Validation and verification

Lecture 10

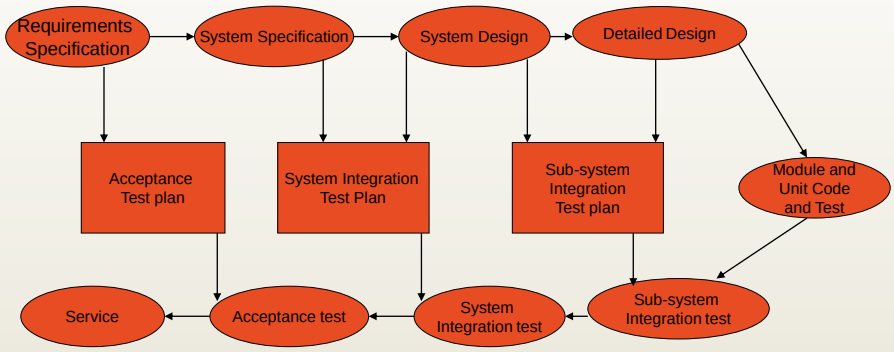
1

Validation and verification

- *Verification* refers to the set of tasks that ensure that software correctly implements a specific function
- *Validation* refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements.
- *Verification:* "Are we building the product right?"
- *Validation:* "Are we building the right product?"
- Process designed to establish confidence that the system is fit for the stated purpose.

2

The Verification and Validation Process



Courtesy: Software Engineering: Somerville

3

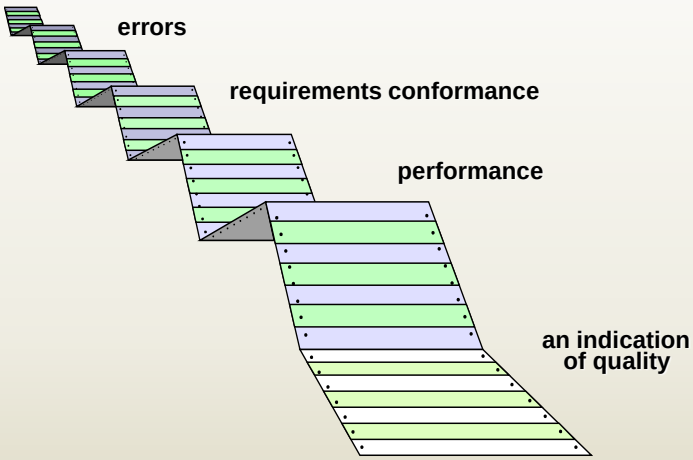
Software Testing

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

- Types of testing
  - ✓ Validation Testing
    - Intended to show that the software meets its requirements
  - ✓ Defect Testing
    - Intended to reveal defects in a system.
    - The goal of defect testing is to find inconsistencies between a program and its specification

4

What Testing Shows



5

Strategic Approach

- Testing begins at the component level and works "outward" toward the integration of the entire computer-based system.
- Different testing techniques are appropriate for different software engineering approaches and at different points in time.
- Testing is conducted by the developer of the software and (for large projects) an independent test group.
- Testing and debugging are different activities, but debugging must be accommodated in any testing strategy.

6

Testing Strategy

- We begin by 'testing-in-the-small' and move toward 'testing-in-the-large'
- For conventional software
  - ✓ The module (component) is our initial focus
  - ✓ Integration of modules follows
- For OO software
  - ✓ our focus when "testing in the small" changes from an individual module (the conventional view) to an OO class that encompasses attributes and operations and implies communication and collaboration

7

Object-Oriented Testing

- begins by evaluating the correctness and consistency of the analysis and design 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

8

How much testing?

- May depend on System Purpose, User Expectation, and Marketing Environment
- System Purpose
  - ✓ More testing would be required for a safety critical system than for a prototype to demonstrate features
- User Expectation
  - ✓ Users may be willing to accept system failures if the benefits of use outweigh the disadvantages. Since the 1990's however, User tolerance of system failures has been generally on the decline.
- Marketing Environment
  - ✓ Competing programs and the price customers are willing to pay need to be taken into account.
  - ✓ If a developing company wants first-to-market advantage, it may decide to release a program before it is fully tested and debugged.
  - ✓ If customers are not willing to pay high prices for software , they may be willing to tolerate faults

9

Who Tests the Software?

developer

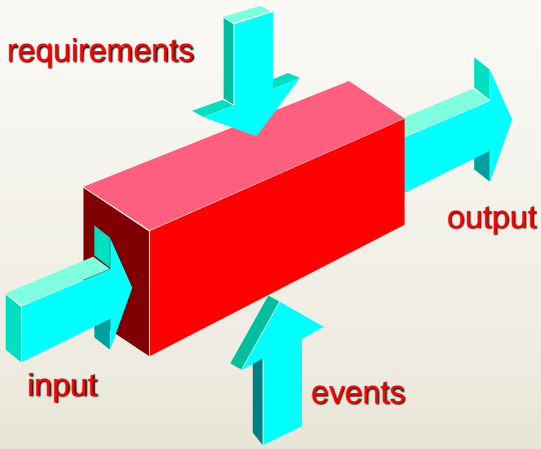
Understands the system
but, will test "gently"
and, is driven by "delivery"

independent tester

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

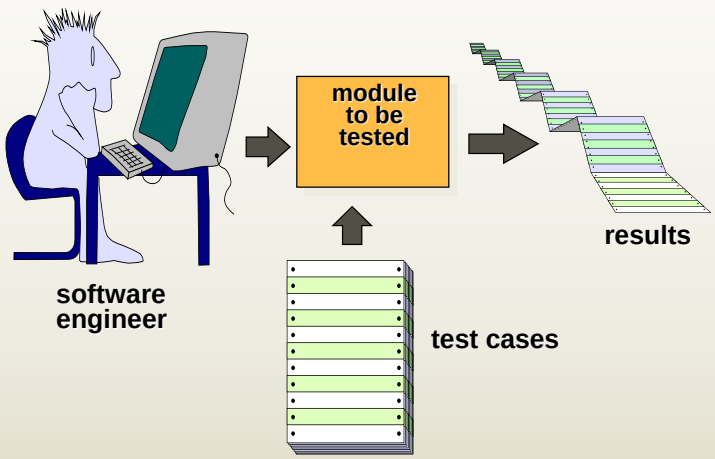
10

Black-Box Testing



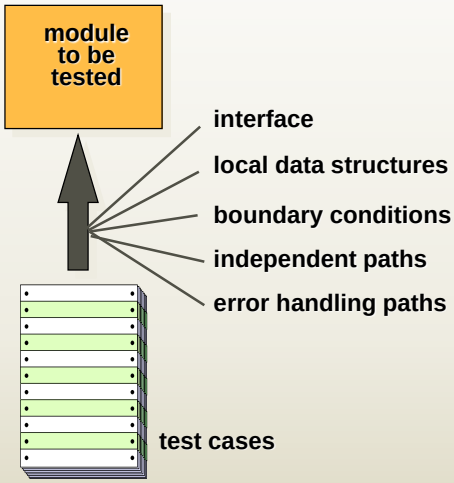
11

Unit Testing



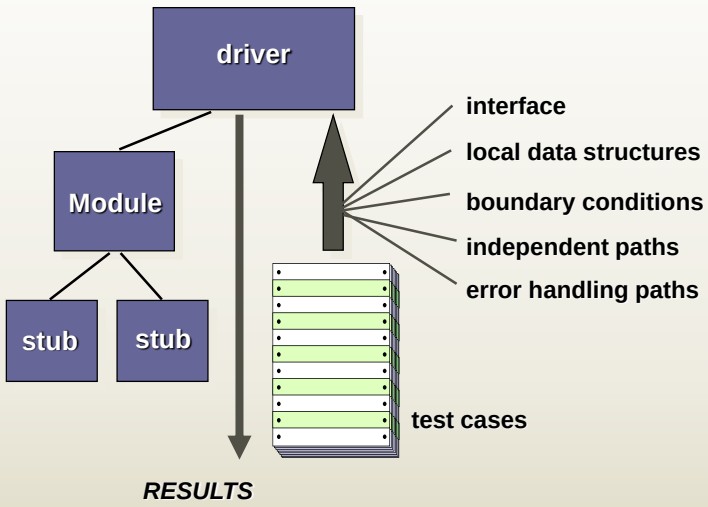
12

Unit Testing



13

Unit Test Environment

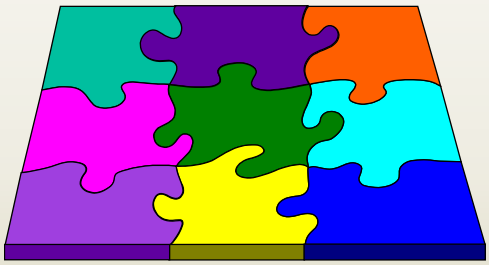


14

Integration Testing Strategies

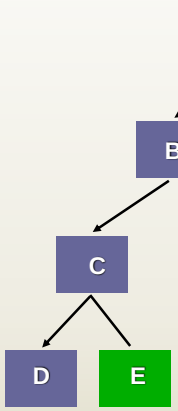
Options:

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



15

Top Down Integration



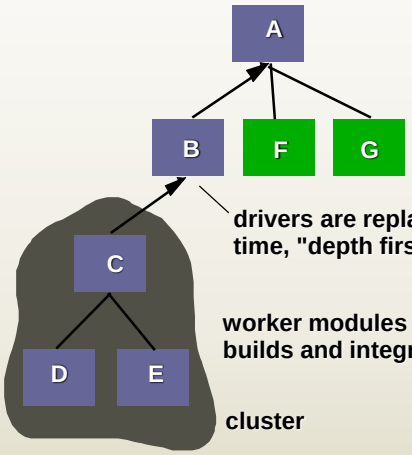
top module is tested with stubs

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

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

16

Bottom-Up Integration



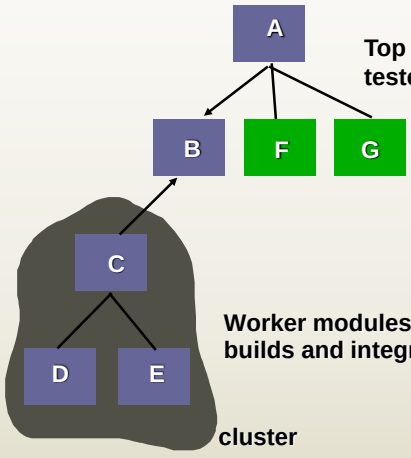
drivers are replaced one at a time, "depth first"

worker modules are grouped into builds and integrated

cluster

17

Sandwich Testing



Top modules are tested with stubs

Worker modules are grouped into builds and integrated

cluster

18

Regression Testing

- *Regression testing* is the re-execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects
- Whenever software is corrected, some aspect of the software configuration (the program, its documentation, or the data that support it) is changed
- Regression testing helps to ensure that changes (due to testing or for other reasons) do not introduce unintended behavior or additional errors
- Regression testing may be conducted manually, by re-executing a subset of all test cases or using automated capture/playback tools.

19

High Order Testing

(after unit 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

20

Test Planning

- The testing process will be composed of one or more test plans
- A test plan may have one or more of the following sections
  - ✓ The testing Process
    - Describing major phases of the process
  - ✓ Requirements Traceability
    - Test should explicitly indicate the if requirements are met
  - ✓ Tested Items
    - Should state the items which are to be tested

21

Test Planning (Cont'd)

- ✓ Testing Schedule
  - Schedule of testing and associated resources
- ✓ Test recording procedures
  - Results of testing , and policies governing the recording of these results
- ✓ Hardware and software requirements
  - Tools required for testing
- ✓ Constraints
  - Constraints such as staff shortages

22

Test Plan Example

- Test Process
  - ✓ Project Quality Manager creates test plan
  - ✓ Developer executes test plan and corrects program if necessary
  - ✓ User executes test plan as User Acceptance Test (UAT)
- Requirements being tested
  - ✓ Current test verifies the access control policy stated in the requirements, indicating which features of the application the user has access to
- Item(s) tested
  - ✓ Login Screen

23

Test Plan Example (Cont'd)

- Test execution and results recording template

Test	Expected Result	Actual Result	Pass/Fail	Comment
System detects if user has logged in correctly	System verifies user name and password against database			
For a teller , only transaction screen and price list displayed	If User is teller , only transaction screen and price list displayed			

24

Test Plan Example (cont'd)

Test	Expected Result	Actual Result	Pass/Fail	Comment
Case1:User name +password Correct	Access Granted			
Case 2:User Name Correct, Password Incorrect				

For vital features, a record of all possible inputs and results can be made , eg. In the table
Note – when detailing situations as below – if there are two input situations , and all Input situation combinations are valid , the number of possible cases is $2^2 = 4$
In general, if there are n possible input situations and all combinations are valid, the Number of test cases = 2^n

25

Active vs Static Testing

- Active Testing
  - ✓ Involves executing the designed system , and documenting results (if fault occurred, or if feature is satisfactory)
  - ✓ Development team will need to be involved
  - ✓ Can be time consuming depending on number of test cases
  - ✓ Can reveal emergent properties
- Static Testing
  - ✓ Involves looking through executing the designed system , and documenting results (if fault occurred, or if feature is satisfactory)
  - ✓ Inspections are an example of static testing, where other individuals "walk through" code with the developer.
  - ✓ Apart from detecting defects , can also be used to check for standards conformance and programming quality.
  - ✓ Can also be time consuming
  - ✓ Can be performed on selected modules before the entire system is complete

26

Common faults detected by Inspections

- Data Faults
  - ✓ Are all variables initialized before use?
  - ✓ Have all required constants been named?
  - ✓ Should the array upper bound be size of array, or size -1?
  - ✓ Is there a delimiter for character strings?
- Control Faults
  - ✓ Are conditions used correct?
  - ✓ Will all loops terminate?
  - ✓ Are compound statements correctly bracketed?
  - ✓ Are all cases accounted for in case statements?
  - ✓ If a break is required after case statements , is it included?
- Input/output faults
  - ✓ Are all input variables used?
  - ✓ Are all output variables assigned a value before they are output?
  - ✓ Can unexpected inputs cause corruption?

27

Common faults detected by Inspections (cont'd)

- Interface faults
  - ✓ Do all function and method calls have the correct number of parameters?
  - ✓ Do formal and actual parameter types match?
  - ✓ Are the parameters in the right order?
  - ✓ If components access shared memory, do they have the same model of the shared memory structure?
- Storage Management Faults
  - ✓ If a linked structure is modified, have all links been correctly reassigned?
  - ✓ If dynamic storage is used, has space been allocated correctly?
  - ✓ Is space explicitly de-allocated after it is no longer required?
- Exception Management Faults
  - ✓ Have all possible error management conditions been taken into account?

28

Organizing the testing process

- In deciding what to include in test cases, several approaches which can be used are:
- Requirements based testing
  - ✓ Testing the system based on documented requirements
- Partition based testing
  - ✓ Testing based on inputs which have common properties (eg – items from a selected menu)
- Structural testing
  - ✓ Test based on modules in the system (also called "white box" testing- because of the need to know details of the system to test for characteristics)
- One commonly used approach is to start executing high level Requirements testing , and progressively add more detailed instances of partition based testing and structural testing where applicable.

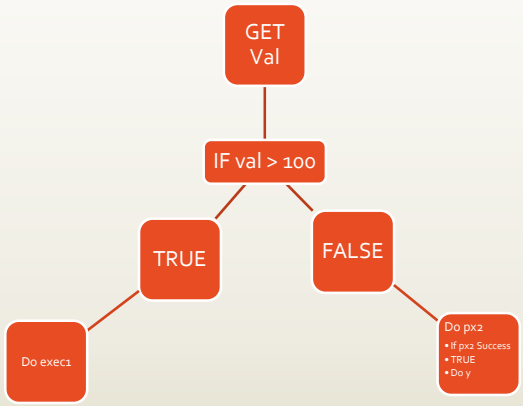
29

Path Testing

- Structural testing activity aimed at exercising every independent execution path through a component or program.
- All statements should be executed at least once at the end of a well designed path test.
- Normally used during components or smaller modules due to feasibility
- Involves traversing general program flow through every single condition , both true and false.
- Number of independent paths in a program flow is called its cyclomatic complexity.
- If there are no goto statements, the cyclomatic complexity is one more than the number of conditions in the program.

30

Path Testing Example



31

System Testing

- Aimed at testing an entire system
- Involves
  - ✓ Integration testing
    - Putting together features and testing
    - Integration for testing may be done
      - Top Down
      - Bottom up
    - Regression testing (re-running previous tests) are often done
  - ✓ Release testing or user acceptance testing (UAT)
    - Usually black box testing
  - ✓ May include performance testing

32

Automating testing

- Testing can be a time intensive and laborious process (which naturally translates to expense)
- Regression tests , to make sure that previously implemented features still work properly , also need to be executed
- Exacerbating the issue is the fact that if errors are detected , the test process may have to be repeated
- The availability of automated processes can help to control testing costs and speed up the process
- It will make sense to automate a test if the time taken to prepare and execute the automated test is comparable to a small number of cycles of a manual test
- Automated tests may include
  - ✓ Automated static analysis
  - ✓ A software testing workbench

33

Automated Static Analysis

- Software tools can be used to scan source program text and detect possible faults and anomalies, using
- Control flow analysis
  - ✓ Looking for issues such as loops with multiple entry/exit points and unreachable code
- Data Use Analysis
  - ✓ Detects Eg. Variables declared but never used
- Interface analysis
  - ✓ Detects eg. Consistency of declarations and use
- Information flow analysis
  - ✓ Identifies dependence between input and output variables
- Path analysis
  - ✓ Identifies all possible paths through a program

34

Software testing workbenches

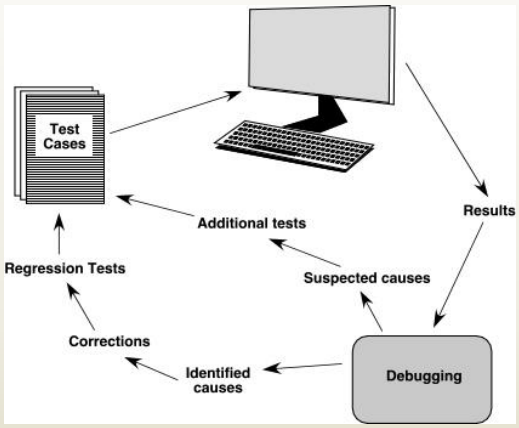
- Integrated set of tools to support testing process
- May include automated test execution support as well as tools such as :
- Test managers
  - ✓ Manages running of tests
- Test data generators
  - ✓ Creates data – either from database or randomly
- Oracles
  - ✓ Predicts expected test results, normally either previous versions or prototypes
- File Comparators
  - ✓ Compares expected with previous test results and reports differences
- Report generator
  - ✓ For formatted reporting or test results
- Dynamic analyzers
  - ✓ Adds code to count number of times each statement is executed
- Simulators
  - ✓ Can be use to simulate factors such as target environment and user interface

35

Debugging: A Diagnostic Process

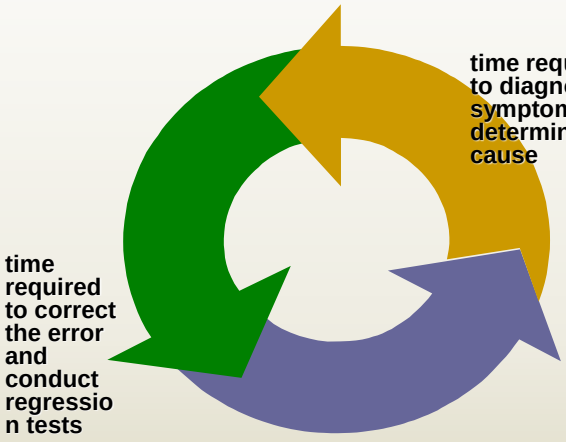
36

The Debugging Process



37

Debugging Effort



38

Correcting the Error

- *Is the cause of the bug reproduced in another part of the program?*
  - ✓ In many situations, a program defect is caused by an erroneous pattern of logic that may be reproduced elsewhere.
- *What "next bug" might be introduced by the fix I'm about to make?*
  - ✓ Before the correction is made, the source code (or, better, the design) should be evaluated to assess coupling of logic and data structures.
- *What could we have done to prevent this bug in the first place?*
  - ✓ This question is the first step toward establishing a statistical software quality assurance approach.
  - ✓ If you correct the process as well as the product, the bug will be removed from the current program and may be eliminated from all future programs.

39

REFERENCES

- The materials in this lecture are primarily based on the text:
  - ✓ Software Engineering : A Practitioner's Approach, 7th Edition by Roger S. Pressman
- Additional supplemental materials included are taken from:
  - ✓ Software Engineering, 9th Edition, by Ian Sommerville
  - ✓ Object-Oriented Software Engineering, 3rd Edition, by Bernd Bruegge and Allen H. Dutoit

40