Software Testing
in the
Modern Era

Enterprise Computing Conference
2016

Michael Gildein
Scott Sigalas
Agenda

• Tester mindset
• What is testing?
• Why test?
• Testing phases
  • Agile testing
• Test Techniques
How do we test a pencil sharpener?
Pencil Sharpener Test

- Does it sharpen a pencil?
- How sharp does it make a pencil?
- What if we have different lead densities?
- Can it handle different shaped pencils?
- How many pencils can I sharpen in a row before it breaks?
- What happens if batteries are inserted or removed with a pencil in?
- What happens if a pen is put in the sharpener?
- What happens if someone puts a finger in the sharpener?
- How many pencils can it sharpen before the catch is full?
- If the catch is full, does it still sharpen?
- What is the range of operating temperatures?
- Does the sharpener operate according to specs?
- ...
What is test?
Test Definition

“A procedure intended to establish the quality, performance, or reliability of something, especially before it is taken into widespread use.”

Purpose

1. Remove defects
2. Validate conformity to requirements (not specification!)
Why test?
Roles

Developer
• Maker
• Creative
• Optimistic
• +Testability
• +Static code analysis
• +Defensive programming

Tester
• Breaker
• Devious
• What if?
• Out of the box thinking
• 360 degree view
Defect Impact

Financial cost
- Code, test, and deploy fix
- Service Level Agreements (SLA)

Reputation/Trust
- Loss of customers
- Instant media coverage
- Social media viral explosion
- Reviews drive customer decisions

Legal issues
- Diminished reputation
- Costs incurred
- Liability

Applied Software Measurement, Capers Jones, 1996
Quality Software

Static
• Documentation
• Code structure
• Code maintenance (comments & readability)
• Testability

Dynamic
• Correctness
• Reliability
  • Probability of failure free operation
• Completeness
  • Requirements implemented
• Consistency
• Usability
• Performance
• Security
Classes of Software

- **Prototype or Beta**
  - No warranty; AS-IS
  - Defects expected
  - Developer preview edition

- **Consumer**
  - 99% uptime
  - Defect affects single user; retry/restart
  - Mobile app (1 sec attention)
Classes of Software

- **Enterprise**
  - 99.99%+ uptime; SLAs
  - Defects impact large number of users; lost revenue/productivity
  - Warehouse merchandise ordering; shipment tracking

- **Critical (Mission or Business)**
  - Near 100% uptime; SLAs
  - Substantial impact by defects
  - Transaction authorization services; Web services

- **Life Critical**
  - 100% reliability
  - Failure is unacceptable
  - Autonomous vehicles; pacemakers
Testing Escapes

- **Hubble Telescope**
  - Mirror grinding error due to null corrector test device miss-assembly by 1.3mm
  - Corrective optical components with inverse of error
  - $10 billion total cost in 2010
- **Intel Pentium Chip**
  - Floating point unit error causing potential incorrect decimal values while dividing
  - Circuit design error
  - Recalled chips $475 million
  - Public relation problems
- **Intel Skylake**
  - Complex workload causes hang or unpredictable system behavior
  - BIOS fix
- **Health Insurance Marketplace**
  - Public relation problems
  - High traffic as root cause
  - Design issues
- **Target Corp.**
  - Security breach
  - 70 million people affected
  - $10 million + credit monitoring
Testing Phases
Standard Test Phases

1. Unit Test
2. Integration Test
3. System Test
4. Acceptance/Beta Test
5. Maintenance

Launch
Release
General Availability (GA)
Enterprise Test Phases

1. Unit Test
2. Function Test
3. System Test
4. Integration Test
5. Beta Test
6. Service Test

- Algorithm Verification Test
- Performance Test
- Benchmark Test
- Internationalization Test

Launch Release General Availability (GA)
Test Phase Focus

- Code
  - Unit
- Design
  - Integration
- Requirements
  - System
Agile Methodologies

- 4-6 week iterations
- Continuous delivery
- Balance quality and time to market
- Applies to all software

Test Implications

- Ongoing communication
- Build automation
- Regression automation
- Test automation frameworks
- Prioritization is key
- Code fencing
Testing Techniques
Combinatorial Test Design
How many exhaustive combinations (test variants)?

- **Android level**
  - 4.3, 4.4, 5.0, 5.1, 6.0
- **Physical device**
  - Nexus 5, Nexus 6, Nexus 6P, Nexus 5x, LG G4, LG V10......
- **Down level devices**

http://developer.android.com/about/dashboards/index.html

<table>
<thead>
<tr>
<th>Version</th>
<th>Codename</th>
<th>API</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2</td>
<td>Froyo</td>
<td>8</td>
<td>0.1%</td>
</tr>
<tr>
<td>2.3.3 - 2.3.7</td>
<td>Gingerbread</td>
<td>10</td>
<td>2.2%</td>
</tr>
<tr>
<td>4.0.3 - 4.0.4</td>
<td>Ice Cream Sandwich</td>
<td>15</td>
<td>2.0%</td>
</tr>
<tr>
<td>4.1.x</td>
<td>Jelly Bean</td>
<td>16</td>
<td>7.2%</td>
</tr>
<tr>
<td>4.2.x</td>
<td></td>
<td>17</td>
<td>10.0%</td>
</tr>
<tr>
<td>4.3</td>
<td></td>
<td>18</td>
<td>2.9%</td>
</tr>
<tr>
<td>4.4</td>
<td>KitKat</td>
<td>19</td>
<td>32.5%</td>
</tr>
<tr>
<td>5.0</td>
<td>Lollipop</td>
<td>21</td>
<td>16.2%</td>
</tr>
<tr>
<td>5.1</td>
<td></td>
<td>22</td>
<td>19.4%</td>
</tr>
<tr>
<td>6.0</td>
<td>Marshmallow</td>
<td>23</td>
<td>7.5%</td>
</tr>
</tbody>
</table>

*Data collected during a 7-day period ending on May 2, 2016. Any versions with less than 0.1% distribution are not shown.*
What is Combinatorial Test Design

• Interactions introduce chance for error
• Non-interaction problems should be resolved in Unit Test
• CTD is an approach which ensures that each possible value for each parameter interacts at least once with every other value in your test plan
  • This is known as pairwise testing
  • Expandable to any $n$ less than the size of your test space (becomes exhaustive)
• Input or config parameters

CTD Example

• Consider a function with three parameters, each of which can take three values
  • Color: red, white, blue
  • Size: small, medium, large
  • Quantity: 1, 2, 3
• Mechanically taking each combination is 27 cases (3x3x3)
• Instead, construct sets that maximize efficiency by duplicating tests as little as possible
  • (red,small,1), (red,medium,2), (red,large,3),
  • (white,small,2), (white,medium,3), (white,large,1)
  • (blue,small,3), (blue,medium,1), (blue,large,2)
CTD Resources

• Microsoft PICT - https://github.com/Microsoft/pict
• ParTeG - http://parteg.sourceforge.net/
• http://www.pairwise.org/tools.asp
Test Driven Development
Test Driven Development (TDD)

- Requirements specified as tests
- Ensure test fails first
- Small dev units
- Agile

Pros
- Increased quality? No proven
- Increased testability
- Everything gets tested (100% coverage)
- Reduced debugging effort
- Revert vs debug
- Less “gold plating”

Cons
- Only applies to unit test phase
Test Driven Development (TDD)

1. Create test
2. Run all tests
3. Write code (only enough to resolve test failure)
4. Run all tests
5. Refactor code

Tests Pass && All Requirements Implemented
Mutation Testing
Test Adequacy

“Goodness” or “thoroughness” of a test compared against a quantitative criterion such as coverage.

Coverage metrics
- Statement
- Branch
- Path

Why?
- How do we know to stop testing? (We are never really done)
- 100,000 tests pass; always taking the happy path; fails on least taken path
- Never tested recovery requirement
Mutation

Process
• Change application under test (AUT) code in a small way == mutant
• Result
  • Test fails => Success (Killed mutant)
  • Test Passes => Failure (Mutant survived)
    • Tests do not adequately test the code!

Mutation Types
• Value
  • Change 0 constant to 10000000
  • (X < Y) becomes (Y < X)
• Decision
  • Change conditional from AND to OR
  • Change arithmetic from + to *
• Statement
  • Delete or duplicate statement as if a copy & paste error
  • Remove else case on if statement
if (foo == 0) {
    bar();
}

if (foo != 0) {
    bar();
}

• Asserting that bar() was only called when foo was 0 then would fail.

int loopCounter = 10;

int loopCounter = 10000;

• Asserting an execution time limit or verifying loop actions then would fail.
Mutation Tools

- On-the-fly mutation automation software required
- Dynamically create changes to code (or test cases)
- Report on test case success (killed mutants)

Tools

- PIT - http://pitest.org
- muJava - https://cs.gmu.edu/~offutt/muja
- Major Mutants - http://mutation-testing.org/
- Mutate - https://github.com/arun-babu/mutate.py
Test Analytics
Test Analytics

- Root Cause Analysis (RCA)
- Defect Sensitivity
- Defects/KLoc (New & Changed)
- Defect Concentrations
- Orthogonal Defect Classification (ODC)
- Test Case/Workload Efficiency
- Testing Based on Build or Changes
Summary

• Testing is non-trivial
• Testing is its own field
• Careers in software test are becoming common place
• Testing theory and practices need to be taught to all in tech
  • Marist MSCS721 – Software Verification and Maintenance
  • IBM Academic Initiative – Enterprise Software Testing
Questions?