# IF2261 Software Engineering

## Software Testing

Program Studi Teknik Informatika
STEI ITB



# Software Testing Objectives

- Testing is the process of executing a program with the intent of finding errors.
- A good test case is one with a high probability of finding an as-yet undiscovered error.
- A successful test is one that discovers an as-yet-undiscovered error.

**** SEPA 6th ed, Roger S. Pressman***

## Strategic Approach to Software Testing

- Testing begins at the component level and works outward toward the integration of the entire computer-based system.
- Different testing techniques are appropriate at different points in time.
- The developer of the software conducts testing and may be assisted by independent test groups (ITG) for large projects.
- Testing and debugging are different activities.
- Debugging must be accommodated in any testing strategy.

**** SEPA 6th ed, Roger S. Pressman***



## Verification and Validation

- Make a distinction between
  - verification (are we building the product right?) and
  - validation (are we building the right product?)
- Software testing is only one element of Software Quality Assurance (SQA)
- Quality must be built in to the development process, you can't use testing to add quality after the fact

**** SEPA 6th ed, Roger S. Pressman***

# Organizing for Software Testing

- The role of the Independent Test Group (ITG) is to remove the conflict of interest inherent when the builder is testing his or her own product.
- Misconceptions regarding the use of independent testing teams are
  - The developer should do no testing at all
  - Software is tossed "over the wall" to people to test it mercilessly
  - Testers are not involved with the project until it is time for it to be tested
- The developer and ITGC must work together throughout the software project to ensure that thorough tests will be conducted

** SEPA 6th ed, Roger S. Pressman*



# Software Testing Strategy for Traditional Software Architectures

- Unit Testing
  - makes heavy use of testing techniques that exercise specific control paths to detect errors in each software component individually
- Integration Testing
  - focuses on issues associated with verification and program construction as components begin interacting with one another
- Validation Testing
  - provides assurance that the software validation criteria (established during requirements analysis) meets all functional, behavioral, and performance requirements
- System Testing
  - verifies that all system elements mesh properly and that overall system function and performance has been achieved

** SEPA 6th ed, Roger S. Pressman*

## Software Testing Strategy for Object-Oriented Architectures

- Unit Testing
  - components being tested are classes not modules
- Integration Testing
  - as classes are integrated into the architecture, regression tests are run to uncover communication and collaboration errors between objects
- Validation Testing
  - Pretty much the same for both conventional and object-oriented software
- Systems Testing
  - the system as a whole is tested to uncover requirement errors

** SEPA 6th ed, Roger S. Pressman*



# Strategic Testing Issues

- Specify product requirements in a quantifiable manner before testing starts.
- Specify testing objectives explicitly.
- Identify categories of users for the software and develop a profile for each.
- Develop a test plan that emphasizes rapid cycle testing.
- Build robust software that is designed to test itself.
- Use effective formal reviews as a filter prior to testing.
- Conduct formal technical reviews to assess the test strategy and test cases.
- Develop a continuous improvement approach for the testing process.

** SEPA 6th ed, Roger S. Pressman*

# Unit Testing

- Module interfaces are tested for proper information flow.
- Local data are examined to ensure that integrity is maintained.
- Boundary conditions are tested.
- Basis (independent) path are tested.
- All error handling paths should be tested.
- Drivers and/or stubs need to be developed to test incomplete software.

** SEPA 6th ed, Roger S. Pressman*



# Integration Testing

- Top-down integration testing
  - Main control module used as a test driver and stubs are substitutes for components directly subordinate to it.
  - Subordinate stubs are replaced one at a time with real components (following the depth-first or breadth-first approach).
  - Tests are conducted as each component is integrated.
  - On completion of each set of tests and other stub is replaced with a real component.
  - Regression testing may be used to ensure that new errors not introduced.
- Bottom-up integration testing
  - Low level components are combined into clusters that perform a specific software function.
  - A driver (control program) is written to coordinate test case input and output.
  - The cluster is tested.
  - Drivers are removed and clusters are combined moving upward in the program structure.

** SEPA 6th ed, Roger S. Pressman*

# Integration Testing (2)

- Regression testing - used to check for defects propagated to other modules by changes made to existing program
  - Representative sample of existing test cases is used to exercise all software functions.
  - Additional test cases focusing software functions likely to be affected by the change.
  - Tests cases that focus on the changed software components.
- Smoke testing
  - Software components already translated into code are integrated into a build.
  - A series of tests designed to expose errors that will keep the build from performing its functions are created.
  - The build is integrated with the other builds and the entire product is smoke tested daily (either top-down or bottom integration may be used).

**** SEPA 6th ed, Roger S. Pressman***



# General Software Test Criteria

- Interface integrity
  - internal and external module interfaces are tested as each module or cluster is added to the software
- Functional validity
  - test to uncover functional defects in the software
- Information content
  - test for errors in local or global data structures
- Performance
  - specified performance bounds are tested

**** SEPA 6th ed, Roger S. Pressman***

# Object-Oriented Unit Testing

- smallest testable unit is the encapsulated class or object
- similar to unit testing of conventional software
- do not test operations in isolation from one another
- driven by class operations and state behavior, not algorithmic detail and data flow across module interface

** SEPA 6th ed, Roger S. Pressman*



# Object-Oriented Integration Testing

- Focuses on groups of classes that collaborate or communicate in some manner
- Integration of operations one at a time into classes is often meaningless
- Approaches:
  - thread-based testing - testing all classes required to respond to one system input or event
  - use-based testing - begins by testing independent classes (classes that use very few server classes) first and the dependent classes that make use of them
- Cluster testing - groups of collaborating classes are tested for interaction errors
- Regression testing is important as each thread, cluster, or subsystem is added to the system

** SEPA 6th ed, Roger S. Pressman*

# Validation Testing

- Focuses on visible user actions and user recognizable outputs from the system
- Validation tests are based on the use-case scenarios, the behavior model, and the event flow diagram created in the analysis model
  - Must ensure that each function or performance characteristic conforms to its specification.
  - Deviations (deficiencies) must be negotiated with the customer to establish a means for resolving the errors.
- Configuration review or audit is used to ensure that all elements of the software configuration have been properly developed, cataloged, and documented to allow its support during its maintenance phase.

**** SEPA 6th ed, Roger S. Pressman***



# Acceptance Testing

- Making sure the software works correctly for intended user in his or her normal work environment.
- Alpha test - version of the complete software is tested by customer under the supervision of the developer at the developer's site
- Beta test - version of the complete software is tested by customer at his or her own site without the developer being present

**** SEPA 6th ed, Roger S. Pressman***

# System Testing

- Recovery testing
  - checks the system's ability to recover from failures
- Security testing
  - verifies that system protection mechanism prevent improper penetration or data alteration
- Stress testing
  - program is checked to see how well it deals with abnormal resource demands (i.e., quantity, frequency, or volume)
- Performance testing
  - designed to test the run-time performance of software, especially real-time software

** SEPA 6th ed, Roger S. Pressman*



# Debugging

- Debugging (removal of a defect) occurs as a consequence of successful testing.
- Some people are better at debugging than others.
- Common approaches:
  - **Brute force** - memory dumps and run-time traces are examined for clues to error causes
  - **Backtracking** - source code is examined by looking backwards from symptom to potential causes of errors
  - **Cause elimination** - uses binary partitioning to reduce the number of locations potential (where errors can exist)

** SEPA 6th ed, Roger S. Pressman*

# Bug Removal Considerations

- Is the cause of the bug reproduced in another part of the program?
- What "next bug" might be introduced by the fix that is being proposed?
- What could have been done to prevent this bug in the first place?

** SEPA 6th ed, Roger S. Pressman*



# Software Testability Checklist

- Operability
  - the better it works the more efficiently it can be tested
- Observabilty
  - what you see is what you test
- Controllability
  - the better software can be controlled the more testing can be automated and optimized
- Decomposability
  - by controlling the scope of testing, the more quickly problems can be isolated and retested intelligently
- Simplicity
  - the less there is to test, the more quickly we can test
- Stability
  - the fewer the changes, the fewer the disruptions to testing
- Understandability
  - the more information known, the smarter the testing

** SEPA 6th ed, Roger S. Pressman*

# Good Test Attributes

- A good test has a high probability of finding an error.
- A good test is not redundant.
- A good test should be best of breed.
- A good test should not be too simple or too complex.

** SEPA 6th ed, Roger S. Pressman*



# Test Case Design Strategies

- Black-box or behavioral testing
  - knowing the specified function a product is to perform and demonstrating correct operation based solely on its specification without regard for its internal logic
- White-box or glass-box testing
  - knowing the internal workings of a product, tests are performed to check the workings of all possible logic paths

** SEPA 6th ed, Roger S. Pressman*

# White-Box Testing

- White-Box Testing Questions
  - Can you guarantee that all independent paths within a module will be executed at least once?
  - Can you exercise all logical decisions on their true and false branches?
  - Will all loops execute at their boundaries and within their operational bounds?
  - Can you exercise internal data structures to ensure their validity?
- Basis Path Testing
  - White-box technique usually based on the program flow graph
  - The cyclomatic complexity of the program computed from its flow graph using the formula V(G) = E - N + 2 or by counting the conditional statements in the program design language (PDL) representation and adding 1
  - Determine the basis set of linearly independent paths
  - Prepare test cases that will force the execution of each path in the basis set.

**** SEPA 6th ed, Roger S. Pressman***



# White-Box Testing (2)

- Control Structure Testing
  - White-box technique focusing on control structures present in the software
  - Condition testing (e.g., branch testing)
    - focuses on testing each decision statement in a software module, it is important to ensure coverage of all logical combinations of data that may be processed by the module (a truth table may be helpful)
  - Data flow testing
    - selects test paths based according to the locations of variable definitions and uses in the program (e.g., definition use chains)
  - Loop testing
    - focuses on the validity of the program loop constructs (i.e., simple loops, concatenated loops, nested loops, unstructured loops), involves checking to ensure loops start and stop when they are supposed to (unstructured loops should be redesigned whenever possible)

**** SEPA 6th ed, Roger S. Pressman***

# Black-Box Testing

- Black-Box Testing Questions
  - How is functional validity tested?
  - How is system behavior and performance tested?
  - What classes of input will make good test cases?
  - Is the system particularly sensitive to certain input values?
  - How are the boundaries of a data class isolated?
  - What data rates and data volume can the system tolerate?
  - What effect will specific combinations of data have on system operation?

**** SEPA 6th ed, Roger S. Pressman***



# Black-Box Testing (2)

- Graph-based Testing Methods

  Black-box methods based on the nature of the relationships (links) among the program objects (nodes), test cases are designed to traverse the entire graph
  - Transaction flow testing - nodes represent steps in some transaction and links represent logical connections between steps that need to be validated
  - Finite state modeling - nodes represent user observable states of the software and links represent transitions between states
  - Data flow modeling - nodes are data objects and links are transformations from one data object to another
  - Timing modeling - nodes are program objects and links are sequential connections between these objects, link weights are required execution times

**** SEPA 6th ed, Roger S. Pressman***

# Black-Box Testing (3)

- Equivalence Partitioning
  - Black-box technique that divides the input domain into classes of data from which test cases can be derived
  - An ideal test case uncovers a class of errors that might require many arbitrary test cases to be executed before a general error is observed
  - Equivalence class guidelines:
    - If input condition specifies a range, one valid and two invalid equivalence classes are defined
    - If an input condition requires a specific value, one valid and two invalid equivalence classes are defined
    - If an input condition specifies a member of a set, one valid and one invalid equivalence class is defined
    - If an input condition is Boolean, one valid and one invalid equivalence class is defined

**** SEPA 6th ed, Roger S. Pressman***



# Black-Box Testing (4)

- Boundary Value Analysis
  - Black-box technique that focuses on the boundaries of the input domain rather than its center
  - BVA guidelines:
    - If input condition specifies a range bounded by values a and b, test cases should include a and b, values just above and just below a and b
    - If an input condition specifies and number of values, test cases should be exercise the minimum and maximum numbers, as well as values just above and just below the minimum and maximum values
    - Apply guidelines 1 and 2 to output conditions, test cases should be designed to produce the minimum and maxim output reports
    - If internal program data structures have boundaries (e.g., size limitations), be certain to test the boundaries

**** SEPA 6th ed, Roger S. Pressman***

# Black-Box Testing (5)

- Comparison Testing
  - Black-box testing for safety critical systems in which independently developed implementations of redundant systems are tested for conformance to specifications
  - Often equivalence class partitioning is used to develop a common set of test cases for each implementation
- Orthogonal Array Testing
  - Black-box technique that enables the design of a reasonably small set of test cases that provide maximum test coverage
  - Focus is on categories of faulty logic likely to be present in the software component (without examining the code)
  - Priorities for assessing tests using an orthogonal array
    - Detect and isolate all single mode faults
    - Detect all double mode faults
    - Multimode faults

**** SEPA 6th ed, Roger S. Pressman***