Software testing

Objectives

- To discuss the distinctions between validation testing and defect testing
- To describe the principles of system and component testing
- To describe strategies for generating system test cases
- To understand the essential characteristics of tool used for test automation

How does software fail?

- Wrong requirement: not what the customer wants
- Missing requirement
- Requirement impossible to implement
- Faulty design
- Faulty code
- Improperly implemented design

Testing goals

- Fault identification: what fault caused the failure?
- Fault correction: change the system
- Fault removal: take out the fault

Terminology

- **Mistake** – a human action that produces an incorrect result.
- **Fault [or Defect]** – an incorrect step, process, or data definition in a program.
- **Failure** – the inability of a system or component to perform its required function within the specified performance requirement.
- **Error** – the difference between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition.
- **Specification** – a document that specifies in a complete, precise, verifiable manner, the requirements, design, behavior, or other characteristic of a system or component, and often the procedures for determining whether these provisions have been satisfied.
- We observe errors, which can often be associated with failures. But the ultimate cause of the fault is often very hard to find.

Types of faults

- Algorithmic fault
- Syntax fault
- Computation and precision fault
- Documentation fault
- Stress or overload fault
- Capacity or boundary fault
- Timing or coordination fault
- Throughput or performance fault
- Recovery fault
- Hardware and system software fault
- Documentation fault
- Standards and procedures fault

Levels of testing

Testing type	Specification	General scope	Opacity	Who does it?
Unit	Low-level design; actual code	Classes	White box	Programmer
Integration	Low level design; High level design	Multiple classes	White box; black box	Programmer
Functional	High level design	Whole product	Black box	Independent tester
System	Requirements Analysis	Whole product in environment	Black box	Independent tester
Acceptance	Requirements Analysis	Whole product in environment	Black box	Customer
Beta	Ad hoc	Whole product in environment	Black box	Customer
Regression	Changed documentation; High level design	Any of the above	Black box; white box	Programmers or independent testers

IBM orthogonal defect classification

Fault type	*Meaning*
Function	Fault that affects capability, end-user interfaces, product interfaces, interface with hardware architecture, or global data structure
Interface	Fault in interacting with other components or drivers via calls, macros, control blocks or parameter lists
Checking	Fault in program logic that fails to validate data and values properly before they are used
Assignment	Fault in data structure or code block initialization.
Timing/serialization	Fault that involves timing of shared and real-time resources
Build/package/merge	Fault that occurs because of problems in repositories, management changes, or version control
Documentation	Fault that affects publications and maintenance notes
Algorithm	Fault involving efficiency or correctness of algorithm or data structure but not design

Typical inspection preparation and meeting times

Development artifact	*Preparation time*	*Meeting time*
Requirements document	25 pages per hour	12 pages per hour
Functional specification	45 pages per hour	15 pages per hour
Logic specification	50 pages per hour	20 pages per hour
Source code	150 lines of code per hour	75 lines of code per hour
User documents	35 pages per hour	20 pages per hour

- Faults found during discovery activities.

Discovery activity	*Faults found per thousand lines of code*
Requirements review	2.5
Design review	5.0
Code inspection	10.0
Integration test	3.0
Acceptance test	2.0

Proving code correct

- Formal proof techniques
- Symbolic execution
- Automated theorem-proving

- Will discuss these next lecture

Test thoroughness

- Statement testing
- Branch testing
- Path testing
- Definition-use testing
- All-uses testing
- All-predicate-uses/some-computational-uses testing
- All-computational-uses/some-predicate-uses testing

Comparing techniques

- Fault discovery percentages by fault origin

Discovery technique	*Requirements*	*Design*	*Coding*	*Documentation*
Prototyping	40	35	35	15
Requirements review	40	15	0	5
Design review	15	55	0	15
Code inspection	20	40	65	25
Unit testing	1	5	20	0

- Effectiveness of fault discovery techniques. (Jones 1991)

	Requirements faults	*Design faults*	*Code faults*	*Documentation faults*
Reviews	Fair	Excellent	Excellent	Good
Prototypes	Good	Fair	Fair	Not applicable
Testing	Poor	Poor	Good	Fair
Correctness proofs	Poor	Poor	Fair	Fair

Test planning

- Establish test objectives
- Design test cases
- Write test cases
- Test test cases
- Execute tests
- Evaluate test results

The testing process

- Component (or unit) testing
  - Testing of individual program components;
  - Usually the responsibility of the component developer (except sometimes for critical systems);
  - Tests are derived from the developer's experience.
- System testing
  - Testing of groups of components integrated to create a system or sub-system;
  - The responsibility of an independent testing team;
  - Tests are based on a system specification.

Testing process goals

- Validation testing
  - To demonstrate to the developer and the system customer that the software meets its requirements;
  - A successful test shows that the system operates as intended.
- Defect testing
  - To discover faults or defects in the software where its behavior is incorrect or not in conformance with its specification;
  - A successful test is a test that makes the system perform incorrectly and so exposes a defect in the system.
  - Tests show the presence not the absence of defects

Testing policies

- Only exhaustive testing can show a program is free from defects. However, exhaustive testing is impossible
- Testing policies define the approach to be used in selecting system tests:
  - All functions accessed through menus should be tested;
  - Combinations of functions accessed through the same menu should be tested;
  - Where user input is required, all functions must be tested with correct and incorrect input.

Integration testing

- Involves building a system from its components and testing it for problems that arise from component interactions.
- Top-down integration
  - Develop the skeleton of the system and populate it with components.
- Bottom-up integration
  - Integrate infrastructure components then add functional components.
- To simplify error localization, systems should be incrementally integrated.

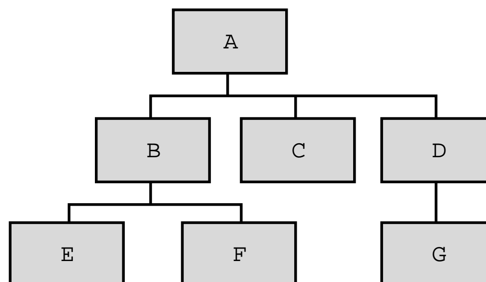
Integration testing

- Bottom-up
- Top-down
- Big-bang
- Sandwich testing
- Modified top-down
- Modified sandwich

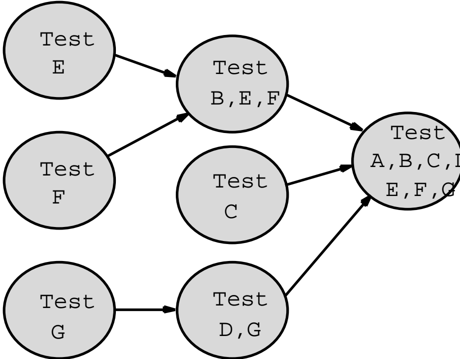
(Non-)Testing

- Beta-testing – Relied on too heavily by large vendors, like Microsoft.
  - Allow early adopters easy access to a new product on the condition that they report errors to vendor. A good way to stress test a new system.

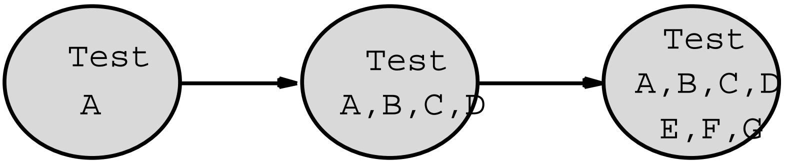
Sample system



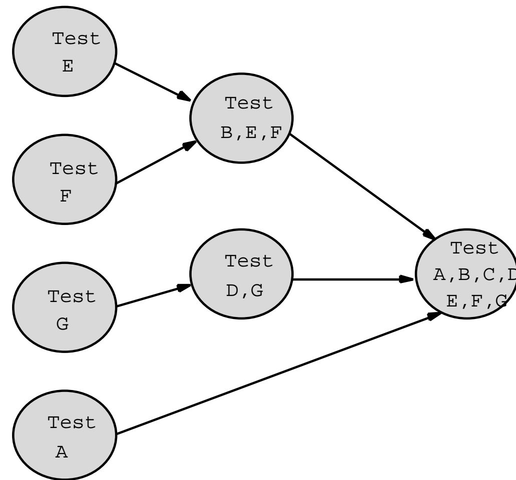
Bottom up testing



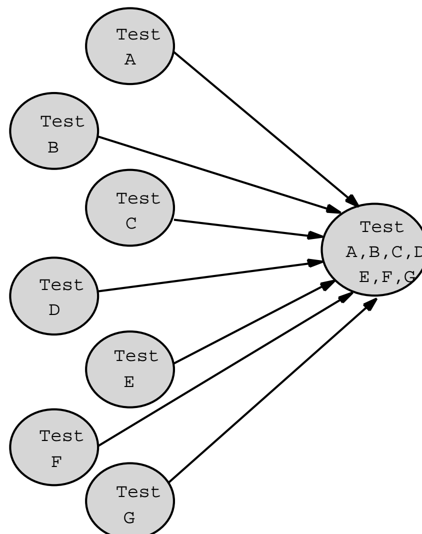
Top down testing



Sandwich testing



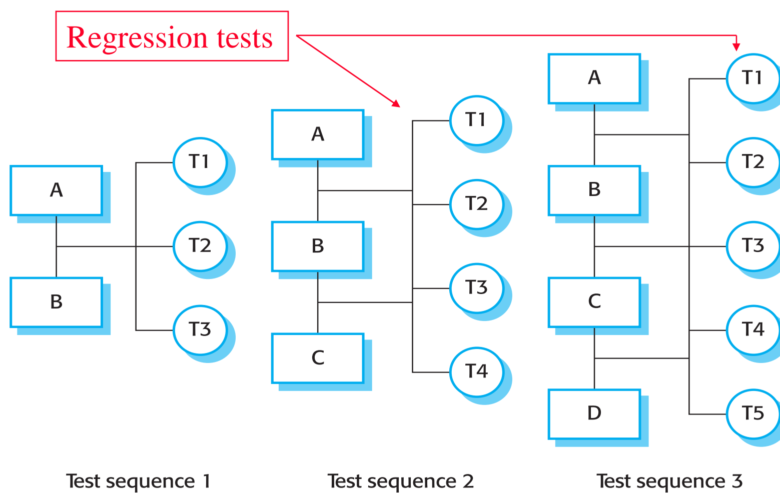
Big bang testing



Comparison of integration strategies

	Bottom-up	Top-down	Modified top-down	Big-bang	Sandwich	Modified sandwich
Integration	Early	Early	Early	Late	Early	Early
Time to basic working program	Late	Early	Early	Late	Early	Early
Component drivers needed	Yes	No	Yes	Yes	Yes	Yes
Stubs needed	No	Yes	Yes	Yes	Yes	Yes
Work parallelism at beginning	Medium	Low	Medium	High	Medium	High
Ability to test particular paths	Easy	Hard	Easy	Easy	Medium	Easy
Ability to plan and control sequence	Easy	Hard	Hard	Easy	Hard	Hard

Incremental integration testing



Testing approaches

- Black box testing, also called functional testing and behavioral testing, focuses on determining whether or not a program does what it is supposed to do based on its functional requirements.
  - No knowledge of internal structure of code
- White-box testing is testing that takes into account the internal mechanism of a system or component. White-box testing is also known as structural testing, clear box testing, and glass box testing.
  - Knowledge of code

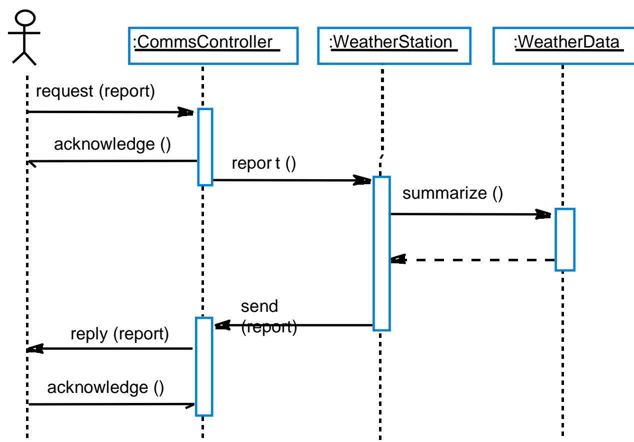
Testing guidelines

- Testing guidelines are hints for the testing team to help them choose tests that will reveal defects in the system
  - Choose inputs that force the system to generate all error messages;
  - Design inputs that cause buffers to overflow;
  - Repeat the same input or input series several times;
  - Force invalid outputs to be generated;
  - Force computation results to be too large or too small.

Use cases

- We saw use cases before under software design processes.
- Use cases can be a basis for deriving the tests for a system. They help identify operations to be tested and help design the required test cases.
- From an associated sequence diagram, the inputs and outputs to be created for the tests can be identified.

Collect weather data sequence chart



Performance testing

- Part of release testing may involve testing the emergent properties of a system, such as performance and reliability.
- Performance tests usually involve planning a series of tests where the load is steadily increased until the system performance becomes unacceptable.

Performance tests

- Stress tests
- Volume tests
- Configuration tests
- Compatibility tests
- Regression tests
- Security tests
- Timing tests
- Environmental tests
- Quality tests
- Recovery tests
- Maintenance tests
- Documentation tests
- Human factors (usability) tests

Stress testing

- Exercises the system beyond its maximum design load. Stressing the system often causes defects to come to light.
- Stressing the system test failure behaviour. Systems should not fail catastrophically. Stress testing checks for unacceptable loss of service or data.
- Stress testing is particularly relevant to distributed systems that can exhibit severe degradation as a network becomes overloaded.

Interface testing

- Objectives are to detect faults due to interface errors or invalid assumptions about interfaces.
- Particularly important for object-oriented development as objects are defined by their interfaces.

Interface errors

- Interface misuse
  - A calling component calls another component and makes an error in its use of its interface e.g. parameters in the wrong order.
- Interface misunderstanding
  - A calling component embeds assumptions about the behaviour of the called component which are incorrect.
- Timing errors
  - The called and the calling component operate at different speeds and out-of-date information is accessed.

Interface testing guidelines

- Design tests so that parameters to a called procedure are at the extreme ends of their ranges.
- Always test pointer parameters with null pointers.
- Design tests which cause the component to fail.
- Use stress testing in message passing systems.
- In shared memory systems, vary the order in which components are activated.

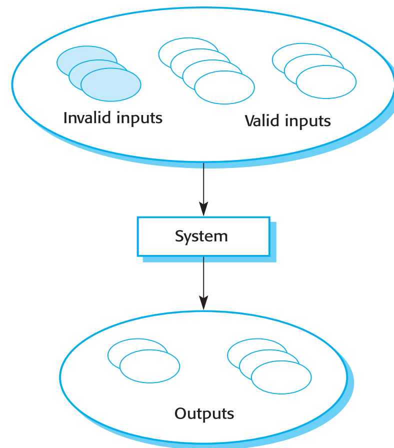
Test case design

- Involves designing the test cases (inputs and outputs) used to test the system.
- The goal of test case design is to create a set of tests that are effective in validation and defect testing.
- Design approaches:
  - Requirements-based testing;
  - Partition testing;
  - Structural testing.

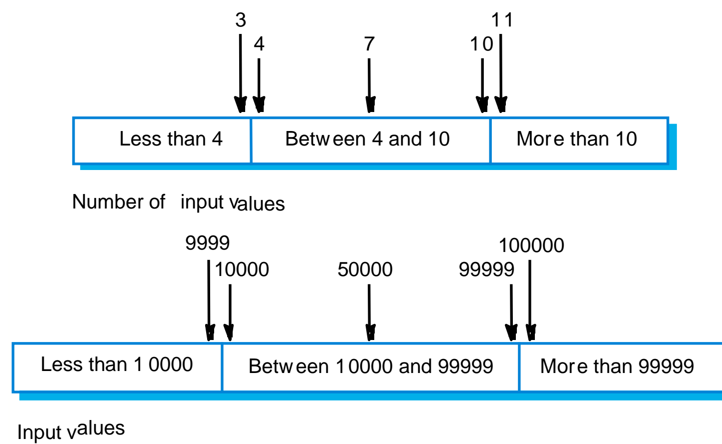
Requirements based testing

- A general principle of requirements engineering is that requirements should be testable.
- Requirements-based testing is a validation testing technique where you consider each requirement and derive a set of tests for that requirement.

Equivalence partitioning



Equivalence partitions



Search routine specification

procedure Search (Key : ELEM ; T: SEQ of ELEM;
Found : **in out** BOOLEAN; L: **in out** ELEM_INDEX) ;

Pre-condition
-- the sequence has at least one element
T'FIRST <= T'LAST
Post-condition
-- the element is found and is referenced by L
(Found and T (L) = Key)
or
-- the element is not in the array
(**not** Found **and**
not (**exists** i, T'FIRST >= i <= T'LAST, T (i) = Key))

Search routine - input partitions

- Inputs which conform to the pre-conditions.
- Inputs where a pre-condition does not hold.
- Inputs where the key element is a member of the array.
- Inputs where the key element is not a member of the array.

Testing guidelines (sequences)

- Test software with sequences which have only a single value.
- Use sequences of different sizes in different tests.
- Derive tests so that the first, middle and last elements of the sequence are accessed.
- Test with sequences of zero length.

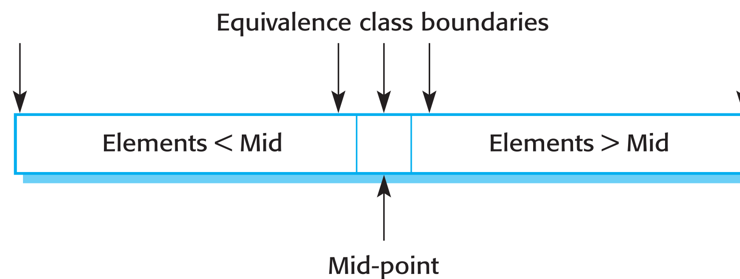
Structural testing

- Sometime called white-box testing.
- Derivation of test cases according to program structure. Knowledge of the program is used to identify additional test cases.
- Objective is to exercise all program statements (not all path combinations).

Binary search - equiv. partitions

- Pre-conditions satisfied, key element in array.
- Pre-conditions satisfied, key element not in array.
- Pre-conditions unsatisfied, key element in array.
- Pre-conditions unsatisfied, key element not in array.
- Input array has a single value.
- Input array has an even number of values.
- Input array has an odd number of values.

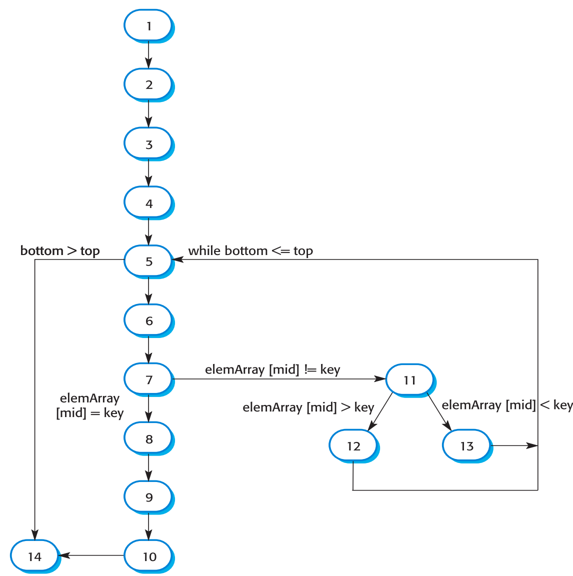
Binary search equiv. partitions



Path testing

- The objective of path testing is to ensure that the set of test cases is such that each path through the program is executed at least once.
- The starting point for path testing is a program flow graph that shows nodes representing program decisions and arcs representing the flow of control.
- Statements with conditions are therefore nodes in the flow graph.

Binary search flow graph



Independent paths

- 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 14
- 1, 2, 3, 4, 5, 14
- 1, 2, 3, 4, 5, 6, 7, 11, 12, 5, …
- 1, 2, 3, 4, 6, 7, 2, 11, 13, 5, …
- Test cases should be derived so that all of these paths are executed
- A dynamic program analyzer may be used to check that paths have been executed

Test automation

- Testing is an expensive process phase. Testing workbenches provide a range of tools to reduce the time required and total testing costs.
- Systems such as Junit support the automatic execution of tests.
- Most testing workbenches are open systems because testing needs are organisation-specific.
- They are sometimes difficult to integrate with closed design and analysis workbenches.

Testing workbench adaptation

- Scripts may be developed for user interface simulators and patterns for test data generators.
- Test outputs may have to be prepared manually for comparison.
- Special-purpose file comparators may be developed.

Automated testing tools

- Code analysis
  - Static analysis
    - code analyzer
    - structure checker
    - data analyzer
    - sequence checker
  - Dynamic analysis
    - program monitor

When to stop testing

- Coverage criteria
- Fault seeding

$$\frac{\text{detected seeded faults}}{\text{total seeded faults}} = \frac{\text{detected non-seeded faults}}{\text{total non-seeded faults}}$$

- Confidence in the software

$$C = \begin{cases} 1 & \text{if } n > N \\ S/(S-N+1) & \text{if } n \leq N \end{cases}$$

$$C = \begin{cases} 1 & \text{if } n > N \\ \binom{S}{s-1} \Big/ \binom{S+N+1}{N+s} & \text{if } n \leq N \end{cases}$$

System testing process

- Function testing: does the integrated system perform as promised by the requirements specification?
- Performance testing: are the non-functional requirements met?
- Acceptance testing: is the system what the customer expects?
- Installation testing: does the system run at the customer site(s)?

Techniques used in system testing

- Build or spin plan for gradual testing
- Configuration management
  - versions and releases
  - production system vs. development system
  - deltas, separate files and conditional compilation
  - change control
- Regression testing

Test team

- Professional testers: organize and run the tests
- Analysts: who created requirements
- System designers: understand the proposed solution
- Configuration management specialists: to help control fixes
- Users: to evaluate issues that arise

Acceptance tests

- Pilot test: install on experimental basis
- Alpha test: in-house test
- Beta test: customer pilot
- Parallel testing: new system operates in parallel with old system

Test documentation

- Test plan: describes system and plan for exercising all functions and characteristics
- Test specification and evaluation: details each test and defines criteria for evaluating each feature
- Test description: test data and procedures for each test
- Test analysis report: results of each test

Testing safety-critical systems

- Design diversity: use different kinds of designs, designers
- Software safety cases: make explicit the ways the software addresses possible problems
  - failure modes and effects analysis
  - hazard and operability studies
- Cleanroom: certifying software with respect to the specification

Key points

- Testing can show the presence of faults in a system; it cannot prove there are no remaining faults.
- Component developers are responsible for component testing; system testing is the responsibility of a separate team.
- Integration testing is testing increments of the system; release testing involves testing a system to be released to a customer.
- Use experience and guidelines to design test cases in defect testing.

Key points

- Interface testing is designed to discover defects in the interfaces of composite components.
- Equivalence partitioning is a way of discovering test cases - all cases in a partition should behave in the same way.
- Structural analysis relies on analysing a program and deriving tests from this analysis.
- Test automation reduces testing costs by supporting the test process with a range of software tools.