Psychology of Testing
(or why our intuition of testing is wrong)
-- Miško Hevery
Testing is not like frosting
Development Model

Software Engineers
- Develop
- Design
- Testing magic

QA
- Manual Tests
- Exploratory Tests

Test Engineers
- Automated Tests
- Tools
- Testing magic
Micro Development Model

Develop → Test → Check-In

automate
Excuses

Common Misconception

Valid Excuse

No Tests

- Legacy code-base
- “Dirties Design”
- It doesn’t catch bugs
- It's slower
- It's boring
- Hard to change
- I write UI
- Testing is for QA
- Too many interfaces

Don’t know how

Common Misconception
How do you write HARD TO TEST code?
Test Driver

Class Under Test

Stimulus

Asserts
Test Driver

Class Under Test
Object Lifetime and Calling

- **Object Instantiated**
- **Object Passed In**
- **Global Object**
Object Graph Construction & Lookup

Business Logic
Global State aka Singletons

• API that lies about what it needs
• Spooky action at a distance
Deceptive API

testCharge() {
    CreditCard cc;
    cc = new CreditCard("1234567890121234");
    cc.charge(100);
}

- At the end of the month I got my Statement!
- I was out $100!
- Spooky action at a distance!
- It never passed in isolation
Deceptive API

testCharge() {
    CreditCard cc;
    cc = new CreditCard(“1234567890121234”);
    cc.charge(100);
}

java.lang.NullPointerException
    at talk3.CreditCard.charge( CreditCard.java:48)
Deceptive API

testCharge() {
    CreditCardProcessor.init(...);
    CreditCard cc;
    cc = new CreditCard("1234567890121234");
    cc.charge(100);
}

java.lang.NullPointerException
   at talk3.CreditCartProcessor.init( CreditCardProcessor.java:146)
Deceptive API

testCharge() {
    OfflineQueue.start();
    CreditCardProcessor.init(...);
    CreditCard cc;
    cc = new CreditCard("1234567890121234");
    cc.charge(100);
}

java.lang.NullPointerException
    at talk3.OfflineQueue.start(OfflineQueue.java:16)
Deceptive API

testCharge() {
    Database.connect(...);
    OfflineQueue.start();
    CreditCardProcessor.init(...);
    CreditCard cc;
    cc = new CreditCard("1234567890121234");
    cc.charge(100);
}

• CreditCard API lies

• It pretends to not need the CreditCardProcessor even thought in reality it does.
Better API

testCharge() {
    db = new Database(...);
    queue = new OfflineQueue(db);
    ccProc = new CCProcessor(queue);
    CreditCard cc;
    cc = new CreditCard("12..34", ccProc);
    cc.charge(100);
}

• Dependency injection enforces the order of initialization at compile time.
Cost of Fixing the Bug

<table>
<thead>
<tr>
<th></th>
<th>Probability of Finding a Bug</th>
<th>Difficulty of Finding the Bug</th>
<th>Difficulty of Fixing the Bug</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical Bugs</td>
<td>HIGH</td>
<td>HARD</td>
<td>HARD</td>
</tr>
<tr>
<td>Wiring Bugs</td>
<td>MEDIUM</td>
<td>MEDIUM</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>Rendering Bugs</td>
<td>MEDIUM</td>
<td>EASY</td>
<td>EASY</td>
</tr>
</tbody>
</table>
But my code is different!
I have a Super-Bug!
Testing is like industrial revolution
Different Kinds of Testing

- **End-to-End**
  - Test the whole system pretending to be user
  - (>10 secs)

- **Functional**
  - (Wiring) Test interaction/contracts between classes
  - (<1 sec)

- **Unit**
  - (if) Test individual classes/methods in isolation
  - (~1ms)

### Execution Time
- Software Engineers
- Test Engineers
- Flakiness/Debugging/Maintenance Cost

### # of Tests
Dev Model Revisited

Education
- Testing on the Toilet
- Tech Talks / Blogs
- 1 on 1 Training
- Mercenaries
- Immersion

Tools
- Analysis
- Reports

Enforcement
- RoboCop
- Testability
- Coverage

CI-build
Check-In
Review

Games
- Dashboards
- Scoreboards
- Trend Graphs
Education: Test Mercenaries

• Internal consulting group specializing in testability refactoring and influencing developers

• 2-3 mercenaries join the group for 3-6 months
Education: Tech Talks

- Weekly series
- Focus on common mistakes which make code hard to test
  - new, global-state, singletons, LoD
- Many available on YouTube
Education:

- We try to stick it on everyone's monitor
- Set of red-flags to look for
- URL pointing to an explanation of the red-flag
- The reviewer can point the author to the explanation page

**Flaw #1: Constructor does Real Work**

**Warning Signs**
- New keyword in a constructor or at field declaration
- Static method calls in a constructor or at field declaration
- Anything more than field assignment in constructors
- Object not fully initialized after the constructor finishes
- Control flow (conditional or looping logic) in a constructor
- CL does complex object graph construction inside a constructor rather than using a factory or builder
- Adding or using an initialization block

Ask the CL author to read [golci-constructors](#)

**Flaw #2: Digging into Collaborators**

**Warning Signs**
- Objects are passed in but never used directly (only used to get access to other objects)
- Law of Demeter violation: method call chain walks an object graph with more than one dot (...)
- Suspicious names: context, environment, principal, container, manager

Ask the CL author to read [golci-digging](#)

**Flaw #3: Brittle Global State & Singletons**

**Warning Signs**
- Adding or using singletons
- Adding or using static fields or static methods
- Adding or using static initialization blocks
- Adding or using registries
- Adding or using service locators

Ask the CL author to read [golci-global-state](#)

**Flaw #4: Class Does Too Much**

**Warning Signs**
- Summing up what the class does includes the word “and”
- Class would be challenging for new team members to read and quickly “get it”
- Class has fields that are only used in some methods
- Class has static methods that only operate on parameters

Ask the CL author to read [golci-too-much](#)

Make suggestions, ask questions, or disagree [golci-field-guide](#)

Brought to you by the Green Beets of the Testability Corps [golci-beets](#)

Volume 1, Version 1.0
Education: Mission Impossible

- Mercenaries in reverse
- Total Immersion
- Work with team for a month which “gets it”
- Very good feedback
Weekly publications

Testing Tips (not concepts)

Captive audience

Very successful
Education: Blog

- Venue for larger concepts which do not fit to TotT.
- Shared with outside world
Education: TDD University

- Build small project from scratch
- Learn how the code should be like
- Rewrite existing piece of code
- To see how it is different
- Refactor existing code
- To learn how to get from here to there
Tools: Coverage

- Simple but effective
- Just having it installed makes people think about improving coverage
- Squeaky wheel gets the grease
Tools: Test Farms

- Farms
- Ability to run massive test suites in parallel
  - Selenium
  - unit-test
  - browser matrix
Tools: Testability Report

- Code coverage for testability
- Get high level view about how testable the code is
Tools: Testability Advice

Example class: SumOfPrimes1

```java
18: public class SumOfPrimes1 {
20:     private final Primeness primeness = new Primeness();
22:     public int sum(int max) {
23:         int sum = 0;
24:         for (int i = 0; i < max; i++) {
25:             if (primeness.isPrime(i)) {
26:                 sum += i;
27:             }
28:         }
29:         return sum;
30:     }
32: }
```

Class com.google.test.metric.example.Lessons.SumOfPrimes1 is hard to test because:

- Some collaborators cannot be mocked, so it is impossible to test this class in isolation from those classes.

Instances created with new: Why is it bad?  Contribution to class cost
- On line 25 boolean isPrime(int) is constructed 50%

Suggestion: pass an instance as a new constructor parameter, so that a unit test may substitute a different implementation that costs less.
Visibility: Games

Game leader board

<table>
<thead>
<tr>
<th>Participant</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Misko Hevery</td>
<td>205.0</td>
</tr>
<tr>
<td>dstalnov</td>
<td>89.0</td>
</tr>
<tr>
<td>Misko.Hevery</td>
<td>86.0</td>
</tr>
<tr>
<td>Misko.Hevery</td>
<td>86.0</td>
</tr>
<tr>
<td>aeagle22206</td>
<td>54.0</td>
</tr>
<tr>
<td>Jawolter</td>
<td>31.0</td>
</tr>
<tr>
<td>adan</td>
<td>28.0</td>
</tr>
<tr>
<td><a href="mailto:aeagle22206@gmail.com">aeagle22206@gmail.com</a></td>
<td>20.0</td>
</tr>
<tr>
<td><a href="mailto:tocmann@gmail.com">tocmann@gmail.com</a></td>
<td>13.0</td>
</tr>
<tr>
<td>chris.park</td>
<td>11.0</td>
</tr>
<tr>
<td>shyamseshadri</td>
<td>7.0</td>
</tr>
<tr>
<td>timothy_high</td>
<td>6.0</td>
</tr>
<tr>
<td>markusclermont</td>
<td>4.0</td>
</tr>
<tr>
<td><a href="mailto:kslundberg@gmail.com">kslundberg@gmail.com</a></td>
<td>3.0</td>
</tr>
<tr>
<td>tocmann</td>
<td>-5.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>User</th>
<th># Sub</th>
<th># Rev</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>corrysmith</td>
<td>31</td>
<td>97</td>
<td>⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐</td>
</tr>
<tr>
<td>jkasky</td>
<td>21</td>
<td>38</td>
<td>⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐</td>
</tr>
<tr>
<td>shyamseshadri</td>
<td>82</td>
<td>97</td>
<td>⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐</td>
</tr>
<tr>
<td>gtafft</td>
<td>18</td>
<td>24</td>
<td>⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐</td>
</tr>
<tr>
<td>pepper</td>
<td>45</td>
<td>78</td>
<td>⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐</td>
</tr>
<tr>
<td>alexeagle</td>
<td>22</td>
<td>49</td>
<td>⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐</td>
</tr>
<tr>
<td>jbakr</td>
<td>14</td>
<td>15</td>
<td>⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐</td>
</tr>
<tr>
<td>misko</td>
<td>30</td>
<td>42</td>
<td>⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐</td>
</tr>
<tr>
<td>arodriques</td>
<td>5</td>
<td>2</td>
<td>⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐</td>
</tr>
<tr>
<td>limpan</td>
<td>0</td>
<td>1</td>
<td>⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐</td>
</tr>
<tr>
<td>kworces</td>
<td>0</td>
<td>3</td>
<td>⭐⭐⭐⭐⭐⭐⭐⭐⭐⭐</td>
</tr>
</tbody>
</table>

- Make it into a game
Visibility: Bubbles

- See quality of check-ins over time
- Time, Tests, User, Size
Visibility: Test Pareto

- Test ratios per developer
- Impact per developer
Visibility: Test Trend

- Test focus over time
Visibility: Certification

- Well defined standards
- To be at N level you need to do X, Y, and Z.
- TC level prestige
Enforcement: Tough Love

- Remove manual testing resources
- Provide knowledge and support
Enforcement: RoboCop

- Coverage can not get worse with any one CL
- Testability can not get worse with any one CL
- Plans for arbitrary rule enforcement
Progress

- Too many tests too run
- CI standard
- Everyone agrees tests are good idea
- TC Level
- Dependency Injection - GUICE
- Ratio System vs Unit is improving