Testing

V22.0474-001 Software Engineering
Lecture 7, Spring 2008

Clark Barrett, New York University
(with slides from Alex Aiken, Tom Ball, George Necula)

Reality

- Many proposals for improving software quality
- But in practice this is mostly testing
  - > 50% of the cost of software development

Role of Testing

- Testing is basic to every engineering discipline
  - Design a drug
  - Manufacture an airplane
  - Etc.
- Why?
  - Because our ability to predict how our creations will behave is imperfect
  - We need to check our work, because we will make mistakes

Testing and Development of Software

- In what way is software different?
- Folklore:
  "Optimism is the occupational hazard of programming; testing is the treatment"
  - The implication is that programmers make poor testers

Why Test?

Mars Climate Orbiter

- Purpose: to relay signals from the Mars Polar Lander once it reached the surface of the planet
- Disaster: smashed into the planet instead of reaching a safe orbit
- Why: Software bug - failure to convert English measures to metric values
- $165M

Shooting Down of Airbus 320

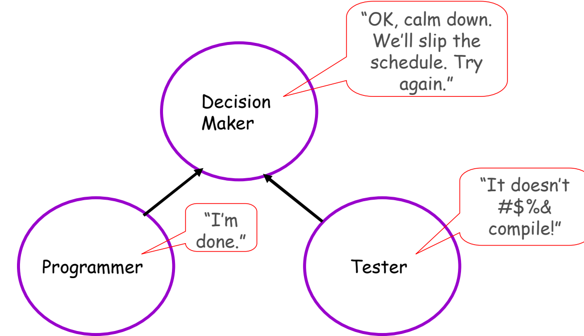
- 1988
- US Vicennes shot down Airbus 320
- Mistook airbus 320 for a F-14
- 290 people dead
- Why: Software bug - cryptic and misleading output displayed by the tracking software

THERAC-25 Radiation Therapy

- THERAC-25, a computer-controlled radiation-therapy machine
- 1986: two cancer patients at the East Texas Cancer Center in Tyler received fatal radiation overdoses
- Why: Software bug - mishandled race condition (i.e., miscoordination between concurrent tasks)

5

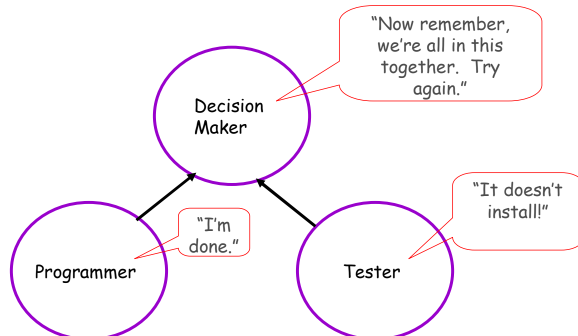
Typical Scenario (1)



Adapted from Prof. Necula, CS 169, Berkeley

6

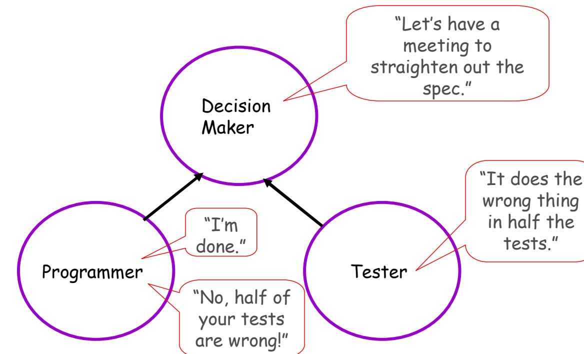
Typical Scenario (2)



Adapted from Prof. Necula, CS 169, Berkeley

7

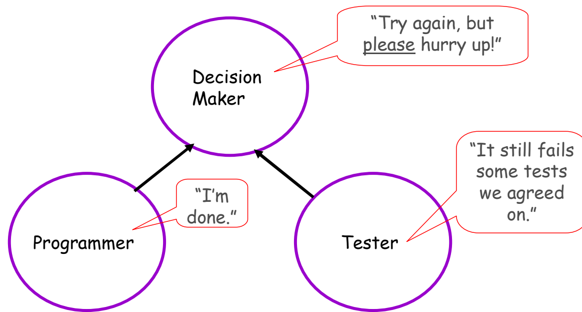
Typical Scenario (3)



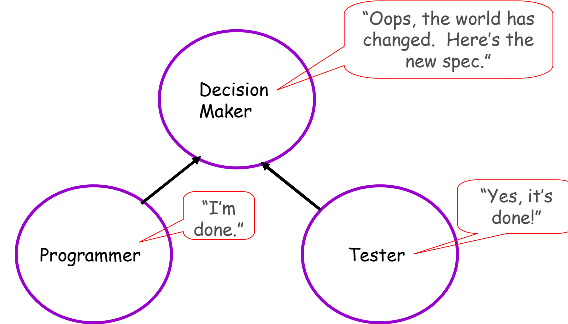
Adapted from Prof. Necula, CS 169, Berkeley

8

Typical Scenario (4)

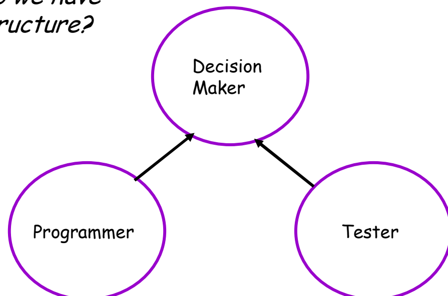


Typical Scenario (5)



Software Development Today

Why do we have this structure?



Key Assumptions

- Human organizations need decision makers
  - To manage (finite) resources (including time)
- Development and testing must be independent

Independent Testing

- Programmers have a hard time believing they made a mistake
  - Plus a vested interest in not finding mistakes
- Design and programming are constructive tasks
  - Testers must seek to break the software

Independent Testing

- Wrong conclusions:
  - The developer should not be testing at all
    - Recall "test before you code"
  - Testers only get involved once software is done
  - Toss the software over the wall for testing
    - Testers and developers collaborate in developing the test suite
  - Testing team is responsible for assuring quality
    - Quality is assured by a good software process

The Purpose of Testing

- Two purposes:
- Find bugs
  - Find important bugs
- Elucidate the specification
  - When testing the prototype or strawman

Example

- Test case

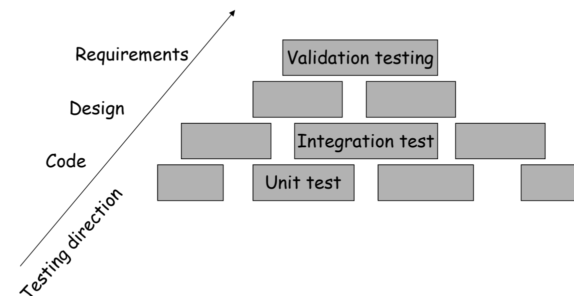
  Add a child to Mary Brown's record

- Version 1
  - Check that Ms. Brown's # of children is one more
- Version 2
  - Also check Mr. Brown's # of children
- Version 3
  - Check that no one else's child counts changed

Specifications

- Good testers clarify the specification
  - This is creative, hard work
- There is no hope tools will automate this
  - This part will stay hard work

Testing Strategies



Unit Tests

- Focus on smallest unit of design
  - A procedure, a class, a component
- Test the following
  - Local data structures
  - Basic algorithm
  - Boundary conditions
  - Error handling
- May need drivers and stubs
- Good idea to plan unit tests ahead

Integration Testing

- If all parts work, how come the whole doesn't?
- For software, the whole is more than the sum of the parts
  - Individual imprecision is magnified (e.g., races)
  - Unclear interface design
- Don't try the "big bang" integration !
- Do incremental integration
  - Top-down integration
  - Bottom-up integration

Top-Down Integration

- Test the main control module first
- Slowly replace stubs with real code
  - Can go depth-first
    - Along a favorite path, to create a working system quickly
  - Or, breadth first
- Problem: you may need complex stubs to test higher-levels

Bottom-Up Integration

- Integrate already tested modules
- No stubs, but need drivers
  - Often the drivers are easier to write
- Example:
  - Financial code that depends on subroutine for computing roots of polynomials
  - We cannot test the code without the subroutine
    - A simple stub might not be enough
  - We can develop and test the subroutine first
- Plan for testability !

Validation Testing

- Culmination of integration testing
  - The software works, but does it do what we need?
- Run acceptance tests
  - Get your customer to define them
- Alpha-testing (in controlled environment)
  - With developer looking over the shoulder
- Beta-testing
  - At end-user sites

Other Forms of High-Level Testing

- System testing
  - Involves non-software components
- Security testing
  - Red-team testing
- Performance testing
  - E.g., real-time systems
- Stress testing …

Stress Testing

- Push system into extreme situations
  - And see if it still works . . .
- Stress
  - Performance
    - Feed data at very high, very low rates
  - Interfaces
    - Replace APIs with badly behaved stubs
  - Internal structures
    - Works for any size array? Try sizes 0 and 1.
  - Resources
    - Set memory artificially low.
    - Same for # of file descriptors, network connections, etc.

Stress Testing (Cont.)

- Stress testing will find many obscure bugs
  - Explores the corner cases of the design

  "Bugs lurk in corners, and congregate at boundaries"
- Some may not be worth fixing
  - Too unlikely in practice
- A corner case now is tomorrow's common case
  - Data rates, data sizes always increasing
  - Your software will be stressed

Assertions

- Use assert(…) liberally
  - Documents important invariants
  - Makes your code self-checking
  - And does it on *every* execution!
- Opinion: Most programmers don't use assert enough

A Problem

- Testing is weak
  - Can never test more than a tiny fraction of possibilities
- Testers don't know as much about the code as the developers
  - But developers can only do so much testing
- What can we do?

Code Inspections

- Here's an idea: Understand the code!
  - One person explains to a group of programmers how a piece of code works
- Key points
  - Don't try to read too much code at one sitting
    - A few pages at most
  - Everyone comes prepared
    - Distribute code beforehand
  - No blame
    - Goal is to understand, clarify code, not roast programmers

Experience with Inspections

- Inspections work!
  - Finds 70%-90% of bugs in studies
  - Dramatically reduces cost of finding bugs
- Other advantages
  - Teaches everyone the code
  - Finds bugs earlier than testing
- Bottom line: More than pays for itself

Notes

- Some distinguish "walkthroughs" and "inspections"
- Walkthroughs are informal
- Inspections are formal
  - Extensive records kept
  - Metrics computed
  - Etc.

Manual Testing

- Test cases are lists of instructions
  - "test scripts"
- Someone manually executes the script
  - Do each action, step-by-step
    - Click on "login"
    - Enter username and password
    - Click "OK"
    - ...
  - And manually records results
- Low-tech, simple to implement

Manual Testing

- Manual testing is very widespread
  - Probably not dominant, but very, very common
- Why? Because
  - Some tests can't be automated
    - Usability testing
  - Some tests shouldn't be automated
    - Not worth the cost

Manual Testing

- Those are the best reasons
- There are also not-so-good reasons
  - Not-so-good because innovation could remove them
  - Testers aren't skilled enough to handle automation
  - Automation tools are too hard to use
  - The cost of automating a test is 10x doing a manual test

Automated Testing

- Idea:
  - Record manual test
  - Play back on demand
- This doesn't work as well as expected
  - E.g., Some tests can't/shouldn't be automated

Fragility

- Test recording is usually very fragile
  - Breaks if environment changes anything
  - E.g., location, background color of textbox
- More generally, automation tools cannot generalize
  - They literally record exactly what happened
  - If anything changes, the test breaks
- A hidden strength of manual testing
  - Because people are doing the tests, ability to adapt tests to slightly modified situations is built-in

Breaking Tests

- When code evolves, tests break
  - E.g., change the name of a dialog box
  - Any test that depends on the name of that box breaks
- Maintaining tests is a lot of work
  - Broken tests must be fixed; this is expensive
  - Cost is proportional to the number of tests
  - Implies that more tests is not necessarily better

Improved Automated Testing

- Recorded tests are too low level
  - E.g., every test contains the name of the dialog box
- Need to abstract tests
  - Replace dialog box string by variable name X
  - Variable name X is maintained in one place
    - So that when the dialog box name changes, only X needs to be updated and all the tests work again
- This is just structured programming
  - Just as hard as any other system design

Regression Testing

- Idea
  - When you find a bug,
  - Write a test that exhibits the bug,
  - And always run that test when the code changes,
  - So that the bug doesn't reappear
- Without regression testing, it is surprising how often old bugs reoccur

Regression Testing (Cont.)

- Regression testing ensures forward progress
  - We never go back to old bugs
- Regression testing can be manual or automatic
  - Ideally, run regressions after every change
  - To detect problems as quickly as possible
- But, regression testing is expensive
  - Limits how often it can be run in practice
  - Reducing cost is a long-standing research problem

Regression Testing (Cont.)

- Other tests (besides bug tests) can be checked for regression
  - Requirements/acceptance tests
  - Performance tests
- Ideally, entire suite of tests is rerun on a regular basis to assure old tests still work

Nightly Build

- Build and test the system regularly
  - Every night
- Why? Because it is easier to fix problems earlier
  - Easier to find the cause after one change than after 1,000
  - Avoids new code from building on the buggy code
- Test is usually subset of full regression test
  - "smoke test"
  - Just make sure there is nothing horribly wrong

Discussion

- Testers have two jobs
  - Clarify the specification
  - Find (important) bugs
- Only the latter is subject to automation
- Helps explain why there is so much manual testing
- Nevertheless, automate as much as you can

Back to Design

- Testing has a profound impact on design
  - Because some designs are easier to test
- Design software so it can be tested!
- Or at least avoid designing software that cannot be tested

Principles of Testability

- Avoid unpredictable results
  - No unnecessary non-deterministic behavior
- Design in self-checking
  - At appropriate places have system check its own work
    - Asserts
  - May require adding some redundancy to the code

Principles of Testability

- Avoid system state
  - System retains nothing across units of work
    - A transaction, a session, etc.
  - System returns to well-known state after each task is complete
    - Easiest system to test
- Minimize interactions between features
  - Number of interactions can easily grow huge
  - Rich breeding ground for bugs
- Have a test interface

Testing Frameworks

- Key components of a test system are
  - Building the system to test
    - May build many different versions to test
  - Running the tests
  - Deciding whether tests passed/failed
    - Sometimes a non-trivial task (e.g., compilers) !
  - Reporting results
- Testing frameworks provide these functions
  - E.g., Tinderbox, JUnit

Summary

- Testing requires a certain mindset
  - Want to break the code
- Good testing is hard work
  - Requires real insight into the nature of the system
  - Will help elucidate the spec

Project

- Design and Test Document Assignment due Feb. 21
- Design Presentations on Feb. 26th
- It's not too early to start writing code: e.g. prototype pieces that you think may be challenging.