Winning the Battle against Automated Testing

Elena Laskavaia
March 2016
Foundation of Quality

Quality

People  Process  Tools
Development vs Testing

- Developers don’t test
- Testers don’t develop
- Testers don’t have to be skilled
- Separate Developers and Testers
- Make the Test team responsible for quality
One Team
Quality is a team responsibility
The Process

When quality is bad let's add more steps to the process.
Story about the broken Trunk

Thousands of developers
Continuous stability is a must
“Trunk is broken” too often
Huge show stopper for R&D
People did root-cause analysis
Came up with Improved Process
“Improved” Pre-Commit Process

1. Pull/Update All Source
2. Clean compile All
3. Re-build and re-deploy whole system
4. Manually execute sanity test cases
5. Repeat for all hardware variants
Trunk is still broken. Why?

- Process was not followed
- Process is too complex
- Process is too boring
- Process is too time consuming
- Environment / Hardware limitations
- Developers are lazy
- Developers don’t know about the process
- Developers don’t know about the process
Automated Pre-Commit Testing
Pre-Commit Tests with Source Management System

- Fix
- Push
- Checks
  - Peer reviews
  - Robots checks

master
Randomly pick a tool

Spend 6 month developing testing framework

Need a person to run it for every build

Oops our tester quit, who knows how to run it?

It does not work at all now!

Oh well we don’t have any more budget and time, let go back to manual testing
Continuous Testing
Continuous Quality
Cost of Automation

• Cost of Tools
• User Training
• Integration and Customization
• Writing Test Cases
• Executing Test Cases
• Maintaining Test Cases
Jump Start

1. Make one team responsible
2. Setup continuous integration
3. Add pre-commit hooks
4. Establish simple self-verifying process
5. Add one automated test
Key Principles of successful automated testing
Gate Keeper

test system must guard the gate
100% Success

100% of tests must pass. Zero tolerance
NO random failures

- Remove such tests from automation
- Use repeaters to keep intermittent tests
- Be prepared for the noise
- Modify AUT to remove source of randomness for tests
• No monkeys pushing buttons to start the testing
• No monkeys watching automated UI testing
• Hooks on code-submission (pre-commit, fast)
• Hooks on build promotion (overnight)
Fast and Furious

- Feedback for pre-commit <=10 min
- Overnight is absolute maximum
- More tests degrade the system response time
- **Not all tests are born equal!**
- Use tagging and filtering
- Distribute or run in parallel
- No sleeps
Timeouts

• Make sure tests are not hanging!
• Use timeouts
• Use external monitors to kill hanging runs
• Do not overestimate timeouts
Test Scripts are Programs

- Automated test cases are programs
- Treat them as source code
- They must be in text form
- They must go to same version control system
- Subject to code inspection, coding standards, build checks, etc
Unit Tests

• Mandatory with commit
• Use servers to run

Part of the process

• Use a mocking framework
• Use UI bot
• Use test generators
• Inline data sources

Easy to write
- Unit tests cannot cover all
- Test actual installed AUT
- Run the program as user would
- Use same language for unit and integration testing
Pick and Choose

Candidates

- Difficult to set-up cases
- Rare but important scenarios
- Check lists
- Module is actively developed
- Long maintenance expected

you should not automate everything
Self-Verification: test the test system?

Automatically Check

✓ Code submission is properly formatted (has bug id, etc)
✓ Code submission has unit tests
✓ Total number of tests is increased
✓ Performance is not declined
✓ Code coverage is not declined
Failed Battles
Tools we used or evaluated and failed

- WinRunner: after 3 months of writing tests, realized it won't work on Linux
- WindowTester: was pretty good until it was bought and it stopped launching with new eclipse
- Jubula: 4 years ago, database, no text for tests, no integration
- Squish: slow, not debuggable, blocks on support. Python
- RCPTT: domain specific language
Working Solution
Continuous Integration Tools

- Unit testing
  - JUnit

- Source Control and Code Review
  - Git/Gerrit

- Static Analysis
  - FindBugs

- Build System
  - Maven/surefire-plugin (for unit tests)
  - Maven/failsafe-plugin (for integration tests)
  - findbugs-plugin for static analysis

- Continuous Integration Server
  - Gerrit Trigger plugin - pre-commit builds and voting
  - FindBugs plugin - reports and status
  - JUnit plugin - reports and status

- Continuous Integration Tools
  - Maven/Tycho
  - Jenkins
  - SWTBot

- GUI testing

- Code Coverage
  - Ecl Emma

- Unit mocking
  - Mockito

- Code Coverage
  - Ecl Emma

- Custom
  - Lots of custom libraries, frameworks and bots
Tips and Tricks
Checks that can be added to every test

- App crashed during a test
- Test timeout exceeded
- App generated unexpected log
- Temp files were not cleaned up
- Resource or memory leaks
- Runtime error detection
public class SomeTest {
    // tests that we don’t leave tmp file behind (this is custom rule not base junit)
    @Rule TmpDirectoryCheck tmpRule = new TmpDirectoryCheck();

    @Test
    void testSomething(){
    }
}

// base class with timeouts
public abstract class TestBase {
    public @Rule Timeout globalTimeout = Timeout.seconds(1); // 1 second
To speed up verification for pre-commit hooks set up multiple jobs which trigger on the same event (i.e. patch submitted).
//    template<typename T2>
//    struct B : public ns::A<T2> {};
//    void test() {
//      B<int>::a;
//    }
public void testInstanceInheritance_258745() {
    getBindingFromFirstIdentifier("a", ICPPField.class);
}
Code Coverage

- Run tests with code coverage
- Not during pre-commit check
- Unless it has validation hooks
- Good tool for unit test design (IDE)
- Never ask for 100% coverage

WANTED

Code Coverage ->
Select Tests

Based on changed code
exclude tests that do not cover
the changes
Static Analysis

- Can be run independently
- Has to be a gatekeeper
- Spent time to tune it (remove all noisy checkers)
- Push to desktop (if running as you type - instantaneous feedback!)
- Use alternative UX on desktop (i.e. code formatter)

WANTED

Jenkins Plugin: Code Reviewer

Post defects from static analysis as reviewer comments on the patch
// tag class with categories in test class
@Category({PrecommitRegression.class, FastTests.class})
public class SomeClassTest {
    @Test
    public void someTest() {
    }
}

// in maven pom.xml
<build>
    <plugins>
        <plugin>
            <artifactId>maven-surefire-plugin</artifactId>
            <configuration>
                <groups>com.example.PrecommitRegression</groups>
            </configuration>
        </plugin>
    </plugins>
</build>
// skip test entirely if not running in osgi

@Before
public void setUp() {
    Assume.assumeTrue( Activator.isOsgiRunning() );
}
Intermittent Test: Junit Rule

You can create a runner or define a rule which repeats a test if it fails. Junit itself does not define either, you have to add it yourself (2 classes, 62 lines).

```java
public class SomeClassTest {
    public @Rule InterimittentRule irule = new InterimittentRule();

    // repeat this up to 3 times if it failing
    @Intermittent(repetition = 3)
    @Test
    public void someTest() {} 
}
```