

# How to get to “DONE”

## Agile and Automated Testing Techniques and Tools

Grig Gheorghiu

[grig@gheorghiu.net](mailto:grig@gheorghiu.net)

[agiletesting.blogspot.com](http://agiletesting.blogspot.com)

XPSocal Meeting, Irvine, May 20th 2009

# What is testing?

- Making sure the application does the RIGHT things RIGHT (Uncle Bob's definition)
- The activity of finding bugs
  - What is a bug?
- Are QA and testing the same thing?
  - What is 'quality'? “Value to some person.” -- Jerry Weinberg
  - No “throwing code over the wall” to QA

# What is agile testing?

- 2 words: rapid feedback
- Short feedback loop between testers and developers
  - Find critical bugs early rather than late
  - Testers are service providers, not last line of defense
- Involve customers early in writing tests
- Focus on automation

# What is automated testing?

- Testing can be dull when repetitive
  - Enter scripting and automation
- Automated vs. automation-aided testing (agile vs. context-driven)
- Developers running unit tests before check-in
- Tests running unattended in continuous integration tool
- Testers using their investigative talents in performing exploratory testing

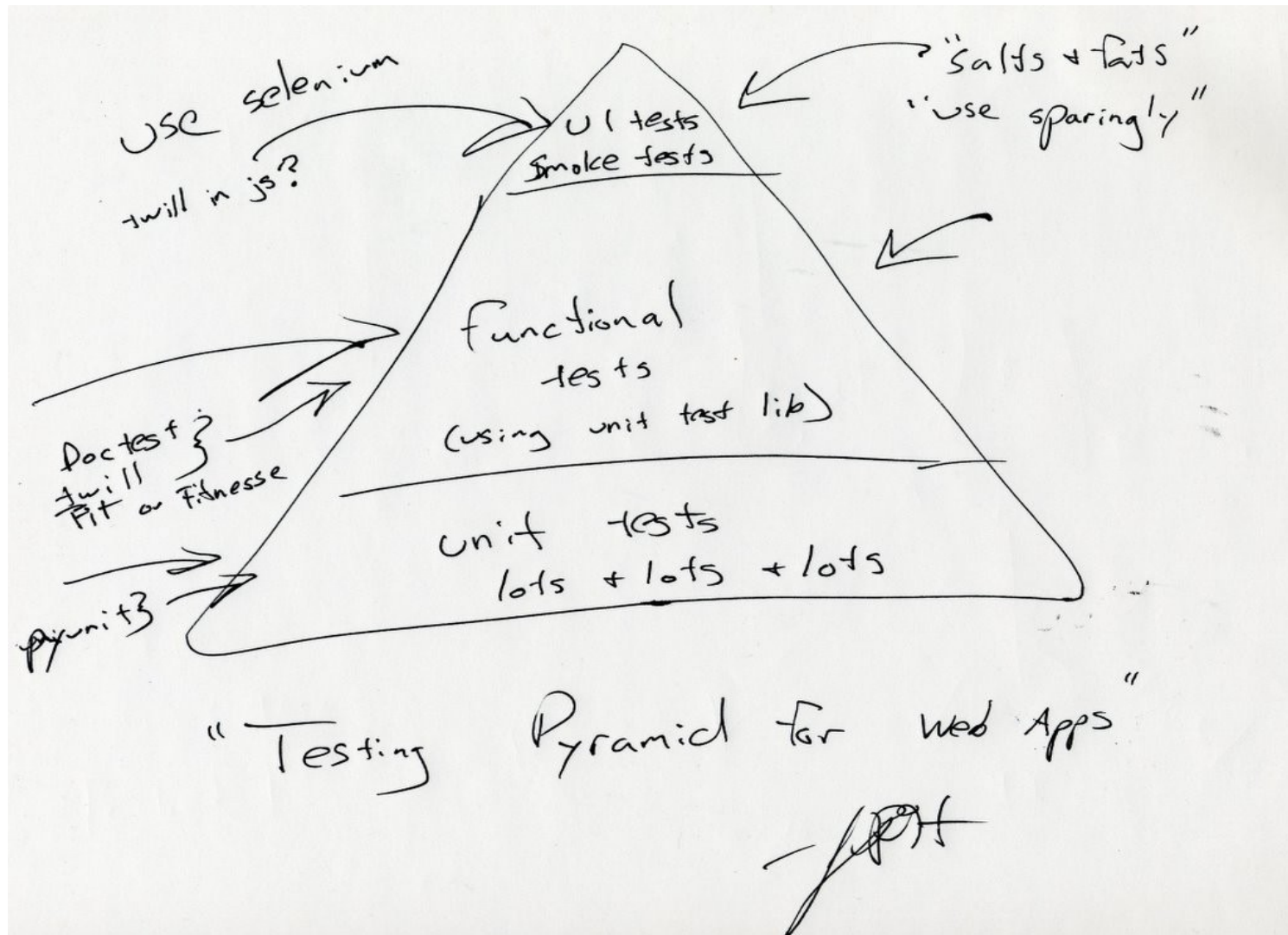
# What is automated testing? (cont.)

- Smoke tests
- Regression tests
- Long-running tests (can expose memory leaks)
- Test data generation

# Agile and automated testing is HARD

- Agile processes require discipline
  - Very easy not to write tests
  - Very easy to see test writing as a waste of development time
- Need somebody in charge of CI/automated testing processes
  - Do not let test results go red
  - Broken windows theory
  - Bitrot (Pybots example)

# Types of testing – Jason Huggins's Testing Pyramid



# Unit testing

- Test one function/method at a time
- **“Doing things RIGHT”**
- Written by developers; need to be fast
- Use unit test frameworks
  - setup/teardown methods
  - automatic test discovery
- Best bang for your automated testing buck
- Safety net for merciless refactoring
- Great way to learn 3<sup>rd</sup> party libraries

# Mock testing

- Simulate application interaction with external resources not under your control
- Eliminate data randomness
- Simulate exceptions and hard-to-reproduce errors
- Examples: databases, web services, weather data
- Mocks vs. stubs (expectations vs. canned data)
  - Great article by Martin Fowler

# Functional testing

- More coarse-grained than unit tests
- Test at the requirement/story level
- Test various paths through the application
- Written by testers and/or developers
- Still “**doing things RIGHT**”
- GUI or command-line

# Acceptance testing

- Similar to functional testing... but
- Business-facing (vs. code-facing)
- **“Doing the RIGHT things”**
- Ideally brings together customers/business analysts, developers and testers
- Agile aspects
  - Executable requirements
  - Executable documentation

# Integration testing

- End-to-end testing
- Tracer bullet development and testing
  - Candle making
- Tied closely to Continuous Integration processes and tools
- Single platform vs. multi-platform/OS/environment (buildbot)

# Performance/load/stress testing

- Performance testing – not about finding bugs
  - Eliminate bottlenecks, establish baselines
  - Careful analysis and measurement (“white box”)
  - Needs well defined expectations
- Load testing – test system behavior under constant load (volume, endurance)
  - Throughput, large data sets (“black box”)
- Stress testing – test system behavior under very high load
  - Does the system fail and recover gracefully?

# Deployment testing

- We live in a Web 2.0 world
  - Users want instant gratification
- Some Web apps have multiple deployments per day; cloud computing make it easy
- Sysadmins need to be testers too
  - Most stop at step #1 (automation)
  - Need to go to step #2 (testing the deployment)
- Chuck Rossi, Release Engineer, Facebook: “As an engineer, you will probably never work in an environment that's quite like Facebook. I can pretty confidently say that there is no other place where changes you make in the code will be in front of your mom and 175 million other people in less than 60 minutes.”

# Other types of testing

- James Bach and Michael Bolton: CRUSPIC 'ilities' testing
  - Capability
  - Reliability
  - Usability
  - Scalability
  - Performance
  - **Installability** (critical; if your app doesn't install, nothing else matters...)
  - Compatibility
- Security – a field in itself

# Fuzz testing

- Fuzz testing: a method for discovering bugs or faults in programs by providing them with random or malicious data as input, and monitoring them for exceptions, segfaults or other errors.
- Wide range of fuzzing tools
  - Throw random data at target application
  - Modify valid samples of input data in various ways
  - Modify data packets for specific protocols
  - Mutate code and run unit test (Jester)
- Easy to run, can find unexpected bugs

# Methods of testing

- Black box vs. white box
- GUI vs. business logic
- Code-facing vs. business facing (Brian Marick)
- Smoke testing

# White/black/gray-box testing

- White-box testing: knowledge of application internals
  - Show me the code
  - Unit, functional, integration tests
  - Knowledge of critical areas to probe
  - Knowledge of bug clusters
- Black-box testing: probe external interfaces of application
  - Apply inputs, observe outputs and behavior
  - Mirrors the user experience
  - Can be at GUI level or command-line level
- Gray-box testing: some knowledge of app. Internals
  - Fine-tune inputs, use testing hooks

# GUI vs. business-logic testing

- Testing at the GUI level
  - Mirrors user experience
  - Gives warm and fuzzy feeling to customers
  - Tests are brittle, need to be changed frequently
  - Still need to test underneath
- Testing at the business-logic level
  - Exercise the logic of your application
  - Did the widget really get added to the order in the DB?
- “Build-operate-check” pattern
  - Build test data, operate against it, check expectations
  - FitNesse: another thin GUI into the business logic

# GUI vs. business-logic (cont.)

- For business-logic level tests to be effective, developers need to provide clear ways to access the internal application logic
- Clean separation between GUI layer and business-logic layer
- MVC pattern
- Testing hooks into internal application functionality can evolve into API (web services)
- Both GUI and business-logic tests are important and necessary (Selenium vs. FitNesse)

# Improving code testability

- Scriptable interfaces into the application -- can evolve into API
- Hidden options for rich debugging output (--pizza)
- Logging
- Clear error and exception messages

# Code-facing vs. business-facing testing

- Code-facing tests
  - Use the language of programmers
  - Low level
  - Unit and functional tests
- Business or customer-facing tests
  - Expressed in the business domain language of the application
  - Higher-level acceptance tests
  - Foster collaboration between customers and developers facilitated by testers

# Smoke testing

- Plug the board in, see if smoke comes out
- “Kick the tires” testing
- Tests if the application works AT ALL
- Tests most common paths through the application
  - can users log in?
  - can users add items to the shopping cart?
- Tests paths most visible by customers
- A build is deemed as failed if it doesn't pass the smoke tests

# Automated testing strategies

- Where do you start? Where you feel the most pain
- TDD vs. TED
- Titus Brown: “I do stupidity-driven testing. When I do something stupid, I write a test to make sure I don't do it again.”
- Add an automated tests when fixing a bug
- Michael Feathers: legacy code has no unit tests

# Is automated testing necessary?

- 2<sup>nd</sup> law of automated testing: “If you ship versioned software, you need automated tests”
  - Regression testing
  - Smoke testing
  - Safety net for refactoring
- No automated tests == nauseating feeling in pit of stomach upon every launch
  - Web 2.0 world: launches can happen daily
  - Continuous deployments: “fear is the mind killer” post by Eric Ries: <http://startuplessonslearned.blogspot.com/2009/05/fear-is-mind-killer.html>

# Is automated testing necessary? (cont.)

- Some exceptions
- Throw-away Web sites (event-based, temporary)
- Most notable: Linux kernel
  - But ESR's law applies: “Given enough eyeballs, all bugs are shallow”

# No automated testing?

## Welcome to the Software Testing Death Spiral

Titus Brown: what happens to projects that lack both automated tests and an exponentially increasing team of testers?

1. They manually test the new features and bug fixes that they've just added.
2. They release their software.
3. Their software breaks in unexpected locations, bug reports are filed, and (optimistically) those bugs are fixed. Go to #1.

# Test code needs to be maintainable

- Automated test suites need to be maintained just like production code
- Bitrot happens quickly
- Don't underestimate effort to maintain test suites
  - GUI-level test suites are especially brittle and will need to be changed often

# Continuous Integration

- Critical piece of development and testing infrastructure
- Automated builds and automated test runs
- Start small, iterate (candle making again)
- Run fast tests upon each check-in
- Run slower tests periodically (at least once/day)
- Invaluable for finding OS/platform/environment-specific issues
  - Unit tests might work in individual developer environments, but could break in a clean env.

# Continuous Integration (cont.)

- Invaluable for finding OS/platform/environment-specific issues
  - Unit tests might work in individual developer environments, but could break in a clean env.
- Server-based vs. client-based applications
  - Single OS vs. multiple OS
  - Web application vs. Python interpreter
- Pybots – set of build slaves running tests upon check-ins to the main Python interpreter

# Virtualization

- Essential for testing in different types of environments
  - Development, staging, production
  - Single OS vs. multiple OS
- Cloud computing makes things easier, as long as you automate
- Development and staging environments may involve mock testing techniques (random data)
- NOTHING is like production (application melting under production traffic)

# Code coverage

- Shows lines of code executed during tests
- Code coverage tests your tests! (Ned Batchelder <http://us.pycon.org/2009/conference/schedule/event/26/>)
- **Necessary but not sufficient** for your testing
- 100% code coverage is hard to achieve
  - 80/20 rule applies: the last 20% of code coverage will require 80% of your effort
  - Look at trends, not absolute numbers
- Current tools can handle line coverage
  - Branch and path coverage are more elusive goals

# Agile documentation

- Documentation is not fun: "No man but a blockhead ever wrote except for money" -- Samuel Johnson
- Documentation is necessary; Open Source projects live or die by their docs
- Tests as documentation – storytests (Kerievsky)
- "Executable documentation", "executable requirements"
  - FitNesse, Robot framework
  - Python: doctests

# So...when are you “DONE”?

- In an agile world, you're done when the tests for all stories in the current iteration pass
- For the nagging feeling in the pit of the stomach to go away, you need ideally all types of tests:
  - Unit
  - Functional
  - Acceptance
  - Integration
  - Installability/multi-platform
  - Performance, stress, security, etc. etc. etc.

# Resources

- Agile testing
  - Elisabeth Hendrickson: <http://www.qualitytree.com/>
  - Brian Marick: <http://www.exampler.com/>
  - Lisa Crispin: <http://lisacrispin.com>
- Unit and mock testing
  - Michael Feathers book “Working Effectively with Legacy Code”:  
<http://www.amazon.com/Working-Effectively-Legacy-Robert-Martin/dp/013>
  - Roy Osherove book “The Art of Unit Testing”:  
<http://www.manning.com/osherove/>
  - Gerard Mesarosz: <http://xunitpatterns.com/>
  - Martin Fowler:  
<http://www.martinfowler.com/articles/mocksArentStubs.html>

# Resources (cont.)

- Acceptance testing tools
  - FitNesse: <http://fitnesse.org/>
  - Robot Framework: <http://code.google.com/p/robotframework/>
- Web application testing tools
  - Selenium: <http://seleniumhq.org/>
  - Twill: <http://twill.idyll.org/>
- Continuous integration tools
  - Hudson: <https://hudson.dev.java.net/>
  - Buildbot: <http://buildbot.net/trac>

# Resources (cont.)

- Agile documentation
  - Joshua Kerievsky's storytests:  
<http://www.industriallogic.com/papers/storytest.pdf>
  - My presentation at PyCon 2006:  
[http://www.agilistas.org/presentations/pycon06/pycon06\\_agiledoc.pdf](http://www.agilistas.org/presentations/pycon06/pycon06_agiledoc.pdf)
- Python testing tools taxonomy:  
<http://pycheesecake.org/wiki/PythonTestingToolsTaxonomy>
- Some of my blog posts and articles on testing and more:  
<http://agilistas.org/articles/>