Software Te$ting

18-849b Dependable Embedded Systems

Jiantao Pan

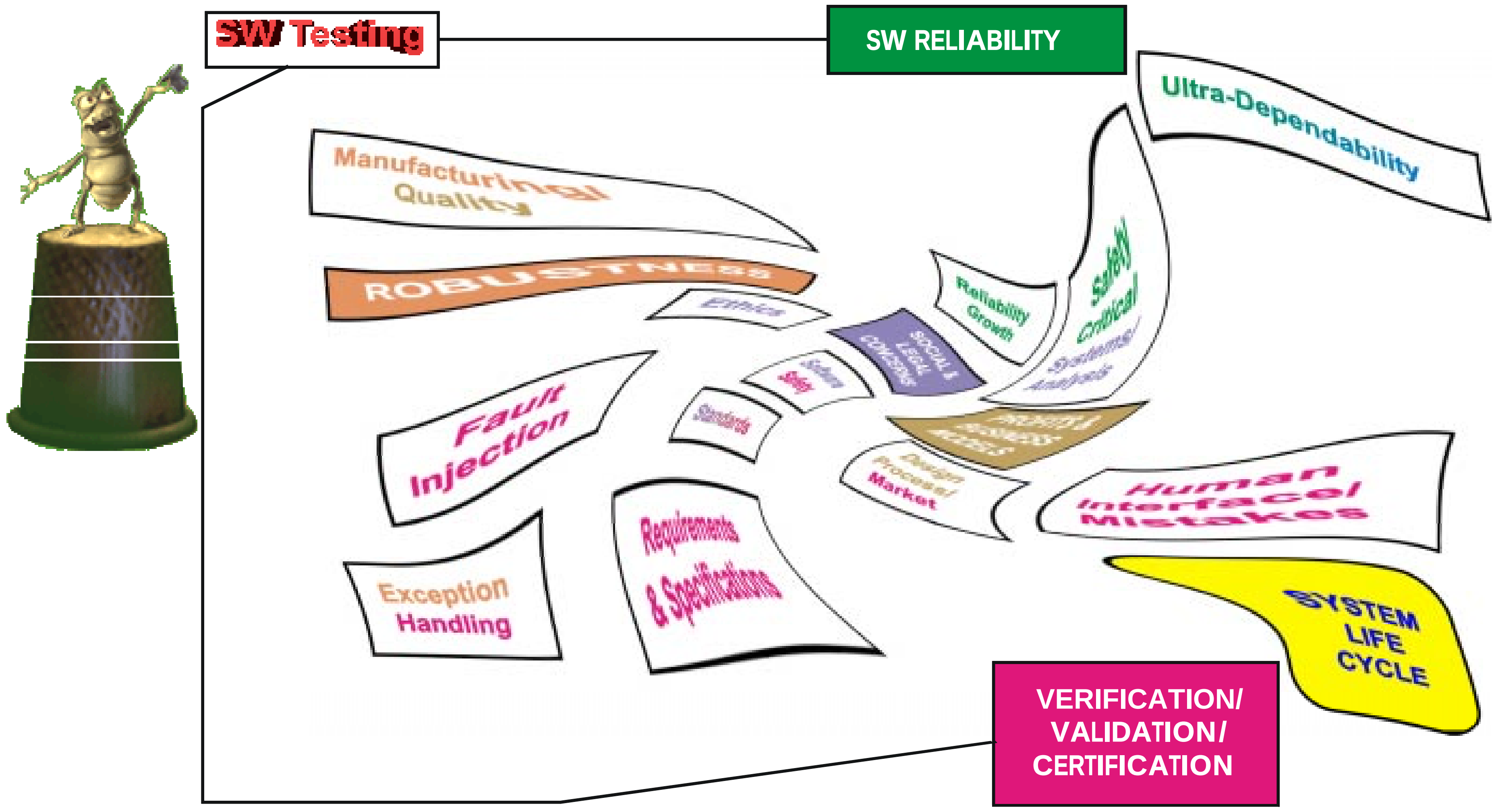
Feb 18, 1999

Required Reading:	Towards Target-Level Testing and Debugging Tools for Embedded Software
Fun Reading:	http://www.cnet.com/Content/Features/Dlife/Bugs/?dd
Best Tutorial:	Software-reliability-engineered testing practice; John D.Musa; Proceedings of the 1997 ICSE, Pages 628 - 629
Authoritative Books:	The Art of Software Testing, Glenford J. Myers, 1979 Black-box Testing, Boris Beizer, 1995 The Complete Guide to Software Testing, Hetzel, William C., 1988

Carnegie Mellon

You Are Here

- A lot of subtle relations to other topics

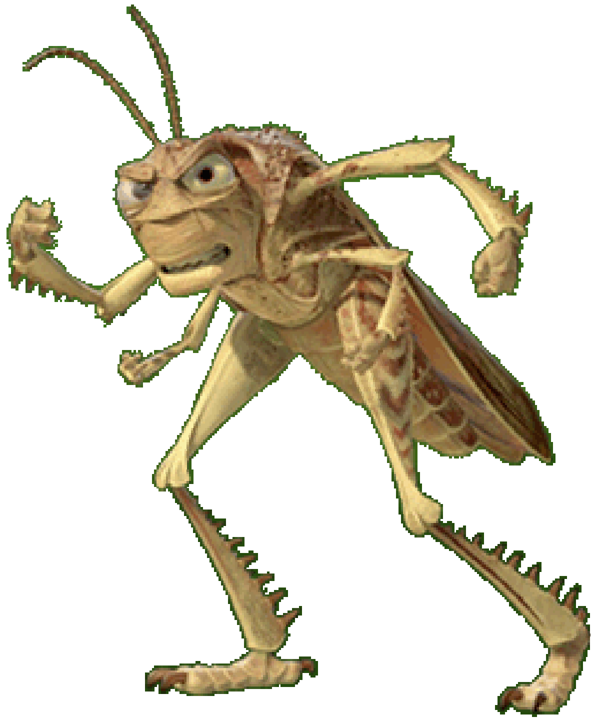


Introduction

- **Definitions of Software Testing**
  - [1]: Testing is the process of executing a program or system with the intent of finding errors.
  - [3]: Testing is any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results.
- **Vocabulary & Concepts**
  - Defects, bugs, faults[ANSI], errata[Intel]
  - Testing is more than debugging[BEIZER90]
- **Software testing is an ……** ART
  - because we still can not make it a science
- **Software testing is everywhere**
  - in every phase of software life cycle, whenever software changes
  - 50%+ time in debugging/testing
- **Software testing is not mature**

Why testing?

- **For Quality**
  - **bugs kill**
    - in a computerized embedded world
  - Defect detection (find problems and get them fixed [KANER93])
    - Better early than late
      - » Difficult to upgrade field software in embedded systems
  - To make quality visible [HETZEL88]
- **For Verification & Validation(V&V):**
  - show it works:
    - clean test/positive test
  - or it can handle exceptional situations:
    - dirty test/negative test
- **For Reliability Estimation [KANER93]**
  - E.g. reliability growth testing



Why software testing is difficult -- principles

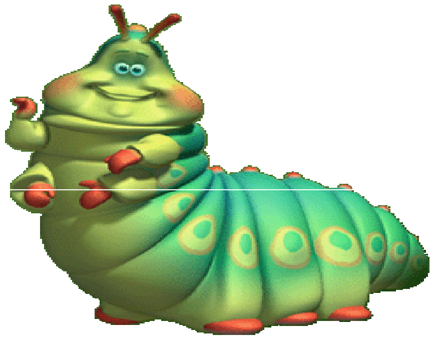
- **Software fails in different ways with physical systems**
- **Imperfection of human nature(to handle complexity)**
- **Cannot exterminate bugs**
  - We cannot test a typical program completely
  - The Pesticide Paradox[BEIZER90]
    - Every method you use to prevent or find bugs leaves a residue of subtler bugs against which those methods are ineffectual.
    - Fixing the previous(easy) bugs will tend to increase software complexity --> introducing new subtler bugs
  - The Complexity Barrier[BEIZER90]
    - Software complexity(and therefore that of bugs) grows to the limits of our ability to manage that complexity.

Software Testing: Taxonomy

- **By purposes**
  - Correctness testing
    - Black-box
    - White-box
  - Performance testing
  - Reliability testing
    - Robustness testing
      - » Exception handling testing
      - » Stress/load testing
  - Security testing
- **By life cycle phase[PERRY95]**
  - Requirements phase testing
  - Design phase testing
  - Program phase testing
  - Evaluating test results
  - Installation phase testing
  - Acceptance testing
  - Testing changes: maintenance
- **By scope**
  - implied in [BEIZER95]
    - Unit testing
    - Component testing
    - Integration testing
    - System testing
  - or in [PERRY90]
    - Unit testing
    - String testing
    - System testing (α test)
    - Acceptance testing (β test)

Correctness Testing

- **Needs some type of oracles**
- **Black-box testing/behavioral testing**
  - also: data-driven; input/output driven[1]; requirements-based[3]
  - Test data are derived solely from the program structure[9]
  - "Exhaustive input testing"[1]
  - But, what about omissions/extras in spec?
- **White-box testing/structural testing**
  - also: logic-driven[1]; design-based[3]
  - Application of test data derived from the specified functional requirements without regard to the final program structure[9]
  - "Exhaustive path testing"[1]
  - But, what about omissions/extras in code?
- **Other than bugs, we may find:**
  - Features
  - Specification problems
  - Design philosophy (e.g. core dumps v.s. error return code)



Correctness Testing Methods/Tools

- **Control-flow testing**
  - Trace control-flow using control-flow graph; coverage
- **Loop testing**
  - A heuristic technique; should be combined with other methods
  - Applied when there is a loop in graph
- **Data-flow testing**
  - Trace data-flow using data-flow graph; coverage
- **Transaction-flow testing**
  - Testing of on-line applications and batch-processing software
  - Has both control-flow and data-flow attributes

Flow-coverage testing

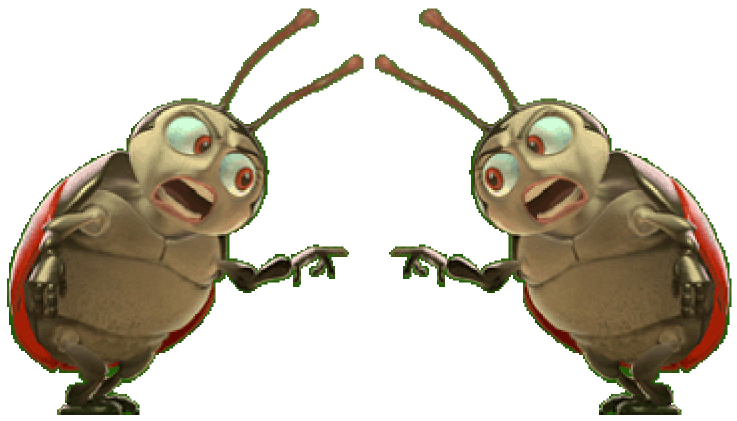
- **Domain testing**
  - Software dominated by numerical processing
- **Syntax testing**
  - Command-driven software and similar applications
- **Finite-state testing**
  - Using finite-state machine model
  - motivated from hardware logic design
  - Excellent for testing menu-driven applications

When to stop testing?

- **Trade-off between budget+time and quality**
  - Part of acceptance testing
- **Stopping rules:**
  - When reliability meets requirement
    - Statistical models
      - » E.g. reliability growth models
    - Data gathering --> modeling -->prediction
    - Not possible to calculate for ultra-dependable system
      - » Because failure data is hard to accumulate
  - When out of resources: test case, money and/or time

Testing is Controversial

- **Alternatives to testing**
  - "human testing[MYERS79]"
    - inspections, walkthroughs, reviews
  - Engineering methods
    - Clean-room v.s. testing
  - Formal Verification v.s. Testing



- **Flames**
  - Traditional coverage-based testing is flawed.
  - Testing can only prove the software is flawed.
  - Inspection/review more effective than testing?
  - "If we have good process, good quality, we don't need much testing"

Conclusions

- **Complete testing is infeasible**
  - Complexity problem
  - Equivalent to Turing halting problem
- **Software testing is immature**
  - Crucial to software quality
- **Testing is more than debugging**
  - For quality assurance, validation and reliability measurement
- **Rules of thumb**
  - Efficiency & effectiveness
  - Automation
- **When to stop: need good metrics**
  - Reliability
  - Time & budget

List of References

- [1][MYERS79] The art of software testing
- [2][BEIZER95] Black-box Testing
- [3][HETZEL88] The Complete Guide to Software Testing
- [4][PHAM95] Software Reliability and Testing, pp29
- [5][KANER93] Testing Computer Software
- [6][PERRY95] Effective Methods for Software Testing, William Perry, 1995 QA76.76.T48P47X
- [7][BEIZER90] Software Testing Techniques
- [8]http://www.cs.jmu.edu/users/foxcj/cs555/Unit12/Testing/index.htm
- [9][PERRY90] A standard for testing application software, William E. Perry, 1990