The Ugly Reality of Software Development

Complex Software

New Feature

More Bugs

Fix Bugs

More Complex

New Feature

More Bugs
Most Software Projects Fail

Fail – cancelled, run late, run over the budget, or deliver less than promised

*Fail – cancelled, run late, run over the budget, or deliver less than promised

Project Success
Smaller initiatives fare better at reaching goals than larger projects do.

- More than $10 million: 2%
- $6 million to $10 million: 11%
- $3 million to $6 million: 23%
- $750,000 to $3 million: 32%
- Less than $750,000: 46%

Source: The Standish Group
Software Projects Fail for Many Reasons

- Badly defined requirements
- Poor management
- Incompetent developers
- ...
Code Quality: A Common Reason For Failure

Continuous Attention to Quality is Important

Source: Applied Software Measurement, Capers Jones, 1996
Developer Testing: Keeping Code at Check

**Developer testing mantra:** test your code before integrating with the rest of the system

- Very effective way to improve software quality
  - Cost of fixing unit-level defects early is smaller
  - Enables the QA team to focus on larger defects
- Typically performed by writing small unit tests
- Does **not** replace system or integration testing
Developer Testing Benefits

- Rapid feedback (on code and design)
- Reduces unit-level bugs in new and existing features
- Forces you to slow down and think (which makes progress faster overall!)
- Tests are good documentation
- Defend against other programmers
- Tests reduce fear of code changes
Why Developer Testing Works

Why Developer Testing Works

With Developer Testing

Continuous Feedback

Without Unit Testing

Development

Code Construction

Quality Assurance

Debug & Rework

Integration & System Testing

Code Quality

Excellent

Good

OK

Poor

Very Poor

Fail / Cancel

Done

Done Sooner!

No Feedback

Time

Limited Adoption of Developer Testing

- Cultural problem: “testing is for QAs”
  - Agile software development methods are popularizing developer testing
  - But more work is needed to get developers “test-infected”

- Tools problem: “it is too time-consuming”
  - Automating tasks that require no human input
  - Facilitating mental transition to testing mode
  - Maintaining a reusable set of regression tests
Unit Testing: What is a “Unit”? 

- A single method/function/procedure
- A collection of related methods/functions/procedures (e.g., one or more Java classes)

- So, how do you test such a unit in isolation from the rest of the system?
Demo 1: Testing a Linked List
What’s Wrong with This Approach to Testing?

✖ Need to run tests manually
✖ Need to verify results manually
✖ Need to create test data manually
✖ Need to know how much to test

♦ …but

✔ You are gaining confidence that your code works as designed
✔ You are receiving early feedback on the API design by using it yourself
Why We Need Unit Testing Tools

- Automation enables continuous monitoring of code quality
  - Know when things work (and break)
- Can automate test execution through scripts and output diff-ing
- Modern unit-testing frameworks simplify automation
  - Java: JUnit, TestNG
  - C#: NUnit, C++: CPPUnit, etc.
Demo 2: Testing with JUnit
Is JUnit a Perfect Solution?

✅ Can run tests automatically
✅ Can verify results automatically
❌ Need to create test data manually
❌ Need to write assertions manually
❌ Need to know how much to test
   ➔ Can use coverage information as a guide
   ➔ Coverage not built into JUnit, but other tools are available, e.g. EclEmma:
Partial Correctness Assertions (PCA)
- Hoare triple: \{P\} S \{Q\}
  - If P is true at the time S executes, then Q must be true after S completes

JUnit tests

```
list = new IntegerLinkedList(); {P}
list.addFirst(42); S
assertTrue(list.getFirst() == 42); {Q}
```
Testing vs. Formal Verification

- Hoare triples were developed for formal verification (proving programs correct)
  - Guarantees that program works in all cases
  - Infeasible for large programs
- Testing only guarantees that your program works in a few specific cases

\[ \text{Testing can be used to show the presence of bugs, but never to show their absence!} \]
\[ \text{(E. W. Dijkstra)} \]

- Good tests require variation of Ps (test data) and Qs (test assertions)
Minding your Ps and Qs: Test Data

- Challenge: cover as many various code paths as possible
  - …including all corner cases
  - …including all error cases
- Example: testing java.util.HashMap
  - Did you test put(k,v) into an empty map?
  - …one element map?
  - …when two keys hash to the same bucket?
  - …when a put() causes internal resize?
- Testing is a combinatorial problem!
Minding your Ps and Qs: Assertions

- Challenge: assertions must reflect how the code should behave
  - …does the specification exist?
  - …are the assertions strong enough?
    - `assertTrue(size > 0)` vs. `assertTrue(size == 5)`

- Developers find it difficult to switch mental context from coding to testing
  - Coding: mostly constructive
  - Testing: mostly destructive
How Can We Build Tools That Help?

- **Test data:**
  - Use static and dynamic source code analysis to create test data to cover as many paths in the code as possible

- **Assertions:**
  - Help the user by computing some possible invariants, but rely on their expertise to decide which ones correspond to the spec
Agitation helps interactively validate intended code behavior

- Discover invariants (or specify and validate them)
- Achieve high data and state coverage without manual setup
- Test without needing application server, database, etc.
Demo 3: Software Agitation
Software Agitation: From Daikon...

- **Daikon**: dynamic detection of likely invariants (Ernst et al., In ICSE’99)
  - Uses existing tests to exercise code
  - Applies dynamic analysis to infer invariant properties
  - Example: \( j \leq i < k, \ a = b + 42 \)
…to Agitator

- **Agitator**: developers check the detected invariants to see if they represent the desired behavior
  - Yes: developers convert observed behavior to assertions (create tests)
  - No: developers fix the program and repeat
  - Generates test inputs (mostly) automatically
  - Detects “invariants” that do not always hold

- Agitator combines Daikon with automatic test-input generation in a tool for developer testing (Boshernitsan et al., In ISSTA’06)
Agitation goal: execution path coverage

Static analysis:
- Path-constraint solving for param. values
- Type-specialized solvers provide values for common non-primitives (String, Collections)

Dynamic analysis:
- Observes execution and tries to force alternative paths (like concolic testing)
- Uses heuristics to guess values and objects that get coverage
Agitation iterates: no need to achieve coverage on first attempt
- Also, invariant detection needs multiple passes to get various data points

Heuristics help performance:
- Only consider partial paths (up to size X)
- Try common values (-1, 0, 1, constants from class files, mutate at random, etc.)
Dynamic Invariant Detection

- Find observable values at a given point
  - Parameters
  - Fields accessed in method-under-test
  - Properties of simple data types (string len.)
  - Getters for objects in scope

- Assume all possible relationships
  - Boolean expressions in source code
  - Relational, ranges, linear, etc.

- Discard relationships as (sufficiently many) counterexamples are found
From Research to Practice: Usability

- Speak the developer’s language
  - ✗ Representing invariants in first order logic
  - ✗ Designing new specification language
  - ✓ Expressing invariants in Java-like syntax

- Make no assumptions about expertise
- Integrate into existing workflow
From Research to Practice: Applicability and Scalability

- Handle real-world code
  - Poorly-structured (2k-line methods!)
  - Highly interdependent
  - Java native methods inhibit analysis
- Support continuously evolving code
- Scale to modern-size applications
Beyond Agitation

- Software agitation enables *exploratory* testing of Java code
  - Discovers unexpected behavior
  - The human (developer) is the test oracle
  - Reduces the grunt work, but leaves important decisions to the user

- Next time:
  - Advanced unit-testing techniques
  - Automated test generation
Questions?

Marat Boshernitsan
marat@agitar.com