Announcements

- ComFoRT tutorial
  - Natasha Sharygina and Sagar Chaki
  - 12:15 Friday, location TBA
- Coming today on course web
  - Project requirements
  - Reading assignment

Software Testing

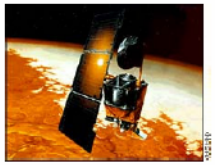
17-654/17-765
Analysis of Software Artifacts
Jonathan Aldrich

These slides prepared by Thomas Ball, with additional material from M. Young, A. Memon and MSR's FSE group. Used by permission.

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

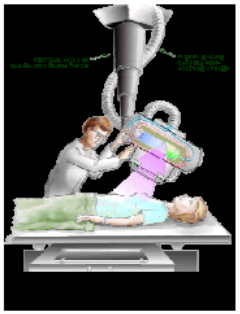


Shooting Down of Airbus 320

- 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)



Testing and The Software Process

- Three steps
  - X = test before coding
  - Y = test during coding
  - Z = test after coding
- Questions:
  - Who are your customers?
  - How to choose X, Y and Z to keep
    - your customers happy
    - yourself healthy

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

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.

::: NaDa ::: - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Search Favorites Media

Address http://www.bernardbelanger.com/computing/NaDa/ Go Links

Welcome to NaDa

NaDa does nothing for everybody




Enjoy the new NaDa™ 0.5 !

NaDa™ is a new concept. A thought, really. It is very light : 1 byte. It doesn't take long to fetch. It doesn't take long to understand. It doesn't disturb your habits nor does it makes you feel insecure. It is a reassuring piece of software that does nothing, and does it very well. That's a lot !

Most products we see on the market want to increase our productivity, organize our screen joyfully or make wonders with our sound card, but NaDa™ does nothing. This is a revolutionary

Free

Free NaDa™ 0.5
Click here to download.
NEW: more download options

Compatible with all Mac OSs, including OS X Jaguar, all Windows™ versions, all flavors of UNIX/Linux, Amiga, BeOS, everything you can think of, because we strongly believe that NaDa™ does nothing for everybody.

Contact NaDa™

How to use

Done Internet

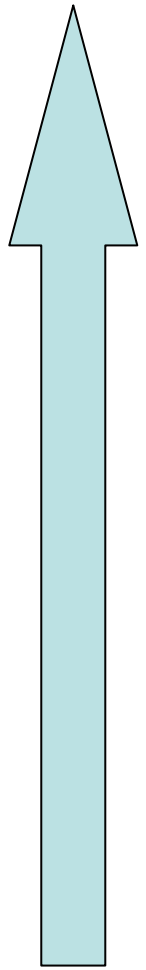
Goals for Understanding Testing

- What is testing and what are the key problems in testing?

- Model-centric testing
- Code-centric testing
- Test selection and prioritization

Standard Testing Questions

- Did this test execution succeed or fail?
- How shall we generate/select test cases?
- How do we know when we've tested enough?
- What do we know when we're done?



1. What do we know when we’re done?

Some Testing Goals

- Reveal faults
  - Glenford Myers, *The Art of Software Testing*
  - Dijsktra
- Establish confidence
  - of reliability
  - of (probable) correctness
  - of detection (therefore absence) of particular faults
- Clarify/infer the specification
- Represent the customer
- Minimize risk

Testing Theory (such as it is)

- Plenty of negative results
  - Nothing *guarantees* correctness
  - Undecidability of even simple properties
  - Combinatorial explosion
  - Statistical confidence is prohibitively expensive
  - Being systematic may not improve fault detection
    - as compared to simple random testing
  - …
- Few positive results
  - theory of finite state machines
  - specification-based testing

What Information Can We Exploit?

- Specifications (formal or informal)
  - in oracles
  - for selection, generation, adequacy
- Designs
  - ...
- Code
  - for selection, generation, adequacy
- Usage
  - historical or models
- Organization experience

Testing for Reliability

- Reliability is statistical, and requires a statistically valid sampling scheme
- Programs are complex human artifacts with few useful statistical properties
- In some cases the environment has useful statistical properties
  - stable, pre-existing systems (telephones)
  - systems with thoroughly modeled environments (avionics)

Process-Based Reliability Testing

- Rather than relying only on properties of the program, we may use historical characteristics of the development process

- Reliability growth models (Musa, Littlewood, et al) project reliability based on experience with the current system and previous similar systems

2. When can we stop?

Historical focus of research in 70s, 80s; no longer active.

“Adequate” testing

- Ideally: adequate testing ensures some property (proof by cases)
  - Origins in [Goodenough & Gerhart], [Weyuker and Ostrand]
  - In reality: as impractical as program proofs

- Practical “adequacy” criteria are really “inadequacy” criteria
  - If no case from class XX has been chosen, surely more testing is needed ...

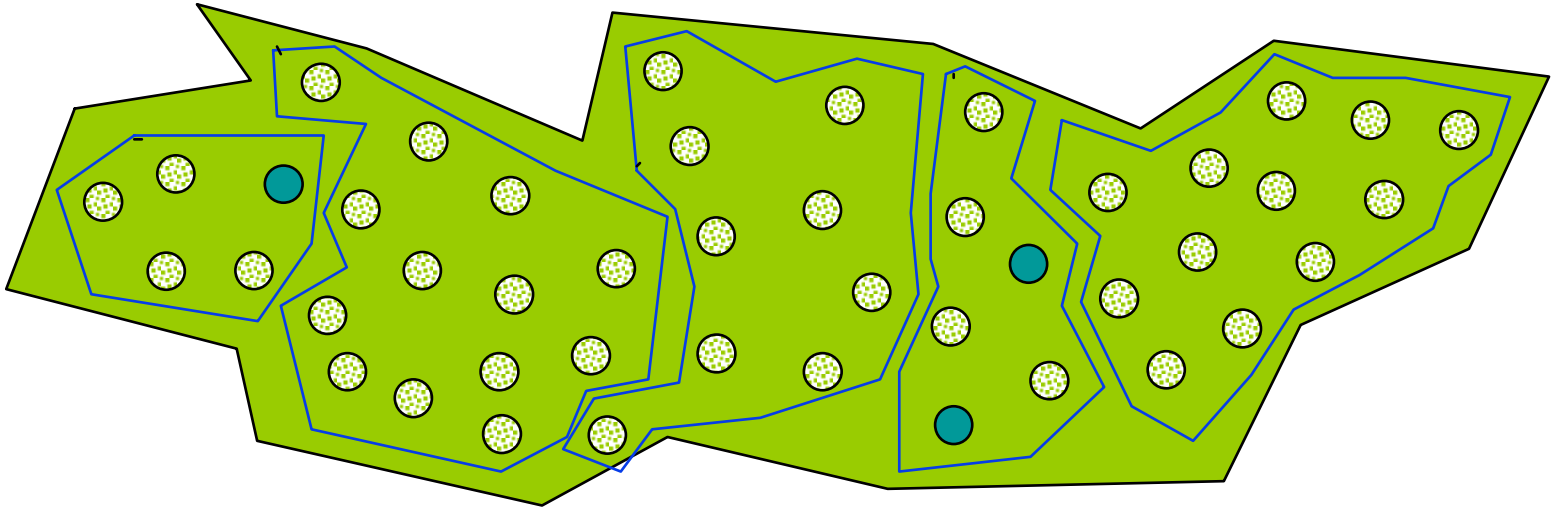
Systematic Testing

- Systematic (non-random) testing is aimed at program improvement, not measurement
  - obtaining valid samples and maximizing fault detection require different approaches
  - it is unlikely that one kind of testing will be satisfactory for both

- “Adequacy” criteria mostly negative: indications of important omissions
  - positive criteria (assurance) are no easier than program proofs

Partition Testing

- Basic idea: Divide program input space into (quasi-) equivalence classes
  - Underlying idea of specification-based, structural, and fault-based testing



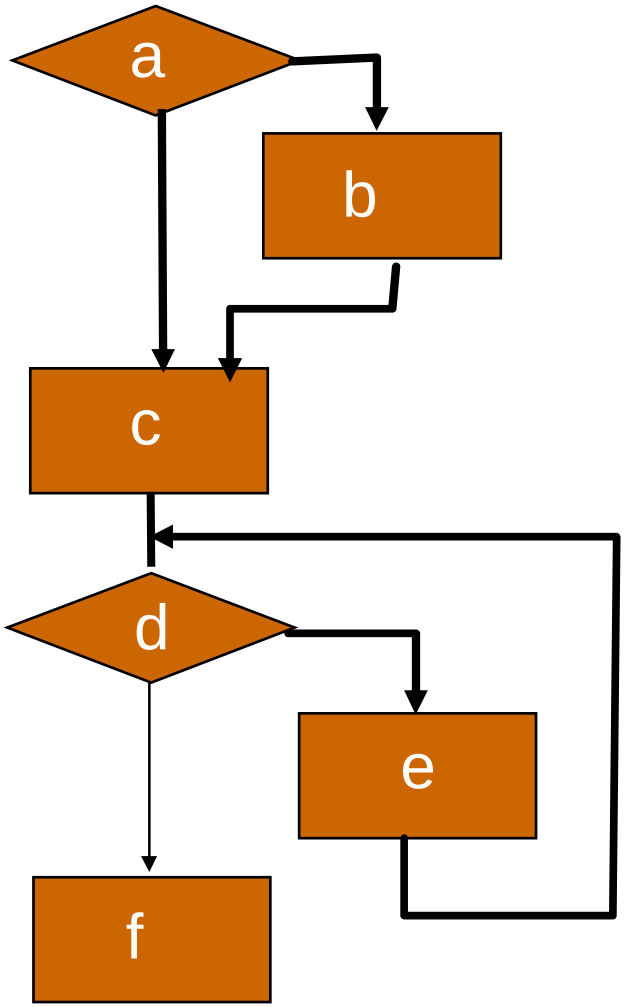
Specification-Based Partition Testing

- Divide the program input space according to identifiable cases in the specification
  - May emphasize boundary cases
  - May include combinations of features or values
    - If all combinations are considered, the space is usually too large
- Systematically “cover” the categories
  - May be driven by scripting tools or input generators
  - Example: Category-Partition testing [Ostrand]

Structural Coverage Testing

- (In)adequacy criteria
  - If significant parts of program structure are not tested, testing is surely inadequate
- Control flow coverage criteria
  - Statement (node, basic block) coverage
  - Branch (edge) and condition coverage
  - Data flow (syntactic dependency) coverage
  - Various control-flow criteria
- Attempted compromise between the impossible and the inadequate

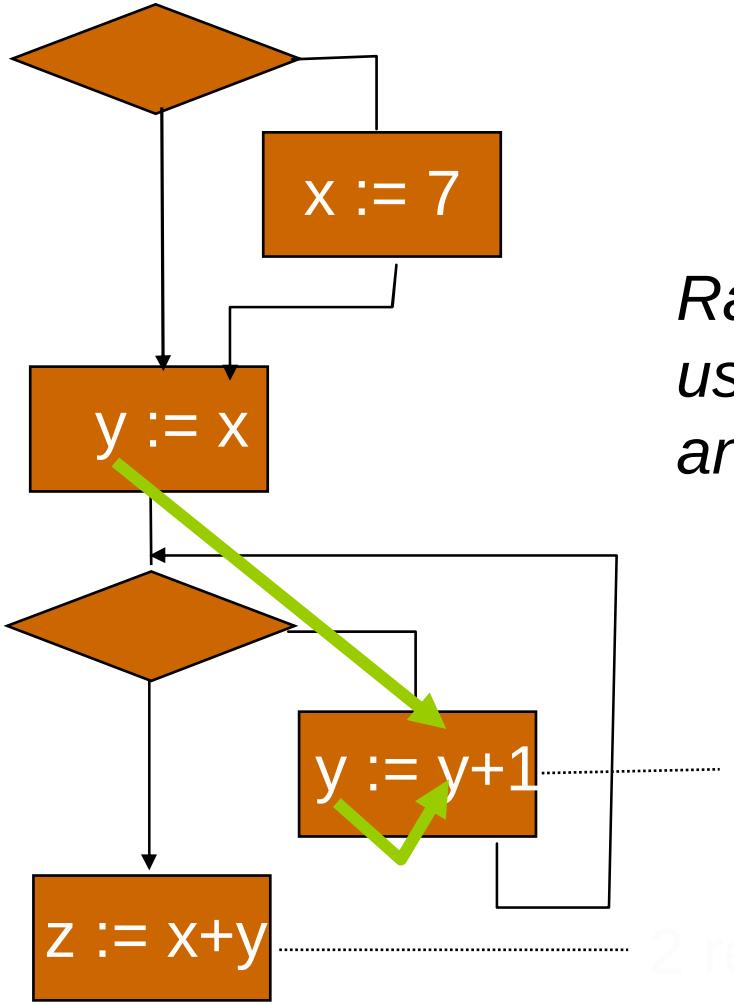
Basic structural criteria (ex.)



Edge ac is required by all-edges but not by all-nodes coverage

Typical loop coverage criterion would require zero iterations (cdf), one iteration (cdedf), and multiple iterations (cdededed...df)

Data flow coverage criteria (ex.)



Rationale: An untested def-use association could hide an erroneous computation

Structural Coverage in Practice

- Statement and sometimes edge or condition coverage is used in practice
  - Simple lower bounds on adequate testing; may even be harmful if inappropriately used for test selection
- Additional control flow heuristics sometimes used
  - Loops (never, once, many), combinations of conditions

Fault-based Testing

- Given a fault model
  - hypothesized set of deviations from correct program
  - typically, simple syntactic *mutations*; relies on coupling of simple faults with complex faults
- Coverage criterion: Test set should be adequate to reveal (all, or x%) faults generated by the model
  - similar to hardware test coverage

Fault Models

- Fault models are key to semiconductor testing
  - Test vectors graded by coverage of accepted model of faults (e.g., “stuck-at” faults)
- What are fault models for software?
  - What would a fault model look like?
  - How general would it be?
    - Across application domains?
    - Across organizations?
    - Across time?
- Defect tracking is a start

The Budget Coverage Criterion

- A common answer to “when is testing done”
  - When the money is used up
  - When the deadline is reached

- This is sometimes a rational approach!
  - Implication 1: Test selection is more important than stopping criteria per se.
  - Implication 2: Practical comparision of approaches must consider the cost of test case selection

3. How shall we generate/select tests?

Test Generation: Standard Advice

- Specification coverage good for generation as well as adequacy
  - applicable to informal as well as formal specs
- Fault-based tests
  - usually ad hoc, sometimes from check-lists
- Program coverage last
  - to suggest uncovered cases, not just to achieve a coverage criterion

Symbolic Execution

- Proposed for test generation in early 70s
- Given finite path to cover
  - generate constraints
  - check for satisfiability
  - if satisfiable then generate input
- Few tools in practice

Testing after change

- Change to spec/code may
  - make some tests obsolete
  - change test results
  - require generation of new tests

- Selective regression testing well studied
  - given a code change, what tests should we run?
  - Scout-tool widely used in MS

4. Was this test execution correct?

The Importance of Oracles

- Much testing research concentrates on adequacy, ignoring oracles
- Much testing practice relies on the “eyeball oracle”
  - Expensive, especially for regression testing
    - makes large numbers of tests infeasible
  - Not dependable
- Automated oracles are essential to cost-effective testing

Sources of Oracles

- Specifications
  - sufficiently formal (e.g., SCR tables)
  - but possibly incomplete (e.g., assertions in Anna, ADL, APP, Nana)
- Design, models
  - treated as specifications, as in protocol conformance testing
- Prior runs (capture/replay)
  - especially important for regression testing and GUIs; hard problem is parameterization

What can be automated?

- Oracles
  - assertions; replay; from some specifications
- Selection (Generation)
  - scripting; specification-driven; replay variations
  - selective regression test
- Coverage
  - statement, branch, dependence
- Management