Verification and Validation

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.

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 useful and useable in an operational situation.

V& V goals

- 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.

V & 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

Program testing

- Can reveal the presence of errors NOT their absence.
- The only validation technique for non-functional requirements as the software has to be executed to see how it behaves.
- 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.
- Debugging involves formulating a hypothesis about program behaviour then testing these hypotheses to find the system error.

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 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 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.

Inspection success

- Many different defects may be discovered in a single inspection.
  - In testing, one defect,may mask another so several executions are required.
- The reuse domain and programming knowledge so reviewers are likely to have seen the types of error that commonly arise.

Inspections and testing

- 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 uninitialised 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.

Use of automatic 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.

Verification and formal methods

- Formal methods can be used when a mathematical specification of the system is produced.
- They are the ultimate static verification technique.
- They involve detailed mathematical analysis of the specification and may develop formal arguments that a program conforms to its mathematical specification.

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.

More details about Software testing

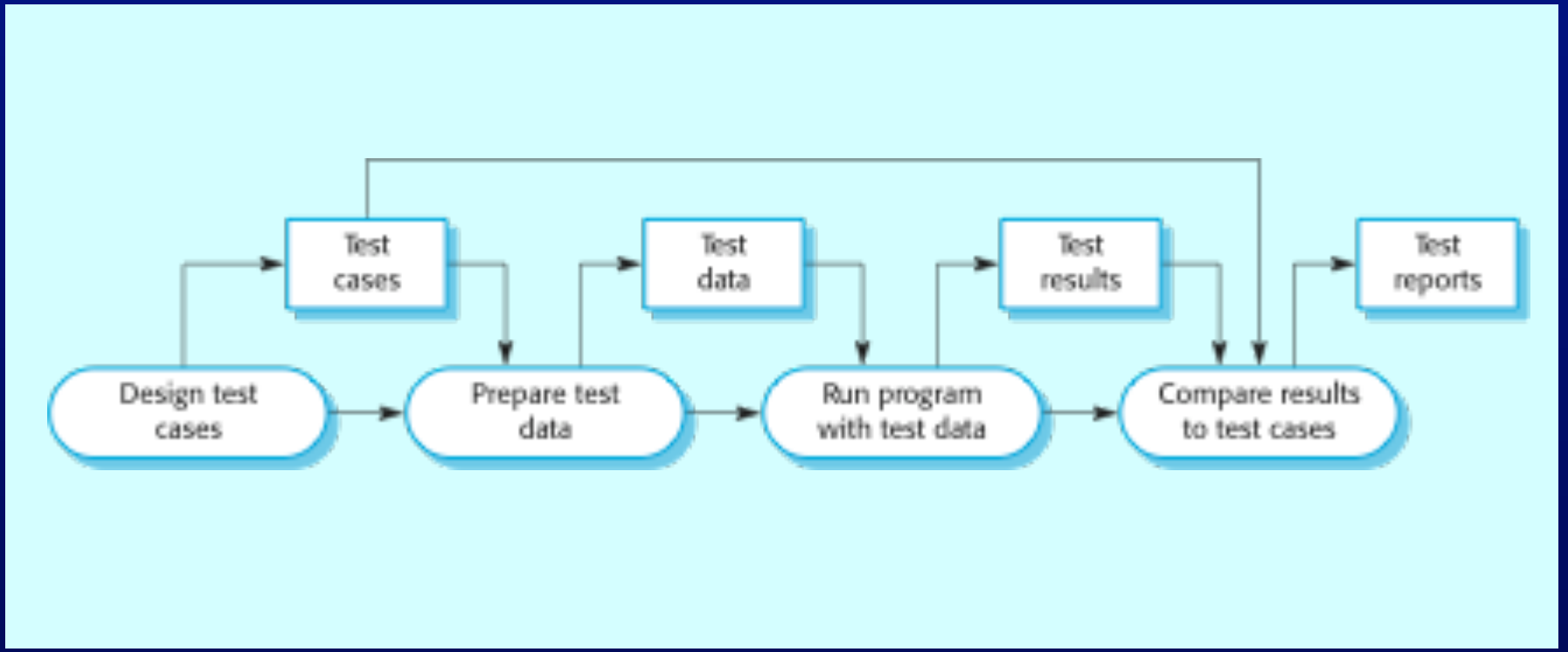
The testing process

- 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.
  - Tests show the presence not the absence of defects

The software testing process



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.

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.

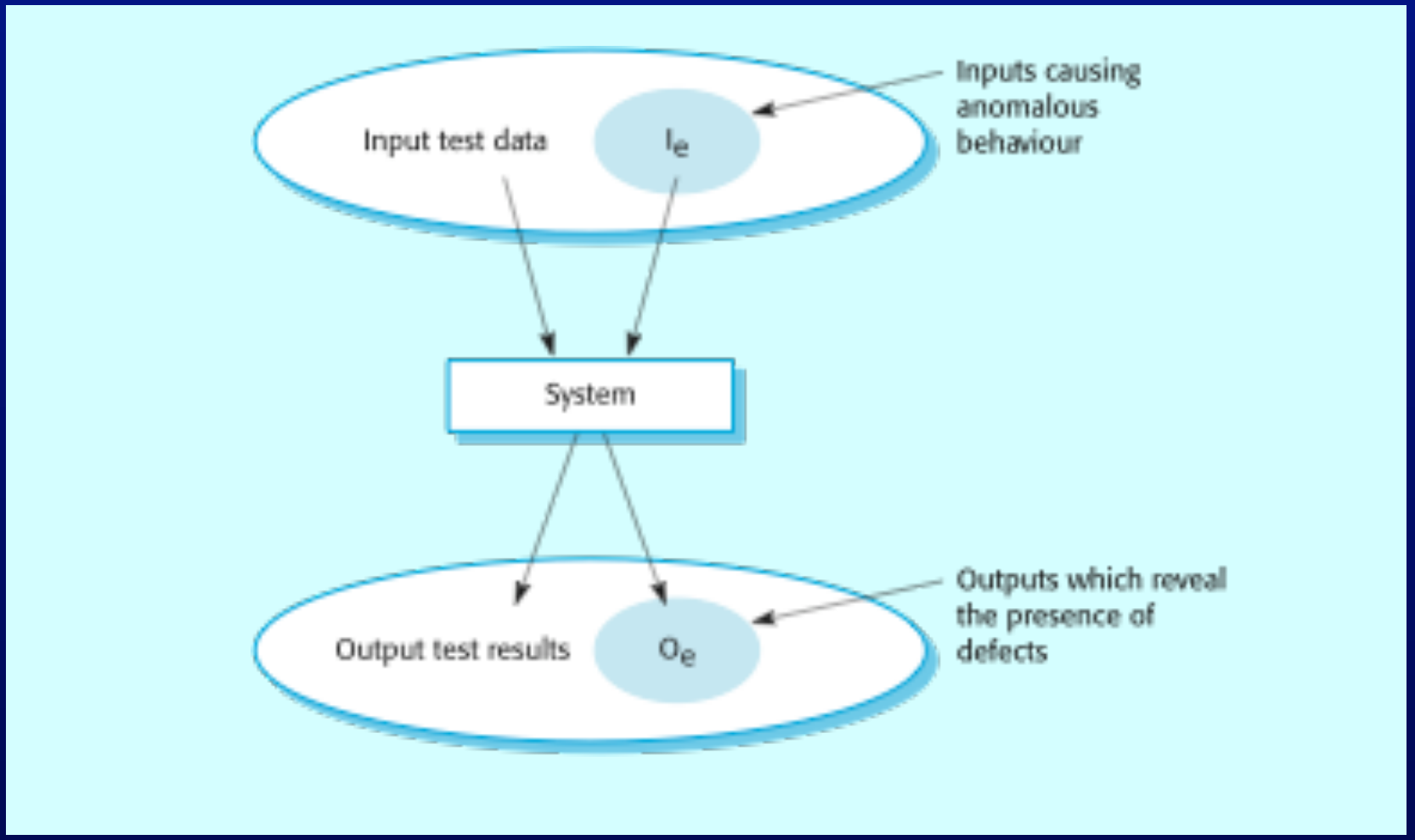
Integration testing

- 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.

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



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

- 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.

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.

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 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.

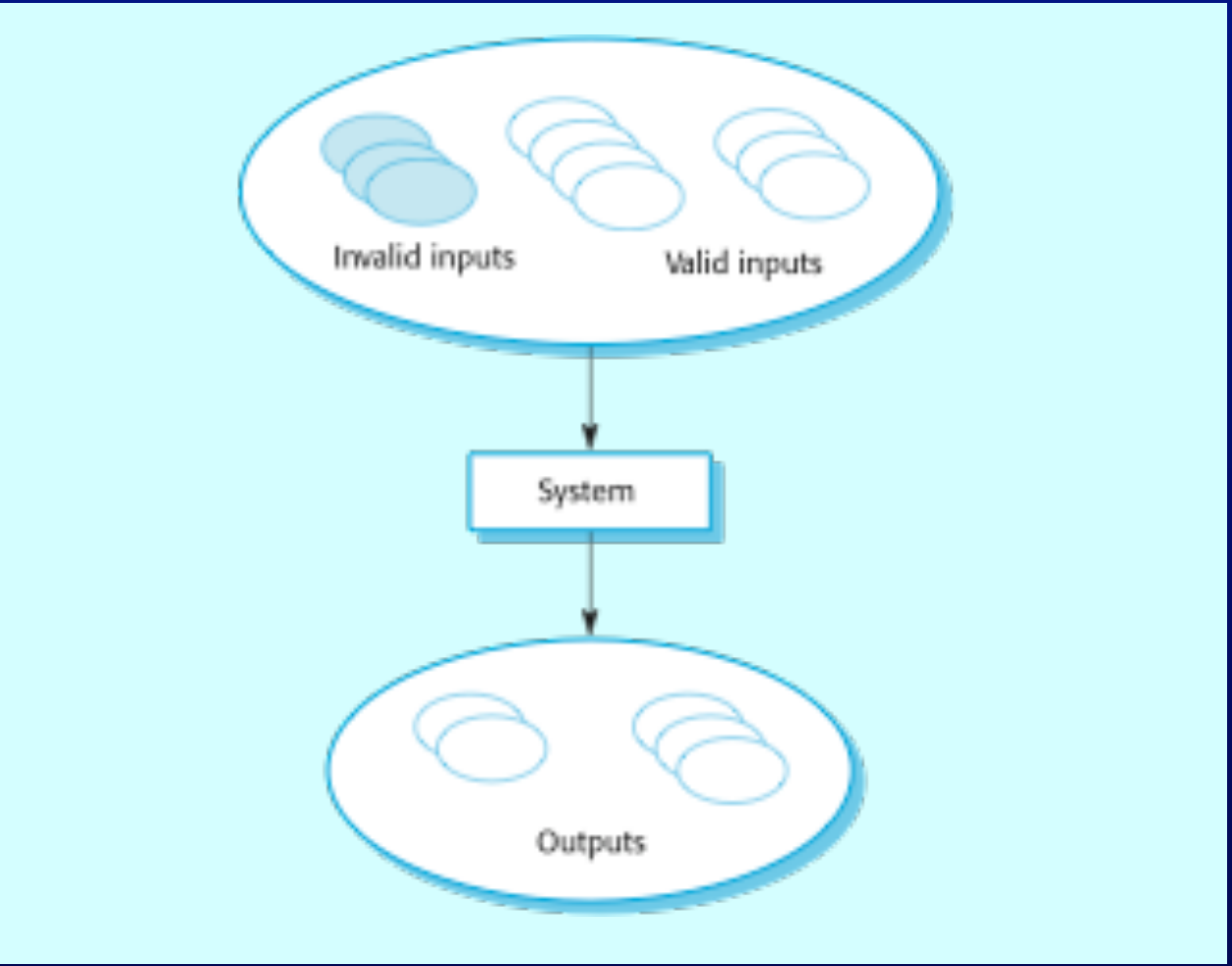
Requirements based testing

- 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.

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.

Equivalence partitioning



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.

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.

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.

Configuration management

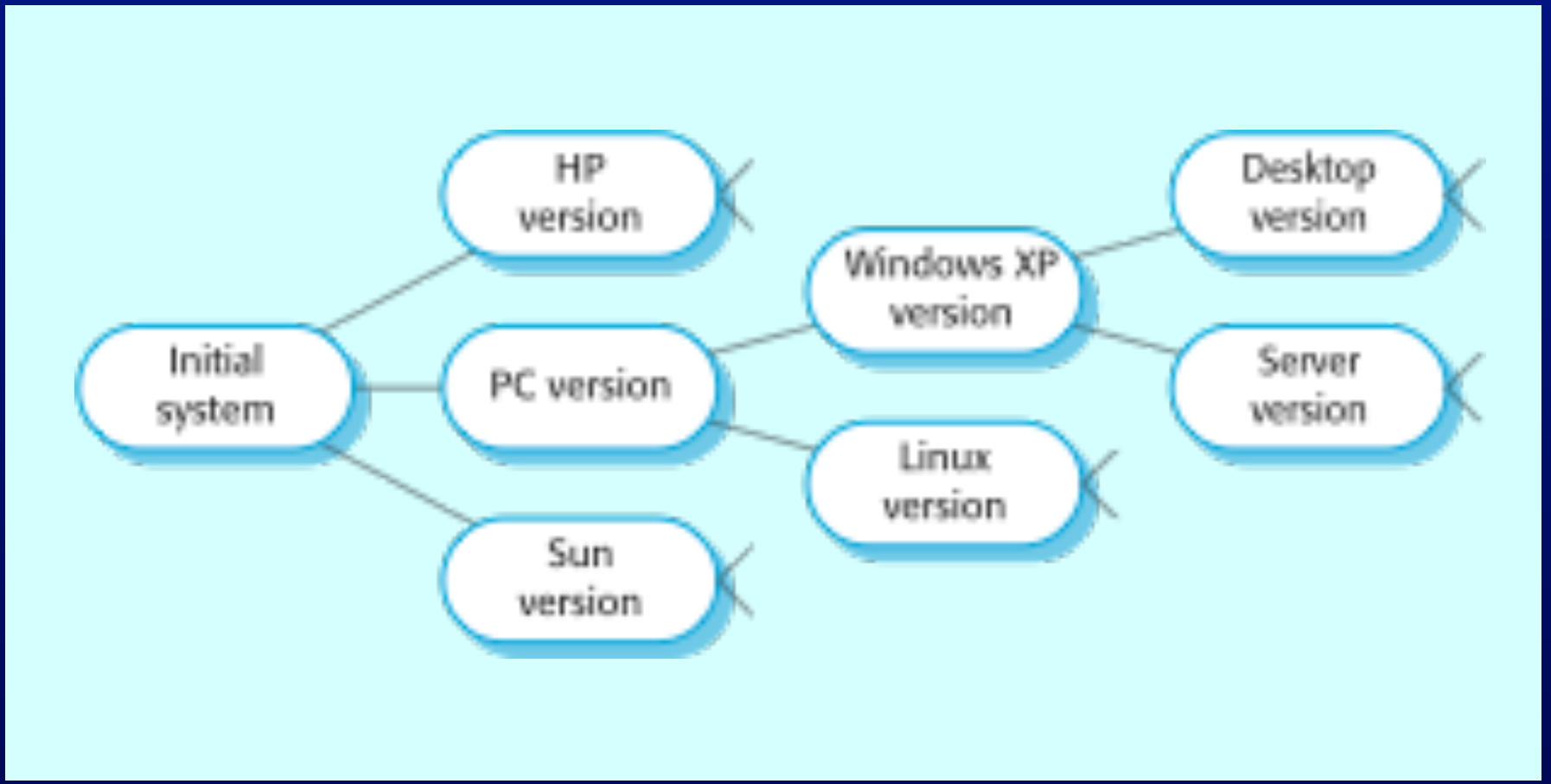
Configuration management

- New versions of software systems are created as they change:
  - For different machines/OS;
  - Offering different functionality;
  - Tailored for particular user requirements.
- Configuration management is concerned with managing evolving software systems:
  - System change is a team activity;
  - CM aims to control the costs and effort involved in making changes to a system.

Configuration management

- Involves the development and application of procedures and standards to manage an evolving software product.
- CM may be seen as part of a more general quality management process.
- When released to CM, software systems are sometimes called *baselines* as they are a starting point for further development.

System families



Configuration management planning

- All products of the software process may have to be managed:
  - Specifications;
  - Designs;
  - Programs;
  - Test data;
  - User manuals.
- Thousands of separate documents may be generated for a large, complex software system.

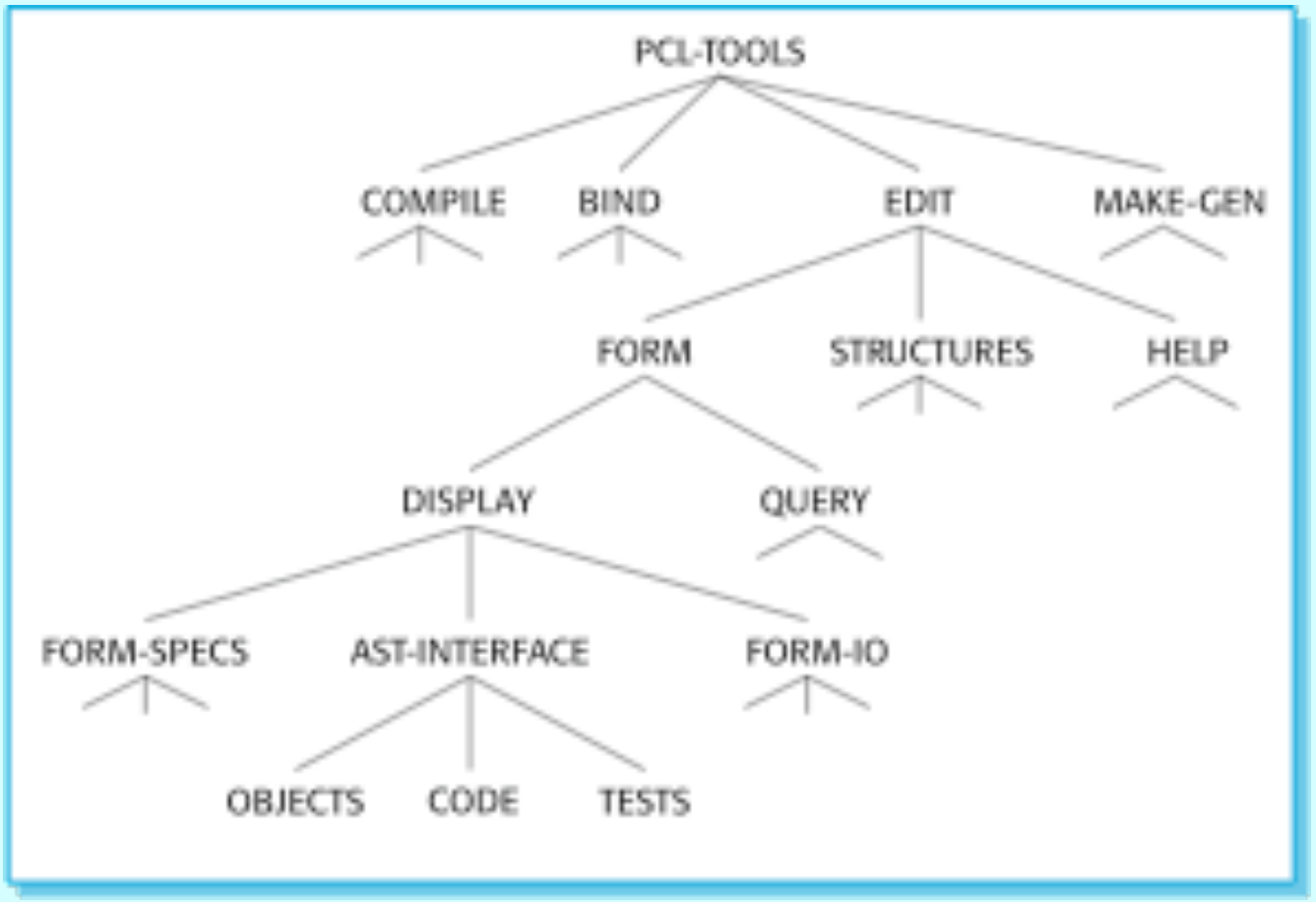
The CM plan

- Defines the types of documents to be managed and a document naming scheme.
- Defines who takes responsibility for the CM procedures and creation of baselines.
- Defines policies for change control and version management.
- Defines the CM records which must be maintained. Describes the tools which should be used to assist the CM process and any limitations on their use.
- Defines the process of tool use.
- Defines the CM database used to record configuration information.
- May include information such as the CM of external software, process auditing, etc.

Configuration item identification

- Large projects typically produce thousands of documents which must be uniquely identified.
- Some of these documents must be maintained for the lifetime of the software.
- Document naming scheme should be defined so that related documents have related names.
- A hierarchical scheme with multi-level names is probably the most flexible approach.
  - PCL-TOOLS/EDIT/FORMS/DISPLAY/AST-INTERFACE/CODE
- Use a Configuration Database
  - Linked with resources

Configuration hierarchy



Change management

- Software systems are subject to continual change requests:
  - From users;
  - From developers;
  - From market forces.
- Change management is concerned with keeping track of these changes and ensuring that they are implemented in the most cost-effective way.
- Major difficulty: tracking change status

Change control board

- Changes should be reviewed by an external group who decide whether or not they are cost-effective from a strategic and organizational viewpoint rather than a technical viewpoint.
- Should be independent of project responsible for system. The group is sometimes called a change control board.
- The CCB may include representatives from client and contractor staff.

Derivation history

- This is a record of changes applied to a document or code component.
- It should record, in outline, the change made, the rationale for the change, who made the change and when it was implemented.
- It may be included as a comment in code. If a standard prologue style is used for the derivation history, tools can process this automatically.

Component header information

```
// BANKSEC project (IST 6087)
//
// BANKSEC-TOOLS/AUTH/RBAC/USER_ROLE
//
// Object: currentRole
// Author: N. Perwaiz
// Creation date: 10th November 2002
//
// © Lancaster University 2002
//
// Modification history
// Version      Modifier     Date          Change        Reason
// 1.0          J. Jones     1/12/2002     Add header    Submitted to CM
// 1.1          N. Perwaiz   9/4/2003      New field     Change req. R07/02
```

Versions/variants/releases

- Version An instance of a system which is functionally distinct in some way from other system instances.
- Variant An instance of a system which is functionally identical but non-functionally distinct from other instances of a system.
- Release An instance of a system which is distributed to users outside of the development team.

Version identification

- Procedures for version identification should define an unambiguous way of identifying component versions.
- There are three basic techniques for component identification
  - Version numbering;
  - Attribute-based identification;
  - Change-oriented identification.

Release management

- Releases must incorporate changes forced on the system by errors discovered by users and by hardware changes.
- They must also incorporate new system functionality.
- Release planning is concerned with when to issue a system version as a release.

System releases

- Not just a set of executable programs.
- May also include:
  - Configuration files defining how the release is configured for a particular installation;
  - Data files needed for system operation;
  - An installation program or shell script to install the system on target hardware;
  - Electronic and paper documentation;
  - Packaging and associated publicity.
- Systems are now normally released on optical disks (CD or DVD) or as downloadable installation files from the web.

Release problems

- Customer may not want a new release of the system
  - They may be happy with their current system as the new version may provide unwanted functionality.
- Release management should not assume that all previous releases have been accepted. All files required for a release should be re-created when a new release is installed.

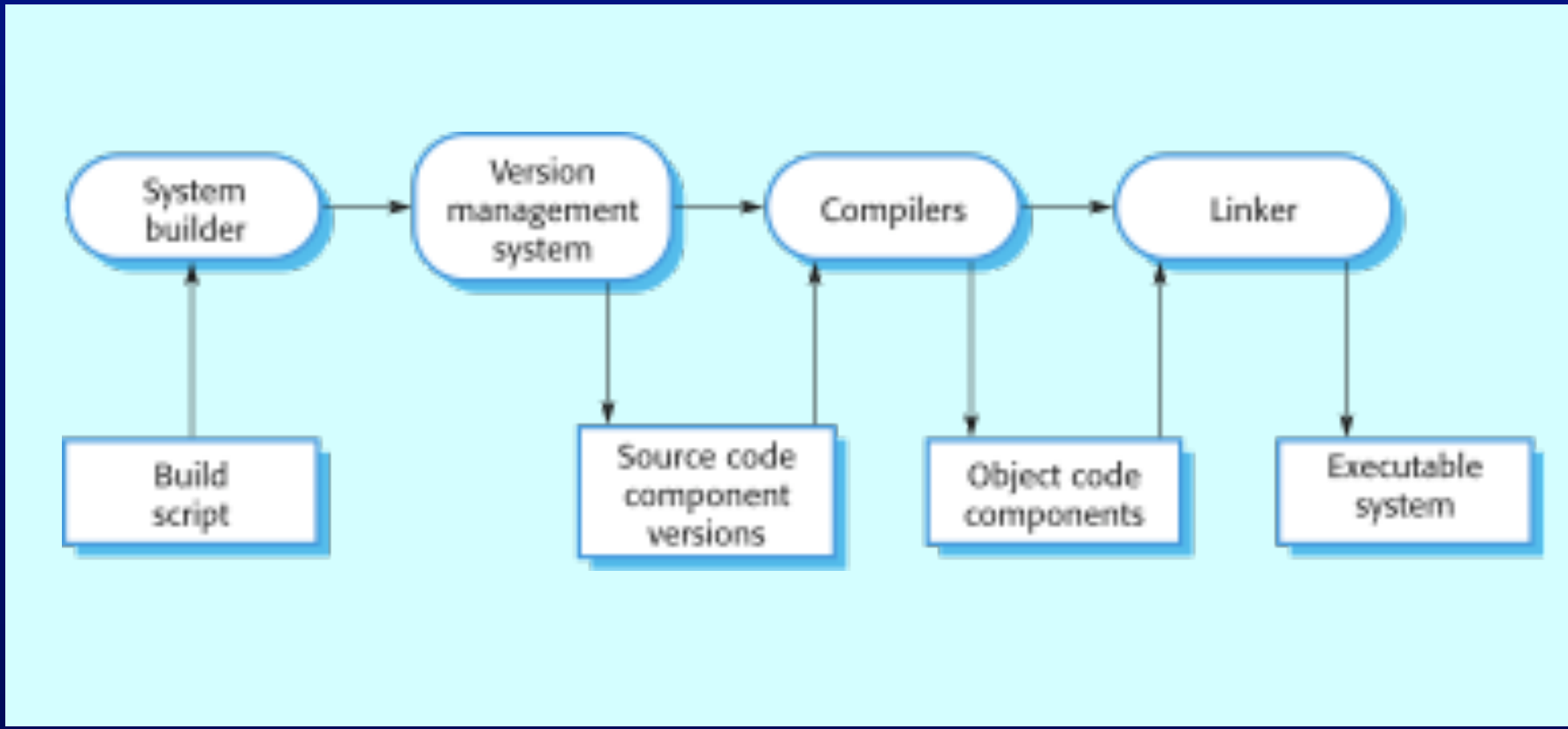
System release strategy

Factor	Description
Technical quality of the system	If serious system faults are reported which affect the way in which many customers use the system, it may be necessary to issue a fault repair release. However, minor system faults may be repaired by issuing patches (often distributed over the Internet) that can be applied to the current release of the system.
Platform changes	You may have to create a new release of a software application when a new version of the operating system platform is released.
Lehman's fifth law (see Chapter 21)	This suggests that the increment of functionality that is included in each release is approximately constant. Therefore, if there has been a system release with significant new functionality, then it may have to be followed by a repair release.
Competition	A new system release may be necessary because a competing product is available.
Marketing requirements	The marketing department of an organisation may have made a commitment for releases to be available at a particular date.
Customer change proposals	For customised systems, customers may have made and paid for a specific set of system change proposals and they expect a system release as soon as these have been implemented.

System building

- The process of compiling and linking software components into an executable system.
- Different systems are built from different combinations of components.
- This process is now always supported by automated tools that are driven by ‘build scripts’.

System building



CASE tools for configuration management

- CM processes are standardised and involve applying pre-defined procedures.
- Large amounts of data must be managed.
- CASE tool support for CM is therefore essential.
- Mature CASE tools to support configuration management are available ranging from stand-alone tools to integrated CM workbenches.

CASE Tools

- Change management tools
  - Form editor; Workflow system (who does what)
  - Change database; Change reporting system (generates management reports)
- Version Management Tools
  - Version and release identification
  - Storage management.
    - stores the differences between versions; Change history recording; Record reasons for version creation.
  - Independent development
  - Project support
    - manage groups of files rather than just single files.
- System building tools may provide
  - A dependency specification language and interpreter;
  - Distributed compilation;
  - Derived object management.