Verification and Validation

- Assuring that a software system meets a user's needs

Objectives

- To introduce software verification and validation and to discuss the distinction between them
- To describe the program inspection process and its role in V & V
- To explain static analysis as a verification technique
- To describe the Cleanroom software development process

Topics covered

- Verification and validation planning
- Software inspections
- Automated static analysis
- Cleanroom software development

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

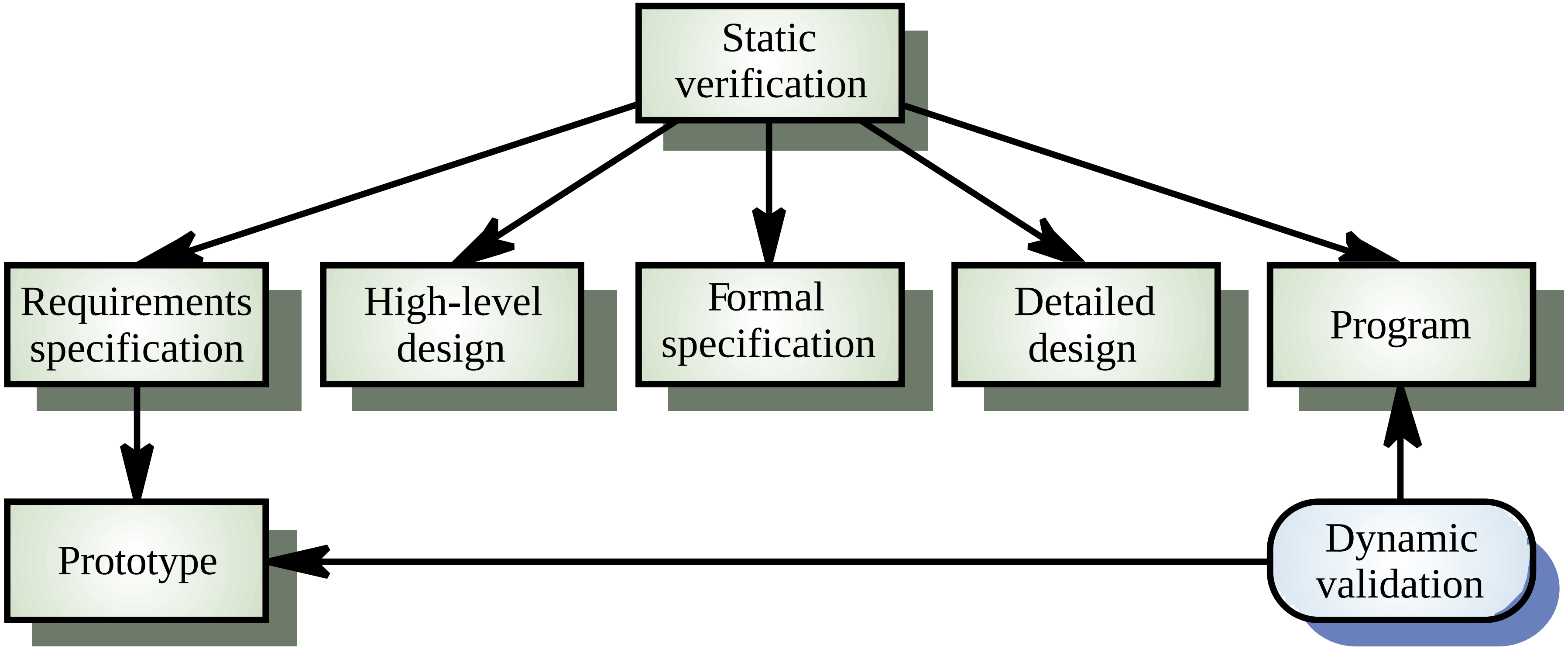
The V & V process

- 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 usable in an operational situation.

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.
  - Covered in Chapter 20
- Statistical testing
  - tests designed to reflect the frequence of user inputs. Used for reliability estimation.
  - Covered in Chapter 21

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

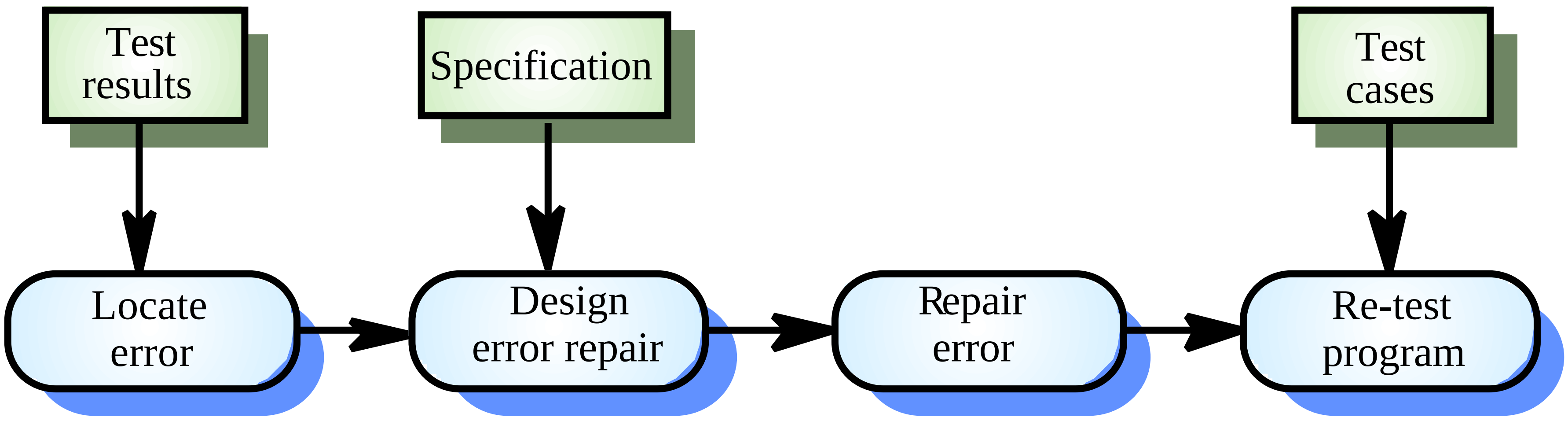
V & V confidence

- 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

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

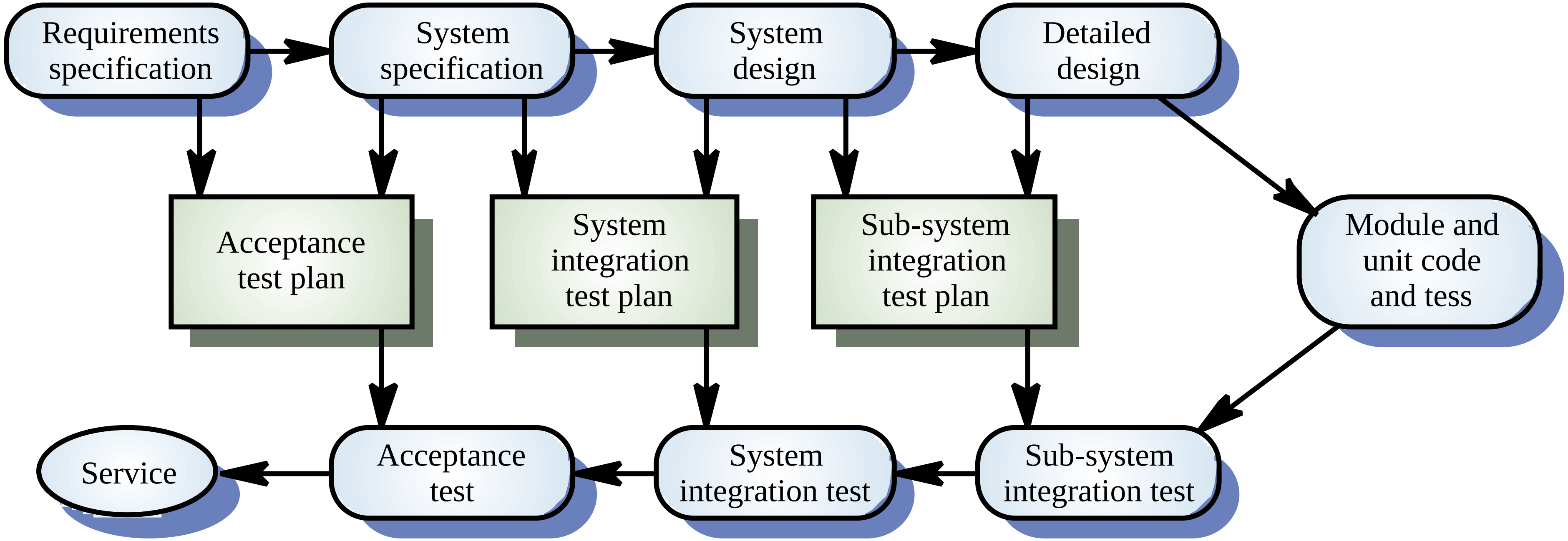
The debugging process



V & 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 V-model of development



The structure of a software test plan

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

Software inspections

- Involve people examining the source representation with the aim of discovering anomalies and defects
- Do not require execution of a system so may be used before implementation
- May be applied to any representation of the system (requirements, design, test data, etc.)
- Very effective technique for discovering errors

Inspection success

- Many diffreent 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.

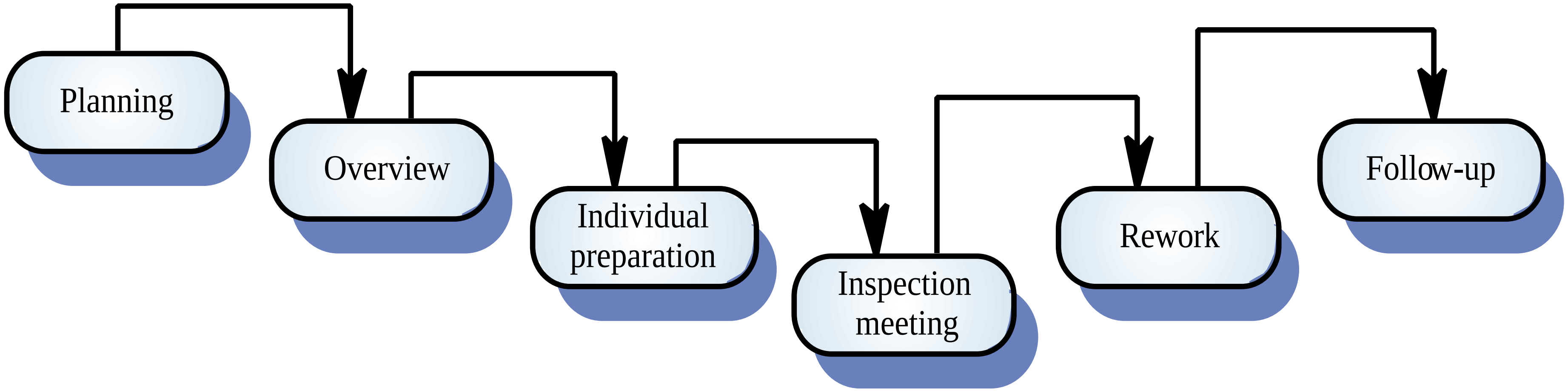
Program inspections

- 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 must be available
- An error checklist should be prepared
- Management must accept that inspection will increase costs early in the software process
- Management must not use inspections for staff appraisal

The inspection process



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

Inspection teams

- Made up of at least 4 members
- Author of the code being inspected
- Inspector who finds errors, omissions and inconsistencies
- Reader who reads the code to the team
- Moderator who chairs the meeting and notes discovered errors
- Other roles are Scribe and Chief moderator

Inspection checklists

- Checklist of common errors should be used to drive the inspection
- Error checklist is programming language dependent
- The 'weaker' the type checking, the larger the checklist
- Examples: Initialisation, Constant naming, loop termination, array bounds, etc.

Fault class	Inspection check
Data faults	Are all program variables initialised before their values are used? Have all constants been named? Should the lower bound of arrays be 0, 1, or something else? 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?
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?
Input/output faults	Are all input variables used? Are all output variables assigned a value before they are output?
Interface faults	Do all function and procedure 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?

Inspection checks

Inspection rate

- 500 statements/hour during overview
- 125 source statement/hour during individual preparation
- 90-125 statements/hour can be inspected
- Inspection is therefore an expensive process
- Inspecting 500 lines costs about 40 man/hours effort = £2800

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
- Very effective as an aid to inspections. A supplement to but not a replacement for inspections

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

Stages of static analysis

- *Control flow analysis.* Checks for loops with multiple exit or entry points, finds unreachable code, etc.
- *Data use analysis.* Detects uninitialised 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

Stages of static analysis

- *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 these stages generate vast amounts of information. Must be used with care.

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
```

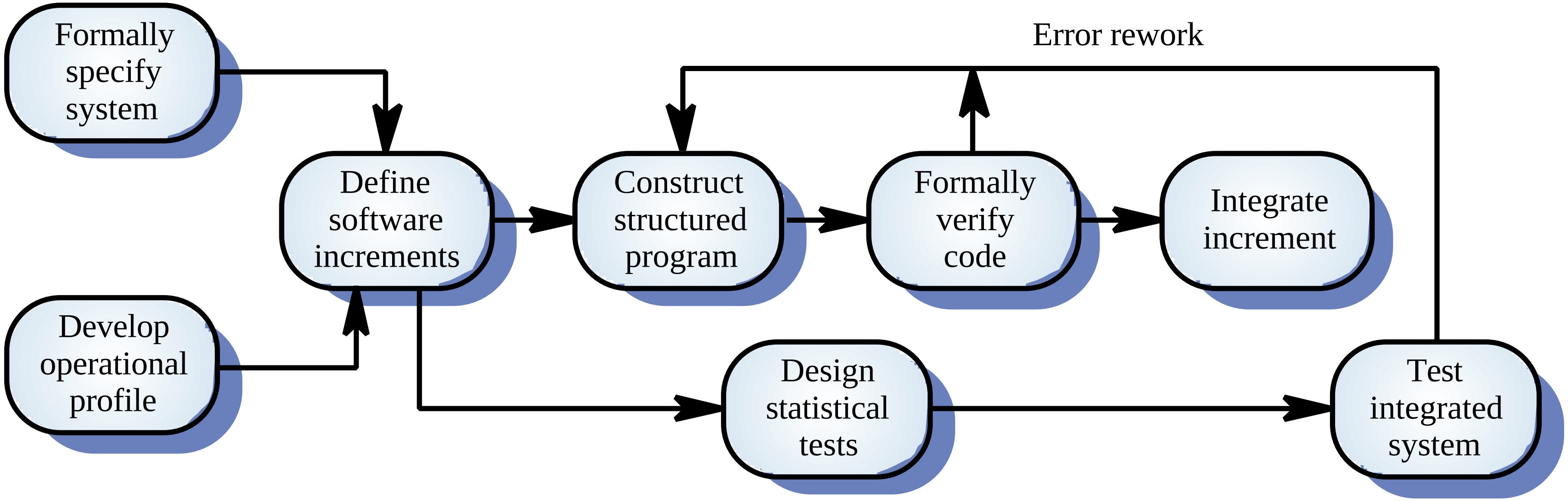
Use of static analysis

- 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

Cleanroom software development

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

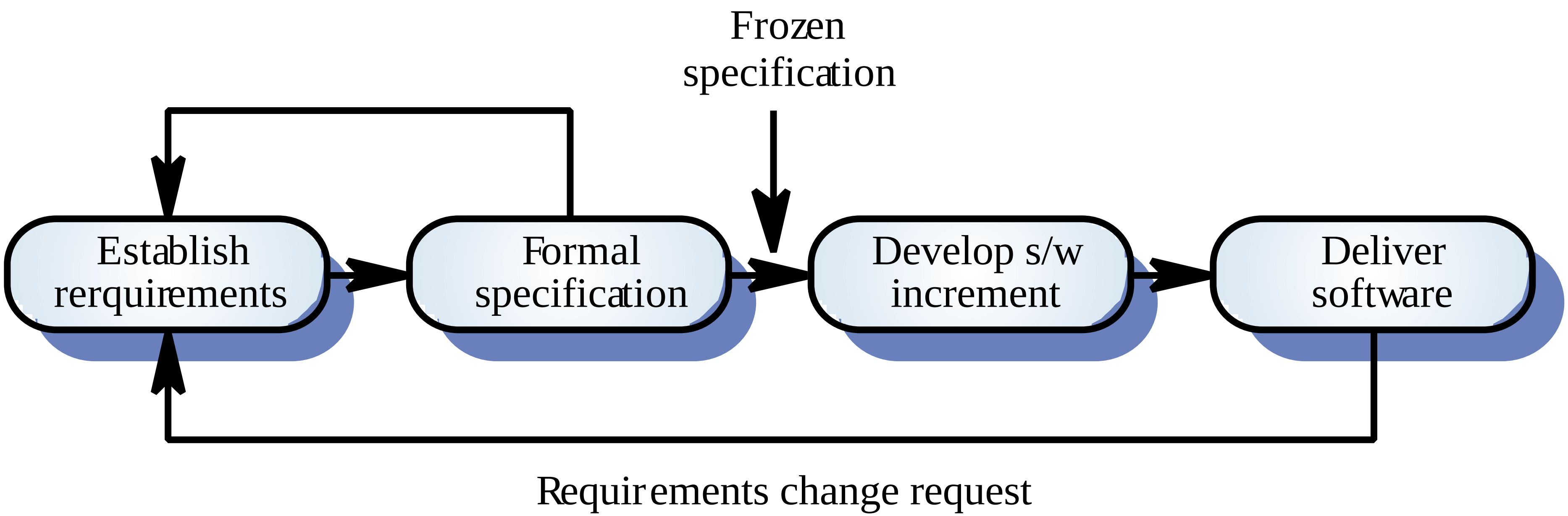
The Cleanroom process



Cleanroom process characteristics

- Formal specification using a state transition model
- Incremental development
- Structured programming - limited control and abstraction constructs are used
- Static verification using rigorous inspections
- Statistical testing of the system (covered in Ch. 21).

Incremental development



Formal specification and inspections

- The state based model is a system specification and the inspection process checks the program against this model
- Programming approach is defined so that the correspondence between the model and the system is clear
- Mathematical arguments (not proofs) are used to increase confidence in the inspection process

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

Cleanroom process evaluation

- Results in IBM have been very impressive with few discovered faults in delivered systems
- Independent assessment shows that the process is no more expensive than other approaches
- Fewer errors than in a 'traditional' development process
- Not clear how this approach can be transferred to an environment with less skilled or less highly motivated engineers

Key points

- 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

Key points

- Program inspections are very effective in discovering errors
- Program code in inspections is 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
- The Cleanroom development process depends on incremental development, static verification and statistical testing