Testing @ Google

Andreas Leitner

ETH Zurich, 9.12.2009

Thanks to Robert Nielson, Christopher Semturs, and Patrick Copeland
Why do we test?
Why do we test?

... to find bugs
Why do we test?

... to find bugs early
Why do we test?

... to find bugs early

... to improve quality and maintainability
Why do we test?

... to find bugs early

... to improve quality and maintainability

not religious about it
What about quality assurance?

- No exclusive effort
- Fault tolerance
  - Hides failures, obstacle to correctness
  - Compensation failure monitoring
What about quality assurance?

• No exclusive effort
• Fault tolerance
  • Hides failures, obstacle to correctness
• Compensation failure monitoring

not covered in talk
How much testing is enough?

- Coverage?
- Tells you only when there are too few tests
- Does not tell you when there are enough
How much testing is enough?

**Cost**
- creation
- execution
- maintenance

**Benefit**
- Checks a property
How much testing is enough?

Cost
- creation
- execution
- maintenance

Benefit
- Checks a property

False Positives
Overview

• People, Process, and Tools
• Testability
If people don't think they have the power to solve their problems, they won't even think about how to solve them.

“Rules for Radicals” by Saul Alinsky
Traditional organization

Engineering

Quality assurance
Engineering

Productivity engineering
Test engineering roles

- Green berets
- Test certified mentors
- Software engineers in test
Test engineering roles

Productivity engineering

- Green berets
- Test certified mentors
- Software engineers in test

Engineering
Test engineering roles

- Productivity engineering
- Test engineer
- Engineering
Software engineers

- Create applications
- Responsible for quality
- Write tests

Test engineer

- Support developers
- Automate
- Provide infrastructure & tools
Software engineers

• Create applications
• Responsible for quality
• Write tests

Test engineer

• Support developers
• Automate
• Provide infrastructure & tools

Reliability engineers

• Keep production systems running
Test engineers
Test engineers
Evangelism and Education

• Testing blog
• Testing on the toilette
• Noogler training
Evangelism and Education

Google Test Automation Conference

www.gtac.biz
Process

- Bottom-up culture
- Many small teams
- Some, but not much process imposed
- Mostly affects how code
  - is checked-in
  - and tested
Code base

- Ownership
- Anybody may change anything
- Code reviews & style guide
- No branching, continuous integration
Code reviews

- Every change reviewed peer(s)
- Tool support
- Peers provide detailed comments
- Can take several iterations
- “Looks good to me”
Code reviews

• More eyes see more
• Make use of developer pride
• Spreads knowledge
• Makes “change anything policy” feasible
When tests are run

- Before every check-in: small minutes
- Continuously: small-large hours
- On demand: very large, load tests days
Test sizes

small

tests behavior of units in isolation

medium

tests the interaction of units

large

tests whole system
Test sizes

**small**
- tests behavior of units in isolation
- xUnit

**medium**
- tests the interaction of units
- xUnit

**large**
- tests whole system
- xUnit Selenium
Validation testing

- Few manual testers
- Alpha testing: Dogfooding
- Beta testing: A/B testing
Tools

- We use many tools, libraries and frameworks
- Internal and external
- Most are open source
- Hiring early adopters to enable quick adoption
Open Source

- Selenium
- Web-driver
- gUnit, jUnit, ...
- FindBugs
- Valgrind / Helgrind
- jMock, EasyMock, Mockito
- Guice
Example: Selenium farm

- Application under Test (eg: Writely, Gmail)
- Selenium Test
- Continuous Build
- Stubby
- Selenium Farm Client
- Selenium Farm Harness Live Servers
- Make it larger!
- Test Machines
  - Selenium Remote Control
  - Selenium Remote Control
  - Selenium Remote Control
Testability in 5 slides

Slide 1
Guidelines for testable software

- Should not use static methods or fields
- Favor dynamic dispatch over conditionals
- Follow law of demeter
- Should not use preconditions
- Avoid mutable state
- Should not use registries
- Wrap external APIs
- Should not use service locators
- Favor composition over inheritance
- Use dependency injection instead of concrete class instantiation

- Keep methods small
- Be exception safe
- Make methods virtual by default
- Don't use sealed unless you have to
- Create an interface per class
- Aim for at least 80% branch coverage
- Write code in a test driven manner
- Constructor should not do real work
- ...

Slide 2
Guidelines for testable software

- Should not use static methods or fields
- Favor dynamic dispatch over conditionals
- Follow law of demeter
- Should not use preconditions
- Avoid mutable state
- Should not use registries
- Wrap external APIs
- Should not use service locators
- Favor composition over inheritance
- Use dependency injection instead of concrete class instantiation

- Keep methods small
- Be exception safe
- Make methods virtual by default
- Don't use sealed unless you have to
- Create an interface per class
- Aim for at least 80% branch coverage
- Write code in a test driven manner
- Constructor should not do real work
- ...

Slide 2

Is it that complicated?
Why should software be testable?

- So we can write tests for it!
- Testable software can be executed outside of its original context
What is testable software?

Testable software is re-usable
Re-useable software

... is weakly coupled
... is highly cohesive

... has lean and stable interfaces
Re-useable software is testable