Testing Strategy

Suradet Jitprapaikulsarn

Derived from Roger S. Pressman, *Software Engineering: A Practitioner's Approach*, 6th Edition, McGraw-Hill 2005

Verification & Validation

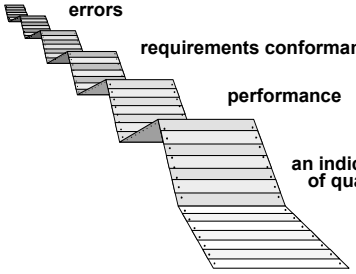
- formal technical reviews
- quality and configuration audits
- performance monitoring
- simulation
- feasibility study
- documentation review
- database review
- algorithm analysis
- development testing
- usability testing
- qualification testing
- installation testing

Software Testing

Testing is the process of exercising a program with the specific intent of finding errors prior to delivery to the end user.

Derived from Roger S. Pressman, Software Engineering: A Practitioner's Approach, 6th Edition, McGraw-Hill, 2005

What Testing Shows



Derived from Roger S. Pressman, Software Engineering: A Practitioner's Approach, 6th Edition, McGraw-Hill, 2005

Who Tests the Software?

developer

Understands the system
but, will test "gently"
and, is driven by "delivery"

independent tester

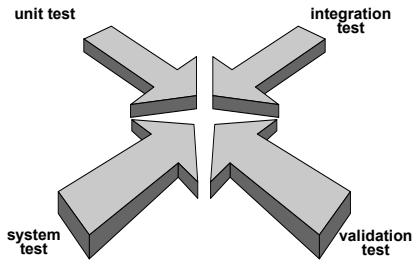
Must learn about the system,
but, will attempt to break it
and, is driven by quality

Derived from Roger S. Pressman, Software Engineering: A Practitioner's Approach, 6th Edition, McGraw-Hill, 2005

Testing Strategy

- Testing planning
- Test case design
- Test execution
- Resultant data collection and evaluation

Testing Strategy

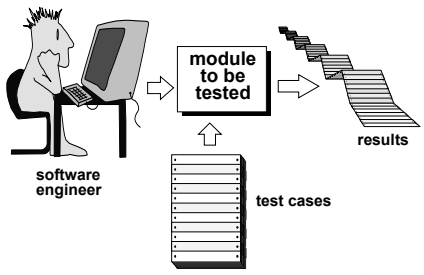


Derived from Roger S. Pressman, Software Engineering: A Practitioner's Approach, 6th Edition, McGraw-Hill, 2005

Testing Strategy

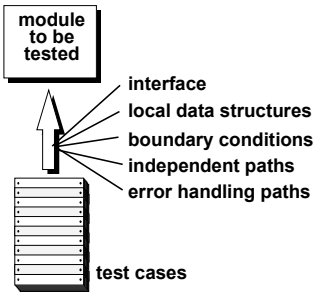
- Quantify requirements
- Start from small to large
- Explicitly define testing objectives
- Know the users
- Emphasize rapid testing
- Build self-diagnose software that accommodating test automation
- Testing as last resource
- Review test strategy and test cases
- Continuous improvement

Unit Testing



Derived from Roger S. Pressman, Software Engineering: A Practitioner's Approach, 6th Edition, McGraw-Hill, 2005

Unit Testing



Derived from Roger S. Pressman, Software Engineering: A Practitioner's Approach, 6th Edition, McGraw-Hill, 2005

Errors commonly found

- Erroneous computation
- Incorrect comparison
- Improper control flow

Errors computation

1. operation precedence
2. mixed mode operations
3. incorrect initialization
4. precision inaccuracy
5. incorrect symbolic representation

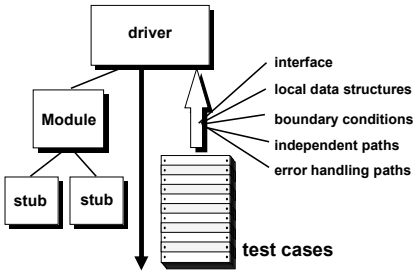
Incorrect Comparison

1. Comparison of different data types
2. Incorrect logical operators or precedence
3. Expectation of equality
4. Incorrect comparison of variables
5. Improper or nonexistent loop termination
6. Failure to exit when divergent iteration is encountered
7. Improperly modified loop variables

Error Handling Check

1. Unintelligible error description
2. Mismatched note
3. External intervention
4. Incorrect handling of exception
5. Insufficient data to identify error spots

Unit Test Environment

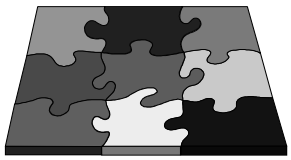


Derived from Roger S. Pressman, Software Engineering: A Practitioner's Approach, 6th Edition, McGraw-Hill, 2005

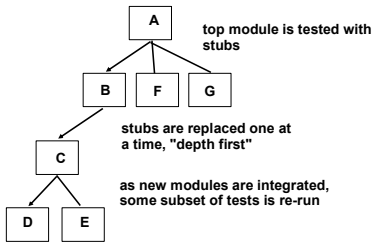
Integration Testing Strategies

Options:

- **the "big bang" approach**
- **an incremental construction strategy**



Top Down Integration

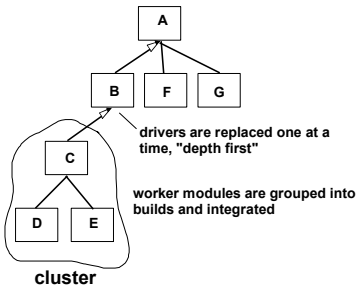


top module is tested with stubs

stubs are replaced one at a time, "depth first"

as new modules are integrated, some subset of tests is re-run

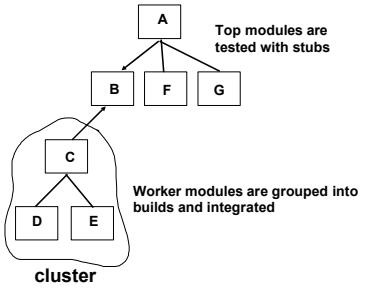
Bottom-Up Integration



drivers are replaced one at a time, "depth first"

worker modules are grouped into builds and integrated

Sandwich Testing



Derived from Roger S. Pressman, Software Engineering: A Practitioner's Approach, 6th Edition, McGraw-Hill, 2005

Regression Testing

- Re-execution of some subset of tests after changes.
- Reducing the side effects
- 3 different class of test cases
  - A representative test
  - Side-effect test
  - Focus test

Smoke Testing

- Integration test while software is being developed
- Pacing mechanism
- Activities
  - Daily build
  - Show stop tests
  - Smoke test daily

High-Order Testing

- Validation testing
  - Focus is on software requirements
- System testing
  - Focus is on system integration
- Alpha/Beta testing
  - Focus is on customer usage
- Recovery testing
  - forces the software to fail in a variety of ways and verifies that recovery is properly performed
- Security testing
  - verifies that protection mechanisms built into a system will, in fact, protect it from improper penetration
- Stress testing
  - executes a system in a manner that demands resources in abnormal quantity, frequency, or volume
- Performance Testing
  - test the run-time performance of software within the context of an integrated system

Object-Oriented Testing

- begins by evaluating the correctness and consistency of the OOA and OOD models
- testing strategy changes
  - the concept of the 'unit' broadens due to encapsulation
  - integration focuses on classes and their execution across a 'thread' or in the context of a usage scenario
  - validation uses conventional black box methods
- test case design draws on conventional methods, but also encompasses special features

Broadening the View of "Testing"

It can be argued that the review of OO analysis and design models is especially useful because the same semantic constructs (e.g., classes, attributes, operations, messages) appear at the analysis, design, and code level. Therefore, a problem in the definition of class attributes that is uncovered during analysis will circumvent side effects that might occur if the problem were not discovered until design or code (or even the next iteration of analysis).

Testing the CRC Model

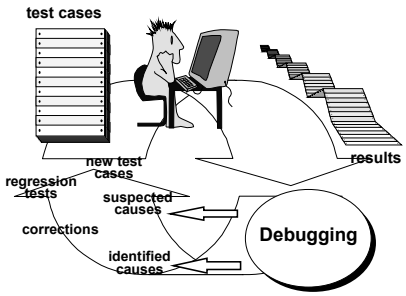
1. Revisit the CRC model and the object-relationship model.
2. Inspect the description of each CRC index card to determine if a delegated responsibility is part of the collaborator's definition.
3. Invert the connection to ensure that each collaborator that is asked for service is receiving requests from a reasonable source.
4. Using the inverted connections examined in step 3, determine whether other classes might be required or whether responsibilities are properly grouped among the classes.
5. Determine whether widely requested responsibilities might be combined into a single responsibility.
6. Steps 1 to 5 are applied iteratively to each class and through each evolution of the OOA model.

OOT Strategy

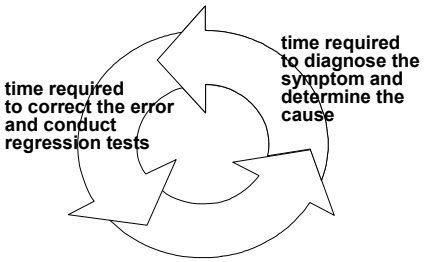
- class testing is the equivalent of unit testing
  - operations within the class are tested
  - the state behavior of the class is examined
- integration applied three different strategies
  - thread-based testing—integrates the set of classes required to respond to one input or event
  - use-based testing—integrates the set of classes required to respond to one use case
  - cluster testing—integrates the set of classes required to demonstrate one collaboration

Debugging: A Diagnostic Process

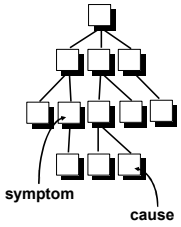
The Debugging Process



Debugging Effort

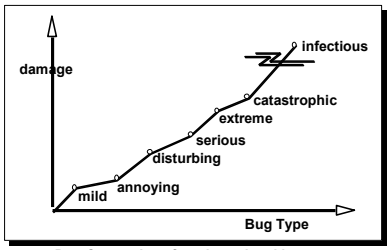


Symptoms & Causes



- symptom and cause may be geographically separated
- symptom may disappear when another problem is fixed
- cause may be due to a combination of non-errors
- cause may be due to a system or compiler error
- cause may be due to assumptions that everyone believes
- symptom may be intermittent

Consequences of Bugs



Bug Categories: **function-related bugs, system-related bugs, data bugs, coding bugs, design bugs, documentation bugs, standards violations, etc.**

Debugging Techniques

- ❑ **brute force / testing**
- ❑ **backtracking**
- ❑ **induction**
- ❑ **deduction**

Debugging: Final Thoughts

1. Don't run off half-cocked, think about the symptom you're seeing.
2. Use tools (e.g., dynamic debugger) to gain more insight.
3. If at an impasse, get help from someone else.
4. Be absolutely sure to conduct regression tests when you do "fix" the bug.