

# Forelesning IMT2243 14.(og 22.) april 2009

Temaer :

BUSTA – eksempel på arkitektur / design

Objektorientert Design (sekvensdiagram og design klassediagram)

Verifikasjon og Validering

Foilene er hentet fra nettsidene til lærebokforfatter Ian Sommerville

Pensumlitteratur (og dagens foilsett) :  
Ian Sommerville kap 14, 22. og 23.

---

---

---

---

---

---

---

---

## V& V goals

- Verification and validation should establish confidence that the software is fit for purpose.
- This does NOT mean completely free of defects.
- Rather, it must be good enough for its intended use and the type of use will determine the degree of confidence that is needed.

---

---

---

---

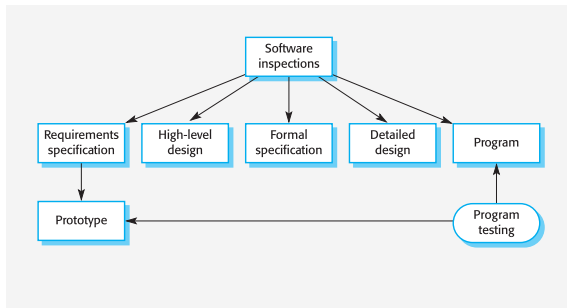
---

---

---

---

## Static and dynamic V&V



---

---

---

---

---

---

---

---

## 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.

---

---

---

---

---

---

---

---

---

---

## Static analysis checks

Fault class	Static analysis check
Data faults	Variables used before initialisation Variables declared but never used Variables assigned twice but never used between assignments Possible array bound violations Undeclared variables
Control faults	Unreachable code Unconditional branches into loops
Input/output faults	Variables output twice with no intervening assignment
Interface faults	Parameter type mismatches Parameter number mismatches Non-usage of the results of functions Uncalled functions and procedures
Storage management faults	Unassigned pointers Pointer arithmetic

---

---

---

---

---

---

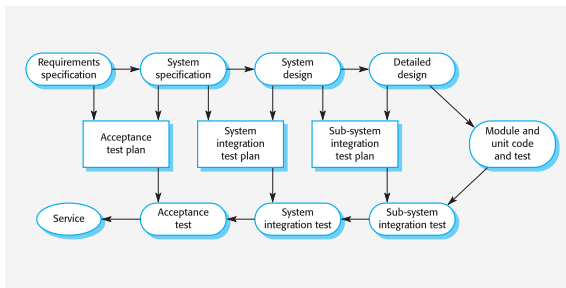
---

---

---

---

## The V-model of development




---

---

---

---

---

---

---

---

---

---

## The software test plan

**The testing process**  
A description of the major phases of the testing process. These might be as described earlier in this chapter.

**Requirements traceability**  
Users are most interested in the system meeting its requirements and testing should be planned so that all requirements are individually tested.

**Tested items**  
The products of the software process that are to be tested should be specified.

**Testing schedule**  
An overall testing schedule and resource allocation for this schedule. This, obviously, is linked to the more general project development schedule.

**Test recording procedures**  
It is not enough simply to run tests. The results of the tests must be systematically recorded. It must be possible to audit the testing process to check that it has been carried out correctly.

**Hardware and software requirements**

This section should set out software tools required and estimated hardware utilisation.

**Constraints**  
Constraints affecting the testing process such as staff shortages should be anticipated in this section.

---

---

---

---

---

---

---

---

---

---

## The testing process

- **Component 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 behaviour 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.

---

---

---

---

---

---

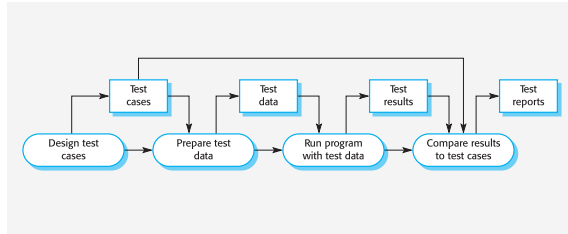
---

---

---

---

## The software testing process



---

---

---

---

---

---

---

---

## 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-systems.

---

---

---

---

---

---

---

---

## 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 localisation, systems should be incrementally integrated.

---

---

---

---

---

---

---

---

## 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.

---

---

---

---

---

---

---

---

## 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.

---

---

---

---

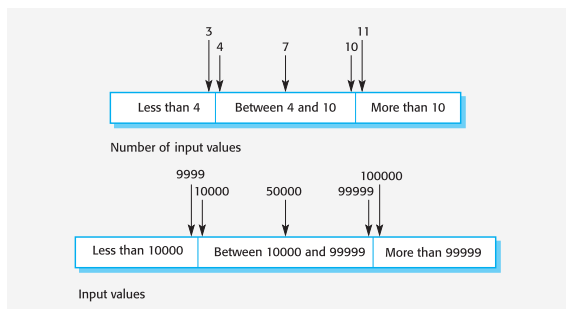
---

---

---

---

## Equivalence partitions



---

---

---

---

---

---

---

---

## 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).

---

---

---

---

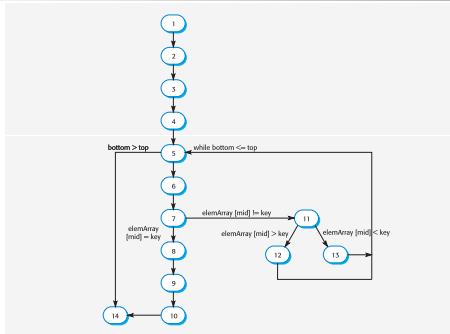
---

---

---

---

## Binary search flow graph



---

---

---

---

---

---

---

---

## 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.

---

---

---

---

---

---

---

---