Testing

15-413: Introduction to Software Engineering

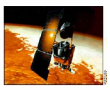
Jonathan Aldrich

Some slides from Tom Ball
and others in MSR's FSE group

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

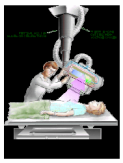


Shooting Down of Airbus 320

- 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)



16 November 2005

Testing: Current Challenges

- Test is huge cost of product development
- Test effectiveness and software quality hard to measure

- Incomplete, informal and changing specifications
- Downstream cost of bugs is enormous
- Lack of spec and implementation testing tools

- Integration testing across product groups
- Patching nightmare
- Versions exploding
- …

Testing Word

- Student-suggested issues
  - It's huge – can't test all combine all features
  - Simulate user interaction
  - Prepare audience different from testers
  - Platforms/hardware
  - Embedded external applications
  - What is the specified behavior / AI
  - Compatibility with old file formats

Testing Word

- inputs
  - keyboard
  - mouse/pen
  - .doc, .htm, .xml, …
- outputs (WYSIWYG)
  - printers
  - displays
  - .doc, .htm, .xml, …
- variables
  - fonts
  - templates
  - languages
  - dictionaries
  - styles
- Interoperability
  - Access
  - Excel
  - COM
  - VB
  - emacs
  - sharepoint
  - internet
- Other features
  - 34 toolbars
  - 100s of commands
  - ? dialogs
- Constraints
  - huge user base

Microsoft Powerpoint EULA Point 11

- **11**. EXCLUSION OF INCIDENTAL, CONSEQUENTIAL AND CERTAIN OTHER DAMAGES. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, **IN NO EVENT SHALL MICROSOFT OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER** (INCLUDING, BUT NOT LIMITED TO, DAMAGES FOR LOSS OF PROFITS OR CONFIDENTIAL OR OTHER INFORMATION, FOR BUSINESS INTERRUPTION, FOR PERSONAL INJURY, FOR LOSS OF PRIVACY, FOR FAILURE TO MEET ANY DUTY INCLUDING OF GOOD FAITH OR OF REASONABLE CARE, FOR NEGLIGENCE, AND FOR ANY OTHER PECUNIARY OR OTHER LOSS WHATSOEVER) **ARISING OUT OF OR IN ANY WAY RELATED TO THE USE OF OR INABILITY TO USE THE SOFTWARE PRODUCT,** THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES, OR OTHERWISE UNDER OR IN CONNECTION WITH ANY PROVISION OF THIS EULA, EVEN IN THE EVENT OF THE FAULT, TORT (INCLUDING NEGLIGENCE), STRICT LIABILITY, BREACH OF CONTRACT OR BREACH OF WARRANTY OF MICROSOFT OR ANY SUPPLIER, AND EVEN IF MICROSOFT OR ANY SUPPLIER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The GPL

- **11.** BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. **THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU**. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
- **12. IN NO EVENT** UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING **WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM** AS PERMITTED ABOVE, **BE LIABLE TO YOU FOR DAMAGES**, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES **ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM** (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

What are the goals of testing?

- Student answers
  - Make sure it doesn't crash
  - Regression testing – no new bugs
  - Make sure you meet the spec
  - Make sure you don't have harmful side effects

What are the goals of testing?

- Reveal faults
- Establish confidence
- Clarify the specification
- Represent the customer/verify contract

Limitations of Testing

- Testing can only show the presence of errors, not their absence
  - Dijkstra, 1972
- Why?

Black-box Testing

- Verify each piece of functionality of the system
  - Black-box: don't look at the code
- Systematic testing
  - Test each use case
  - Test combinations of functionality (bold + italic + font + size)
    - Generally have to sample
  - Test incorrect user input
  - Test each "equivalence class" (similar input/output)
  - Test uncommon cases
    - Generating all error messages
    - Using uncommon functionality
  - Test borderline cases
    - Edges of ranges, overflow inputs, array of size 0 or 1

Exercise: test binary search

- in/not in the array
- array with duplicate elements
- empty array, 1-element array
- even vs. odd array sizes
- unsorted/sorted array
  - Spec says array must be sorted
- < or > every element in array

White-box Testing

- Look at the code (white-box) and try to systematically cause it to fail
- Coverage criteria: a way to be systematic
  - Function coverage
    - Execute each function
  - Statement coverage
    - Most common
  - Edge coverage
    - Take both sides of each branch
  - Path coverage
    - Note: infinite number of paths!
    - Typical compromise: 0-1-many loop iterations
  - Condition coverage
    - Choose a set of predicates
    - Cover each statement with each combination of predicates
  - Exercise data structures
    - Each conceptual state or sequence of states
- Typically cannot reach 100% coverage
  - Especially true of paths, conditions

Unit Tests

- Usually automated
- Focus on one function at a time
  - May need to call other functions for setup
- Often specified by developer
  - Always in XP

Functional Tests

- Test entire end-to-end system functionality
- Often organized by use cases
- Often driven by separate testing team
  - Customer / customer representative in XP

Design for Testing

- Ensure components can be tested in isolation
  - Minimize dependences on other components
  - Provide constructors to set up objects for testing

Acceptance tests

- Functional tests that the customer uses to evaluate the quality of the system

Design by contract

- General meaning
  - Specify a contract between client and implementation of a module
    - Using pre- and post-conditions
    - System works if both parties fulfill their contract
- Specific setting of testing
  - Verify pre- and post-conditions while running
  - Assign blame based on which one fails
  - Turns a system execution into a set of unit tests

Regression Testing

- A suite of tests is run every time the system changes
- Goal: to catch any new bugs introduced by change
  - Need to add tests for new functionality
  - But still test the old functionality also!
  - Note: in some cases, old test cases *should* return a different result, depending on the change that was made

Nightly Builds

- Building a release of a large project every night
  - Catches integration problems where a change “breaks the build”
    - Breaking the build is a BIG deal—may result in midnight calls to the responsible engineer
- Typically, run regression test after building
  - Plot progress on tests over time

When are you done testing?

- Most common
  - Run out of time or money
- Can try to use statistical models
  - Only as good as your characterization of the input
  - Which is often quite bad
    - Exception: stable systems for which you have empirical data (telephones)
    - Exception: good mathematical model (avionics)
- Can seed faults
  - Halt when an “adequate” percentage is found
  - Implication: same percentage of unknown errors found
    - But is this really true?
- Rule of thumb: when error detection rate drops

Testing Quality Attributes

- Throughput
  - Increase load steadily through a series of tests until performance is unacceptable
    - Load profile should match actual operation profile of system
    - “Stress testing” tests the system beyond intended design limits
      - Look at failure behavior
      - Identify defects related to heavy load
- Reliability
  - Run for a period of time against operational profile, estimate reliability metric
  - Challenges:
    - Hard to know correct profile
    - Expensive to generate profile
    - Need large test cases to generate statistical confidence
      - Which is irrelevant anyway if the profile is off
  - Basically no good way to do this

Testing Quality Attributes

- Fault tolerance
  - Programmatically cause a fault and test that the system can recover
- Security
  - Attack team
- Usability
  - Measure user performance on some task
- Portability
  - Test against multiple platforms
- Evolvability
  - Design extension

Defect Tracking

- Organized handling of defects
  - Defect description
  - Problem analysis
  - Product and version affected
  - Originator, Owner
  - Status: open, confirmed, closed
  - Severity
  - Date reported, fixed
- Widely used in open source, industry
  - Tools like Bugzilla

Test Plan

- Strategy
  - Unit? Functional? White/Black box? Design by contract?
  - During requirements? Before coding? During test phase?
  - Quality attribute testing?
  - Nightly builds?
  - Completeness criterion?
- Document acceptance tests
  - Trace each requirement to one or more acceptance tests
- Tools
  - Generation? Regression? Selection? Coverage? Defect tracking?
- People
  - Developer or dedicated testers?