Software Testing

- Verification and validation planning
- Software inspections
- Software Inspection vs. Testing
- Automated static analysis
- Cleanroom software development
- System testing

Verification vs validation

- Verification:
  "Are we building the product right”.
- The software should conform to its specification.
- Validation:
  "Are we building the right product”.
- The software should do what the user really requires.

Static and dynamic verification

- Software inspections. Concerned with analysis of the static system representation to discover problems (static verification)
  - May be supplement by tool-based document and code analysis
- Software testing. Concerned with exercising and observing product behaviour (dynamic verification)
  - The system is executed with test data and its operational behaviour is observed

Software inspections

- Software Inspection involves examining the source representation with the aim of discovering anomalies and defects without execution of a system.
- They may be applied to any representation of the system (requirements, design, configuration data, test data, etc.).

Inspections and testing

- Inspections and testing are complementary and not opposing verification techniques.
- Inspections can check conformance with a specification but not conformance with the customer's real requirements.
- Inspections cannot check non-functional characteristics such as performance, usability, etc.
- Management should not use inspections for staff appraisal i.e. finding out who makes mistakes.

Inspection procedure

- System overview presented to inspection team.
- Code and associated documents are distributed to inspection team in advance.
- Inspection takes place and discovered errors are noted.
- Modifications are made to repair errors.
- Re-inspection may or may not be required.
- Checklist of common errors should be used to drive the inspection. Examples: Initialization, Constant naming, loop termination, array bounds…

Inspection roles

Author or owner	The programmer or designer responsible for producing the program or document. Responsible for fixing defects discovered during the inspection process.
Inspector	Finds errors, omissions and inconsistencies in programs and documents. May also identify broader issues that are outside the scope of the inspection team.
Reader	Presents the code or document at an inspection meeting.
Scribe	Records the results of the inspection meeting.
Chairman or moderator	Manages the process and facilitates the inspection. Reports process results to the Chief moderator.
Chief moderator	Responsible for inspection process improvements, checklist updating, standards development etc.

Inspection checks 1

Data faults	Are all program variables initialised before their values are used? Have all constants been named? Should the upper bound of arrays be equal to the size of the array or Size -1? If character strings are used, is a de limiter explicitly assigned? Is there any possibility of buffer overflow?
Control faults	For each conditional statement, is the condition correct? Is each loop certain to terminate? Are compound statements correctly bracketed? In case statements, are all possible cases accounted for? If a break is required after each case in case statements, has it been 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?

Inspection checks 2

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 conditions been taken into account?

Automated static analysis

- Static analysers are software tools for source text processing.
- They parse the program text and try to discover potentially erroneous conditions and bring these to the attention of the V & V team.
- They are very effective as an aid to inspections - they are a supplement to but not a replacement for inspections.

LINT static analysis

```
138% more lint_ex.c
#include <stdio.h>
printarray (Anarray)
 int Anarray;
{   printf(“%d“,Anarray);  }

main ()
{
 int Anarray[5]; int i; char c;
 printarray (Anarray, i, c);
 printarray (Anarray) ;
}

139% cc lint_ex.c
140% lint lint_ex.c

lint_ex.c(10): warning: c may be used before set
lint_ex.c(10): warning: i may be used before set
printarray: variable # of args. lint_ex.c(4) :: lint_ex.c(10)
printarray, arg. 1 used inconsistently lint_ex.c(4) :: lint_ex.c(10)
printarray, arg. 1 used inconsistently lint_ex.c(4) :: lint_ex.c(11)
printf returns value which is always ignored
```

Cleanroom software development

- The name is derived from the 'Cleanroom' process in semiconductor fabrication. The philosophy is defect avoidance rather than defect removal.
- This software development process is based on:
  - Incremental development;
  - Formal specification;
  - Static verification using correctness arguments;
  - Statistical testing to determine program reliability.

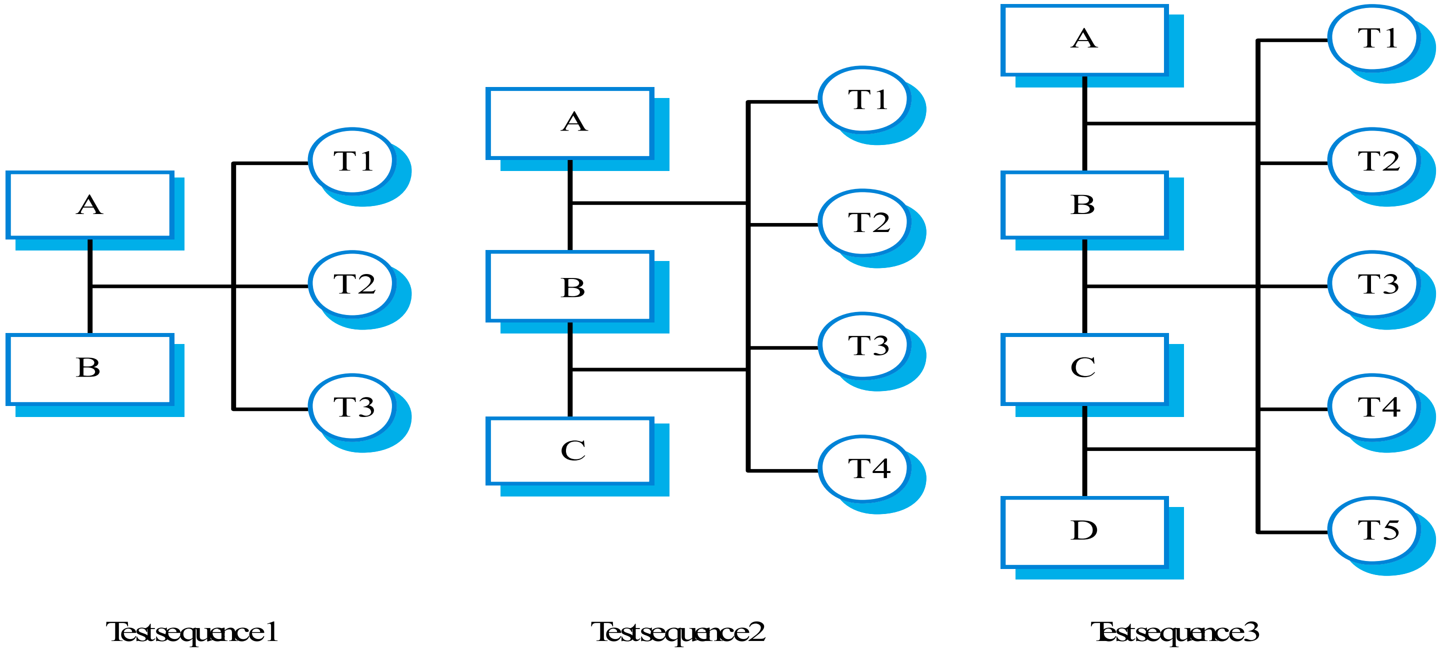
Cleanroom process teams

- Specification team. Responsible for developing and maintaining the system specification.
- Development team. Responsible for developing and verifying the software. The software is NOT executed or even compiled during this process.
- Certification team. Responsible for developing a set of statistical tests to exercise the software after development. Reliability growth models used to determine when reliability is acceptable.

System testing

- Involves integrating components to create a system or sub-system.
- May involve testing an increment to be delivered to the customer.
- Two phases:
  - Integration testing - the test team have access to the system source code. The system is tested as components are integrated.
  - Release testing - the test team test the complete system to be delivered as a black-box.

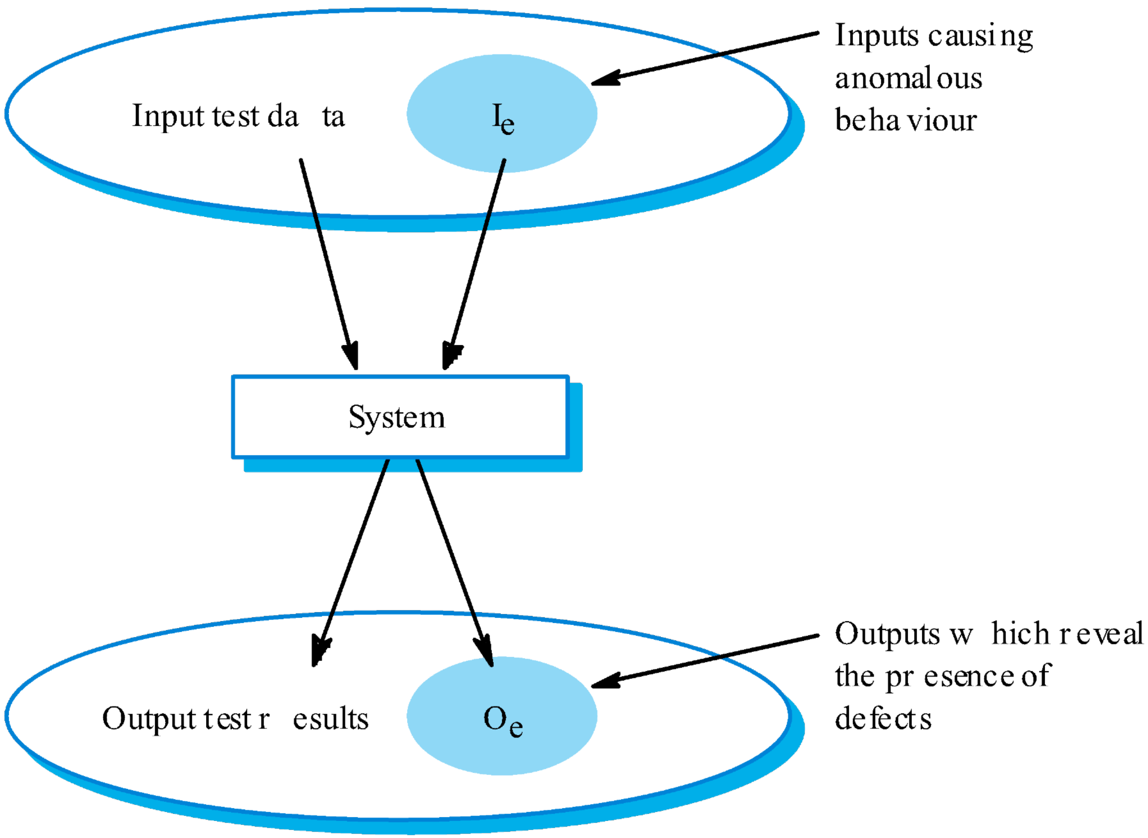
Incremental integration testing



Release testing

- The process of testing a release of a system that will be distributed to customers.
- Primary goal is to increase the supplier's confidence that the system meets its requirements.
- Release testing is usually black-box or functional testing
  - Based on the system specification only;
  - Testers do not have knowledge of the system implementation.

Black-box testing



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.

Component testing

- Component or unit testing is the process of testing individual components in isolation.
- It is a defect testing process.
- Components may be:
  - Individual functions or methods within an object;
  - Object classes with several attributes and methods;
  - Composite components with defined interfaces used to access their functionality.

Object class testing

- Complete test coverage of a class involves
  - Testing all operations associated with an object;
  - Setting and interrogating all object attributes;
  - Exercising the object in all possible states.
- Inheritance makes it more difficult to design object class tests as the information to be tested is not localised.

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 types

- Parameter interfaces
  - Data passed from one procedure to another.
- Shared memory interfaces
  - Block of memory is shared between procedures or functions.
- Procedural interfaces
  - Sub-system encapsulates a set of procedures to be called by other sub-systems.
- Message passing interfaces
  - Sub-systems request services from other sub-system.s

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 (i.e. trace test cases to the requirements)
  - Partition testing;
  - Structural testing.

Partition testing

- Input data and output results often fall into different classes where all members of a class are related.
- Each of these classes is an equivalence partition or domain where the program behaves in an equivalent way for each class member.
- Test cases should be chosen from each partition.

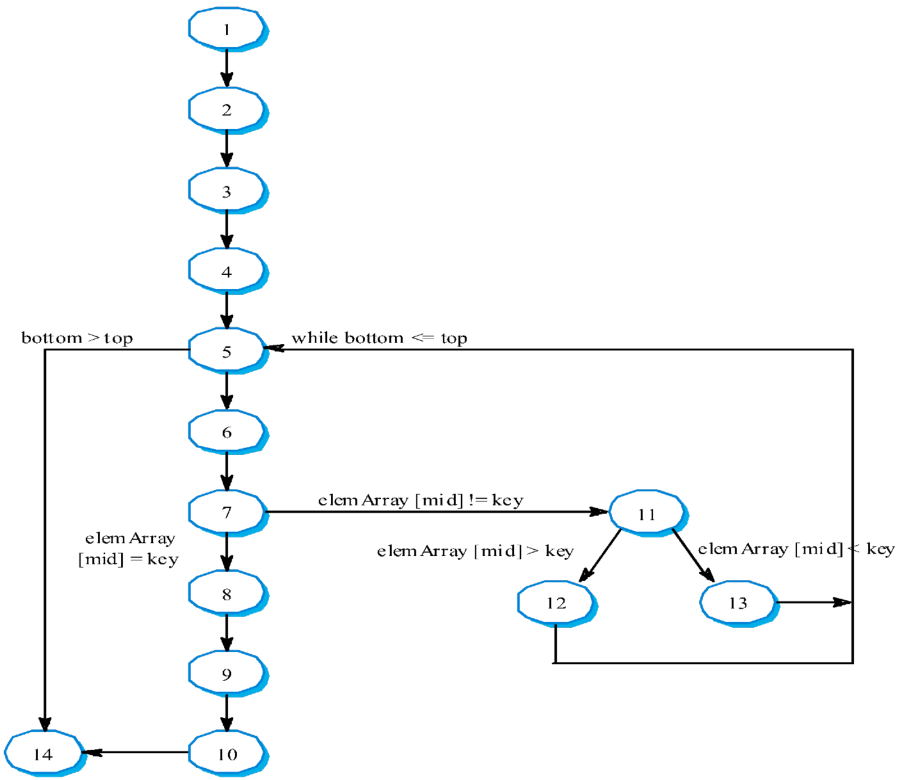
Structural testing

- 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).

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 analyser 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.