Miško Hevery

Influencing Software Developers Towards More Testable Code Base & More Tests
Question

• Which of you are trying to change your organization?

• Where is your organization with respect to unit-testing?

• Do you TDD?

• How do you get started?
Development Model

- Develop
- Test
- Check-In

automate
Excuses

Valid Excuse

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

Common Misconception
Interview Question

How do you write HARD TO TEST code?
Development Model

Software Engineers

Develop

Design

Testing magic

QA

Manual Tests

Test Engineers

Automated Tests

Tools

Testing magic

GeeCON
Let's move the Java world!
Dev Model Revisited

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

Tools
- Analysis
- Reports

Enforcement
- RoboCop
- Testability
- Coverage

Tools
- CI-build
- Check-In

Games
- Dashboards
- Scoreboards
- Trend Graphs

Visibility
• Internal consulting group specializing in testability refactorings 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: Reviewers Guide

- We try to stick it on everyones 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
Education: Mission Impossible

- Mercenaries in reverse
- Total Immersion
- Work with team for a month which “gets it”
- Very good feedback
**Education: TotT**

- Weekly publications
- Testing Tips (not concepts)
- Captive audience
- Very successful

---

**Testing on the Toilet**

**Be an MVP of GUI testing**

by Geoff Hardy (Irvine, CA)

With all the sport drug scandals of late, it's difficult to find good role models these days. However, when your role model is a Domain Model (object model of the business entities), you don't need to cheat to be an MVP—Use Model-View-Presenter!

MVP is very similar to MVC (Model-View-Controller). In **MVC, the presentation logic is shared by Controller and View**, as shown in the diagram below. The View is usually derived directly from visible GUI framework component, observing the Model and presenting it visually to the user. The Controller is responsible for deciding how to translate user events into Model changes. In **MVP**, presentation logic is taken over entirely by a Supervising Controller, also known as a Presenter.

![MVC vs MVP diagram](image)

The View becomes passive, delegating to the Presenter.

```java
public CongressionalHearingView() {
    testimonyWidget.addModifyListener(new ModifyListener() {
        public void modifyText(ModifyEvent e) {
            presenter.cnModifyTestimony(); // presenter decides what action to take
        }
    });
}
```

The Presenter fetches data from the Model and updates the View.

```java
public class CongressionalHearingPresenter {
    public void cnModifyTestimony() {
        model.parseTestimony(view.getTestimonyText()); // manipulate model
    }
    public void setWitness(Witness w) {
        view.setTestimonyText(w.getTestimony()); // update view
    }
}
```

This separation of duties allows for more modular code, and also enables easy unit testing of the Presenter and the View.

```java
public void testSetWitness() {
    spyView = new SpyCongressionalHearingView();
    presenter = new CongressionalHearingPresenter(spyView);
    presenter.setWitness(new Witness("Mark McGwire", "I didn't do it"));
    assertEquals("I didn't do it", spyView.getTestimonyText());
}
```

Note that this makes use of a perfectly legal injection — Dependency injection.
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 thing 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
public class SumOfPrimes1 {
  private final Primeness primeness = new Primeness();
  public int sum(int max) {
    int sum = 0;
    for (int i = 0; i < max; i++) {
      if (primeness.isPrime(i)) {
        sum += i;
      }
    }
    return sum;
  }
}
```

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.

- Personal testability advisor
- Plans to integrate into IDE
- Virtual pairing buddy
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>adam</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>

- 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
Q&A?
Links

- http://googletesting.blogspot.com/
  - http://misko.hevery.com/
- http://code.google.com/p/testability-harvester/