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

- V and V involves:
  - Verification and validation planning
  - Software inspections
  - Automated static analysis

The V & V process

- Is a whole life-cycle process - V & V must be applied at each stage in the software process.
- Has two principal objectives
  - The discovery of defects in a system
  - The assessment of whether or not the system is usable in an operational situation.
- 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.

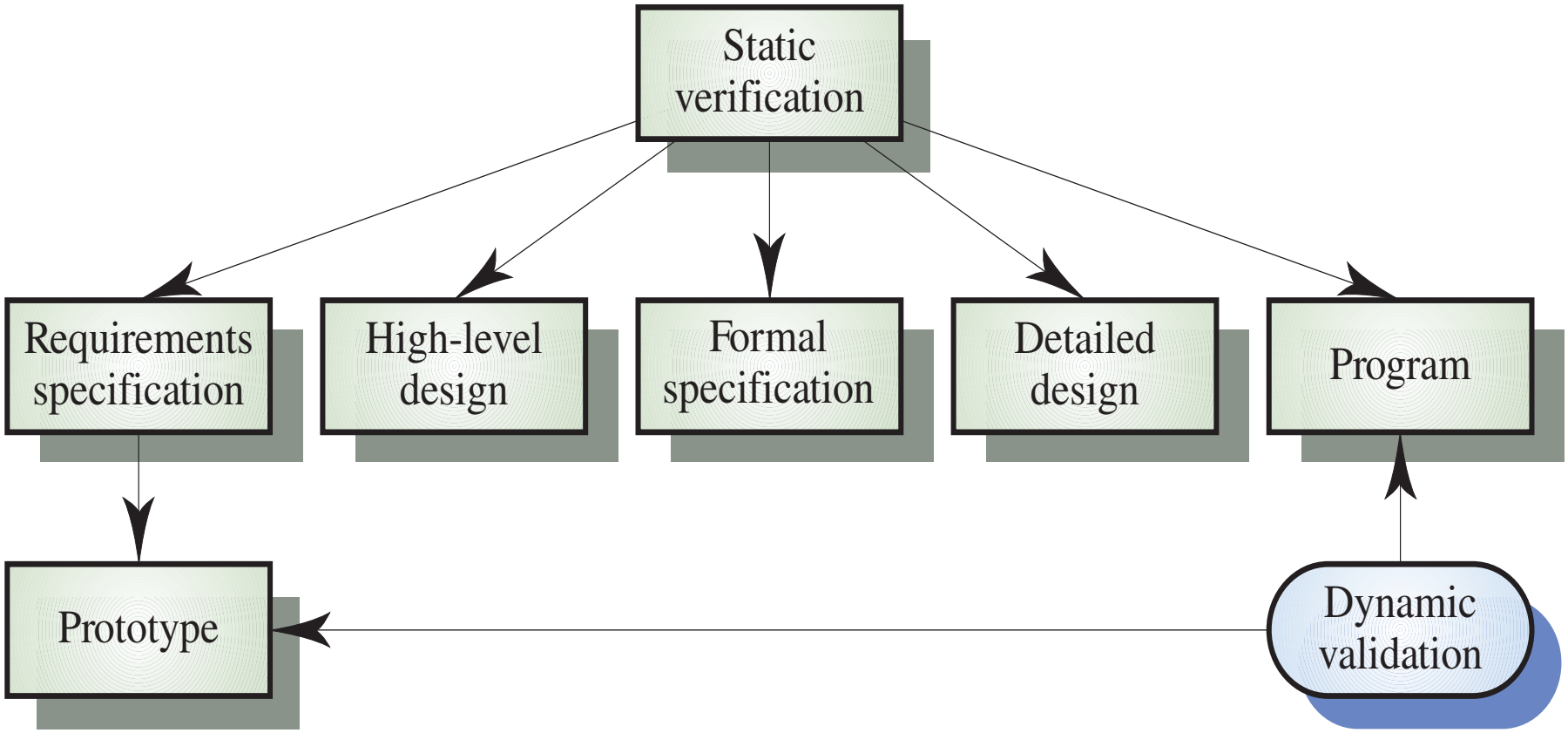
V and V Confidence

- Depends on system's purpose, user expectations and marketing environment
  - Software function
    - The level of confidence depends on how critical the software is to an organisation.
  - User expectations
    - Users may have low expectations of certain kinds of software.
  - Marketing environment
    - Getting a product to market early may be more important than finding defects in the program

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

Static and dynamic V&V

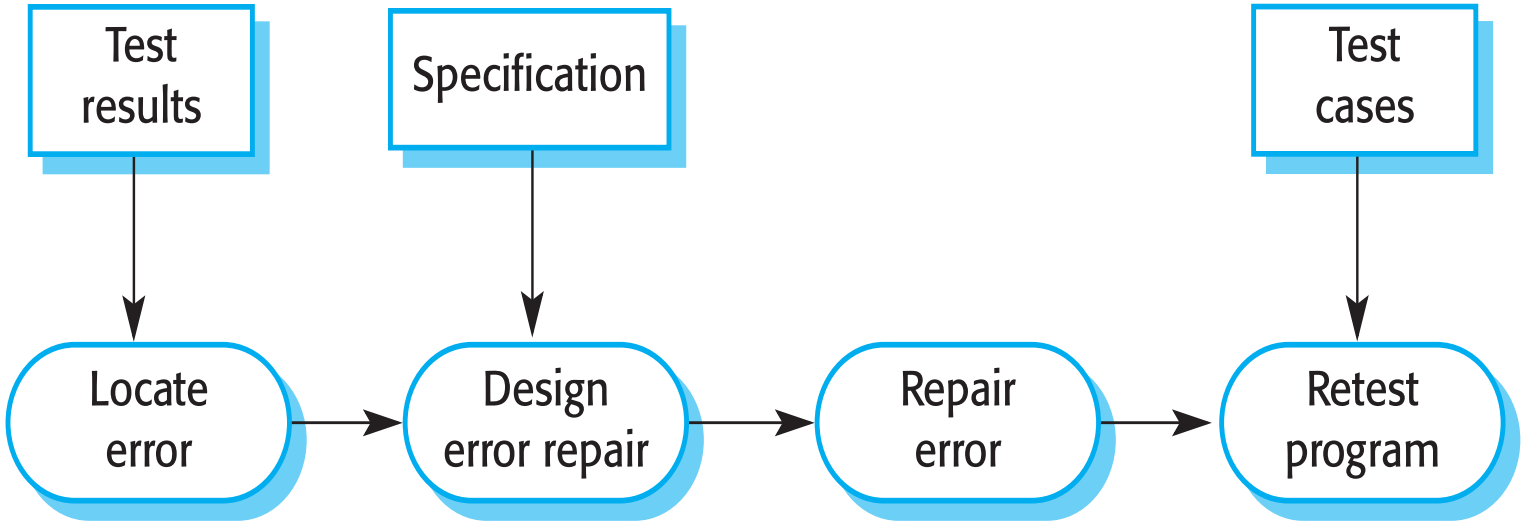


Program testing

- Can reveal the presence of errors NOT their absence
- A successful test is a test which discovers one or more errors
- The only validation technique for non-functional requirements
- Should be used in conjunction with static verification to provide full V&V coverage
- TYPES OF TESTING:
- Defect testing
  - Tests designed to discover system defects.
  - A successful defect test is one which reveals the presence of defects in a system.
- Validation testing
  - Intended to show that the software meets its requirements.
  - A successful test is one that shows that a requirements has been properly implemented.

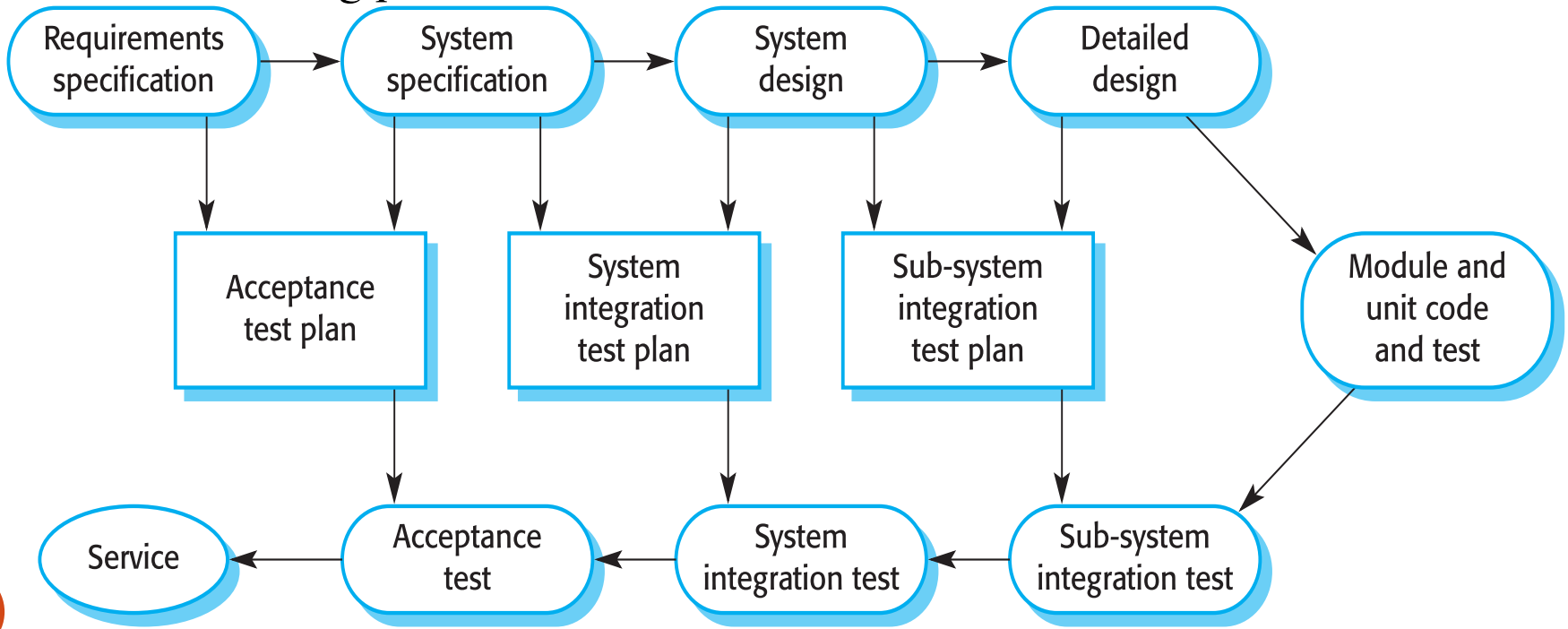
Testing and Debugging

- Defect testing and debugging are distinct processes.
- Verification and validation is concerned with establishing the existence of defects in a program.
- Debugging is concerned with locating and repairing these errors.



V and V Planning

- Careful planning is required to get the most out of testing and inspection processes.
- Planning should start early in the development process.
- The plan should identify the balance between static verification and testing.
- Test planning is about defining standards for the testing process rather than describing product tests.



The structure of a software test plan

- The testing process.
- Requirements traceability.
- Tested items. Testing schedule.
- Test recording procedures.
- Hardware and software requirements.
- Constraints.

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 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.

Software Inspections

- These involve people examining the source representation with the aim of discovering anomalies and defects.
- Inspections do not require execution of a system so may be used before implementation.
- They may be applied to any representation of the system (requirements, design, configuration data, test data, etc.).
- They have been shown to be an effective technique for discovering program errors.
- Inspections and testing are complementary and not opposing verification techniques.
- Both should be used during the V & V process.
- 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.

Program Inspections

- Formalised approach to document reviews
- Intended explicitly for defect detection (not correction).
- Defects may be logical errors, anomalies in the code that might indicate an erroneous condition (e.g. an un-initialised variable) or non-compliance with standards.
- Inspection Pre-Conditions:
- A precise specification must be available.
- Team members must be familiar with the organisation standards.
- Syntactically correct code or other system representations must be available.
- An error checklist should be prepared.
- Management must accept that inspection will increase costs early in the software process.
- Management should not use inspections for staff appraisal ie 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 discovered errors.
- Re-inspection may or may not be required.

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

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 delimiter explicitly assigned? Is there any possibility of a 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?

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.

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

Stages of Static Analysis

- Control flow analysis. Checks for loops with multiple exit or entry points, finds unreachable code, etc.
- Data use analysis. Detects un-initialised variables, variables written twice without an intervening assignment, variables which are declared but never used, etc.
- Interface analysis. Checks the consistency of routine and procedure declarations and their use
- Information flow analysis. Identifies the dependencies of output variables. Does not detect anomalies itself but highlights information for code inspection or review
- Path analysis. Identifies paths through the program and sets out the statements executed in that path. Again, potentially useful in the review process
- Both of these last 2 stages generate vast amounts of information. They must be used with care.

Use of static analysis

- Particularly valuable when a language such as C is used which has weak typing and hence many errors are undetected by the compiler,
- Less cost-effective for languages like Java that have strong type checking and can therefore detect many errors during compilation.
- CONCLUSION:
- Verification and validation are not the same thing. Verification shows conformance with specification; validation shows that the program meets the customer's needs.
- Test plans should be drawn up to guide the testing process.
- Static verification techniques involve examination and analysis of the program for error detection.
- Program inspections are very effective in discovering errors.
- Program code in inspections is systematically checked by a small team to locate software faults.
- Static analysis tools can discover program anomalies which may be an indication of faults in the code.