Test Automation

Jon Schewe - jschewe@bbn.com

BBN Technologies

January 11, 2010

Outline

Why test automation?

- programmers are lazy
- Automated system finding bugs is better than people
- Need a reproducable test system

Test automation setup

- Make sure you reproduce the running system
- May need virtual machines
- Do whatever is necessary to get your test data

Ant

- Least common denominator
- Make this the gold standard
- Don't make developers always wait too long for tests, otherwise they'll skip
- use multiple targets
  - one for quick unit tests
  - one for longer integration tests
  - short test for current task

Continuous Integration (CI)

- Hudson or CruiseControl
- Once you can run automated tests - run them in CI!
- Will send emails on status of builds

Metrics & CI

- Don’t display too much
- Find out what numbers are important
- Play to the developers egos
- Don’t point at problems bugs, point at buggy code

Some Metrics

- McCabe Cylomatic
- http://en.wikipedia.org/wiki/Cyclomatic_complexity
- Code coverage

static code analysis

- run against development code
- run against test code too
- Start with FindBugs
- Move to PMD once FindBugs is clean enough
- Open Tasks
- Copy Paste Detector

Example Metric - Risk

- risk = (McCabe's * call count) * coverage percentage
- Example risk = (70 * 74) * 50
- List top 10 classes by risk
- Watch and see the results

Writing Tests

- Don’t open the Kimono
- Only test the public API

Characteristics of a good test

Right-BICEP

- Are the Results Right
- Boundary Conditions
- Check Inverse Relationships
- Cross-check using Other Means (Test the Oracle)
- Force Error Conditions (Attacks)
- Performance Characteristics

Characteristics of a good test (cont.)

- abstract away the test tools
- keep test methods short
- longer than a page and it needs to be shortened

Mocks & stubs

- Sometimes the terms are interchangble
- stubs are usually written by people
- mocks are automatically created, just count method calls
- If you use Continuous Integration you shouldn't need mocks

Integration vs. Unit tests

- Unit Tests
  - good for starting from new code
- Integration Tests
  - good for legacy code
  - best use of time (for legacy code)
  - will end up with low code coverage percentage
  - but it's the right percentage

Test Driven Design (TDD)

- Write one test then write code to make the test pass
- Causes you to really think
- Use for new code
- will end up with more stable code
- will end up with high code coverage (not goal though)
- Pair programming

Defect Driven Testing (DDT)

- Find a bug
- Add a test
- Jazz it up
  - add tests with variations
  - never write 1 test for a bug

Testable code

- Good testable code does 1 thing and then returns

Testing Multi-tier architecture

- Mock everything up to start (tracer bullets)
- Test everything with canned data
- Allows you to find out if the architecture works early

Picking Tools

- Have 1 person pick the tools and go with their choice
  - otherwise end up with too many options
  - Have them write the test templates
  - Make everyone use it

Database Tools

- Liquibase
- ruby database migrations

Web Testing

- Selenium
  - Firefox plugin for IDE to create tests
  - Can call multiple browsers
- YSlow
  - static web page analysis tool
  - Gives performance tips

UI testing

- Test at the controller and model layer
- Use something like selenium or AWT Robot

Resources

- Pragmatic Programmer
- Pragmatic Unit Testing
- Buildix - can download everything for CI in a vm