr>
<tr>
<td>Conditional Test Logic</td>
<td>37.32%</td>
<td>28.67%</td>
</tr>
<tr>
<td>Constructor Initialization</td>
<td>20.47%</td>
<td>11.70%</td>
</tr>
<tr>
<td>Default Test</td>
<td>42.20%</td>
<td>0.32%</td>
</tr>
<tr>
<td>Duplicate Assert</td>
<td>31.81%</td>
<td>31.33%</td>
</tr>
<tr>
<td>Eager Test</td>
<td>42.99%</td>
<td>38.68%</td>
</tr>
<tr>
<td>Empty Test</td>
<td>16.38%</td>
<td>1.08%</td>
</tr>
<tr>
<td>Exception Handling</td>
<td>84.57%</td>
<td>49.18%</td>
</tr>
<tr>
<td>General Fixture</td>
<td>25.51%</td>
<td>11.67%</td>
</tr>
<tr>
<td>Ignored Test</td>
<td>15.28%</td>
<td>3.00%</td>
</tr>
<tr>
<td>Lazy Test</td>
<td>39.06%</td>
<td>29.50%</td>
</tr>
<tr>
<td>Magic Number Test</td>
<td>77.01%</td>
<td>34.84%</td>
</tr>
<tr>
<td>Mystery Guest</td>
<td>36.38%</td>
<td>11.65%</td>
</tr>
<tr>
<td>Redundant Assertion</td>
<td>12.91%</td>
<td>3.87%</td>
</tr>
<tr>
<td>Redundant Print</td>
<td>14.02%</td>
<td>0.92%</td>
</tr>
<tr>
<td>Resource Optimism</td>
<td>15.75%</td>
<td>9.79%</td>
</tr>
<tr>
<td>Sensitive Equality</td>
<td>21.10%</td>
<td>9.19%</td>
</tr>
<tr>
<td>Sleepy Test</td>
<td>12.60%</td>
<td>2.04%</td>
</tr>
<tr>
<td>Unknown Test</td>
<td>47.09%</td>
<td>34.38%</td>
</tr>
</tbody>
</table>
### RQ1 – Test Smell Occurrence

#### Smell Type Co-Occurrence

<table>
<thead>
<tr>
<th>Smell Type</th>
<th>ASR</th>
<th>CTL</th>
<th>CNI</th>
<th>DFT</th>
<th>EMT</th>
<th>EXP</th>
<th>GFX</th>
<th>MGT</th>
<th>RPR</th>
<th>RAS</th>
<th>SEQ</th>
<th>SLT</th>
<th>EGT</th>
<th>DAS</th>
<th>LZT</th>
<th>UKT</th>
<th>IGT</th>
<th>ROP</th>
<th>MNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASR</td>
<td>31%</td>
<td>9%</td>
<td>0%</td>
<td>1%</td>
<td>49%</td>
<td>13%</td>
<td>13%</td>
<td>1%</td>
<td>3%</td>
<td>11%</td>
<td>2%</td>
<td>54%</td>
<td>46%</td>
<td>37%</td>
<td>23%</td>
<td>3%</td>
<td>13%</td>
<td>52%</td>
<td></td>
</tr>
<tr>
<td>CTL</td>
<td>62%</td>
<td>18%</td>
<td>0%</td>
<td>2%</td>
<td>58%</td>
<td>14%</td>
<td>25%</td>
<td>2%</td>
<td>7%</td>
<td>9%</td>
<td>5%</td>
<td>44%</td>
<td>39%</td>
<td>33%</td>
<td>46%</td>
<td>6%</td>
<td>20%</td>
<td>40%</td>
<td></td>
</tr>
<tr>
<td>CNI</td>
<td>43%</td>
<td>44%</td>
<td>0%</td>
<td>1%</td>
<td>84%</td>
<td>12%</td>
<td>22%</td>
<td>1%</td>
<td>3%</td>
<td>3%</td>
<td>6%</td>
<td>32%</td>
<td>24%</td>
<td>24%</td>
<td>57%</td>
<td>2%</td>
<td>12%</td>
<td>18%</td>
<td></td>
</tr>
<tr>
<td>DFT</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>1%</td>
<td>99%</td>
<td>0%</td>
<td>23%</td>
<td>1%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>2%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>16%</td>
</tr>
<tr>
<td>EMT</td>
<td>69%</td>
<td>45%</td>
<td>10%</td>
<td>0%</td>
<td>42%</td>
<td>28%</td>
<td>8%</td>
<td>0%</td>
<td>0%</td>
<td>4%</td>
<td>1%</td>
<td>35%</td>
<td>32%</td>
<td>18%</td>
<td>100%</td>
<td>2%</td>
<td>2%</td>
<td>47%</td>
<td></td>
</tr>
<tr>
<td>EXP</td>
<td>58%</td>
<td>34%</td>
<td>20%</td>
<td>1%</td>
<td>15%</td>
<td>19%</td>
<td>1%</td>
<td>5%</td>
<td>6%</td>
<td>4%</td>
<td>35%</td>
<td>32%</td>
<td>32%</td>
<td>40%</td>
<td>3%</td>
<td>18%</td>
<td>39%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GFX</td>
<td>66%</td>
<td>35%</td>
<td>12%</td>
<td>0%</td>
<td>10%</td>
<td>1%</td>
<td>1%</td>
<td>1%</td>
<td>10%</td>
<td>3%</td>
<td>49%</td>
<td>42%</td>
<td>47%</td>
<td>43%</td>
<td>3%</td>
<td>8%</td>
<td>38%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MGT</td>
<td>67%</td>
<td>61%</td>
<td>22%</td>
<td>1%</td>
<td>79%</td>
<td>10%</td>
<td>1%</td>
<td>3%</td>
<td>5%</td>
<td>4%</td>
<td>42%</td>
<td>40%</td>
<td>29%</td>
<td>46%</td>
<td>2%</td>
<td>63%</td>
<td>42%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RPR</td>
<td>46%</td>
<td>74%</td>
<td>7%</td>
<td>0%</td>
<td>46%</td>
<td>19%</td>
<td>6%</td>
<td>1%</td>
<td>9%</td>
<td>1%</td>
<td>25%</td>
<td>22%</td>
<td>21%</td>
<td>61%</td>
<td>2%</td>
<td>5%</td>
<td>32%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RAS</td>
<td>48%</td>
<td>50%</td>
<td>8%</td>
<td>0%</td>
<td>70%</td>
<td>4%</td>
<td>10%</td>
<td>0%</td>
<td>2%</td>
<td>3%</td>
<td>46%</td>
<td>14%</td>
<td>40%</td>
<td>4%</td>
<td>8%</td>
<td>7%</td>
<td>40%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEQ</td>
<td>71%</td>
<td>28%</td>
<td>4%</td>
<td>0%</td>
<td>34%</td>
<td>13%</td>
<td>6%</td>
<td>1%</td>
<td>1%</td>
<td>2%</td>
<td>48%</td>
<td>44%</td>
<td>35%</td>
<td>20%</td>
<td>3%</td>
<td>3%</td>
<td>52%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SLT</td>
<td>60%</td>
<td>67%</td>
<td>36%</td>
<td>0%</td>
<td>100%</td>
<td>18%</td>
<td>20%</td>
<td>0%</td>
<td>5%</td>
<td>9%</td>
<td>48%</td>
<td>38%</td>
<td>31%</td>
<td>53%</td>
<td>5%</td>
<td>14%</td>
<td>26%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EGT</td>
<td>82%</td>
<td>33%</td>
<td>15%</td>
<td>0%</td>
<td>45%</td>
<td>15%</td>
<td>13%</td>
<td>1%</td>
<td>5%</td>
<td>11%</td>
<td>3%</td>
<td>46%</td>
<td>61%</td>
<td>19%</td>
<td>1%</td>
<td>11%</td>
<td>49%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DAS</td>
<td>86%</td>
<td>36%</td>
<td>9%</td>
<td>0%</td>
<td>51%</td>
<td>16%</td>
<td>15%</td>
<td>1%</td>
<td>2%</td>
<td>13%</td>
<td>2%</td>
<td>57%</td>
<td>44%</td>
<td>26%</td>
<td>3%</td>
<td>13%</td>
<td>60%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LZT</td>
<td>72%</td>
<td>32%</td>
<td>10%</td>
<td>0%</td>
<td>53%</td>
<td>19%</td>
<td>11%</td>
<td>1%</td>
<td>5%</td>
<td>11%</td>
<td>2%</td>
<td>79%</td>
<td>47%</td>
<td>26%</td>
<td>1%</td>
<td>9%</td>
<td>47%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UKT</td>
<td>39%</td>
<td>38%</td>
<td>19%</td>
<td>0%</td>
<td>15%</td>
<td>15%</td>
<td>16%</td>
<td>2%</td>
<td>0%</td>
<td>5%</td>
<td>3%</td>
<td>21%</td>
<td>24%</td>
<td>22%</td>
<td>7%</td>
<td>14%</td>
<td>25%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IGT</td>
<td>50%</td>
<td>53%</td>
<td>7%</td>
<td>0%</td>
<td>49%</td>
<td>10%</td>
<td>6%</td>
<td>1%</td>
<td>10%</td>
<td>8%</td>
<td>3%</td>
<td>19%</td>
<td>32%</td>
<td>13%</td>
<td>75%</td>
<td>6%</td>
<td>3%</td>
<td>35%</td>
<td></td>
</tr>
<tr>
<td>ROP</td>
<td>77%</td>
<td>60%</td>
<td>15%</td>
<td>0%</td>
<td>92%</td>
<td>10%</td>
<td>75%</td>
<td>0%</td>
<td>3%</td>
<td>3%</td>
<td>44%</td>
<td>41%</td>
<td>26%</td>
<td>48%</td>
<td>2%</td>
<td>45%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MNT</td>
<td>88%</td>
<td>33%</td>
<td>6%</td>
<td>1%</td>
<td>55%</td>
<td>13%</td>
<td>14%</td>
<td>1%</td>
<td>4%</td>
<td>14%</td>
<td>2%</td>
<td>55%</td>
<td>54%</td>
<td>40%</td>
<td>25%</td>
<td>3%</td>
<td>13%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Abbreviation:**
- ASR = Assertion Roulette
- CTL = Conditional Test Logic
- CNI = Constructor Initialization
- DFT = Default Test
- EMT = Empty Test
- EXP = Exception Handling
- GFX = General Fixtures
- MGT = Mystery Guest
- RPR = Redundant Print
- RAS = Redundant Assertion
- SEQ = Sensitive Equality
- SLT = Sleepy Test
- EGT = Eager Test
- DAS = Duplicate Assert
- LZT = Lazy Test
- UKT = Unknown Test
- IGT = Ignored Test
- ROP = Resource Optimism
- MNT = Magic Number Test
RQ2 – Test Smell Trend

Test Smell Introduction

- The first inclusion of a smelly file occurs approximately 23% of the way through the total app commits
- A test file is added with 3 smell types
- *Assertion Roulette* is the frequently the first smell type introduced
- Smells exhibited by a file remains constant throughout all updates to the file
5. Conclusion
Extended the catalog of known unit test smells

Open source test smell detection tool

A study of 656 Android apps showed a high prevalence of test smells in test suites

Smells are introduced early on into the codebase and exist during the lifetime of the app

Comprehensive project website: https://testsmells.github.io
Thanks!

https://testsmells.github.io