



# Lecture 14 – Verification & Validation 2 (Sommerville Ch. 19)

Karl R. Wilcox  
K.R.Wilcox@reading.ac.uk



# Objectives

- To explain static analysis as a verification technique
- To describe the Cleanroom software development process



# 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.**



```
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**



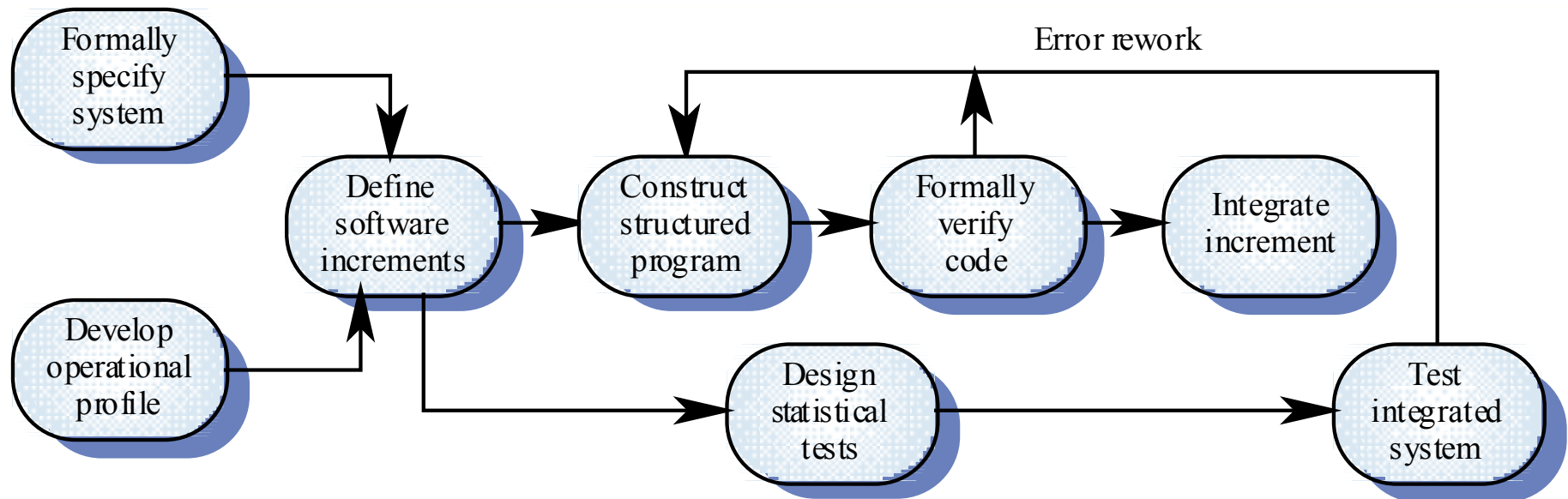


# Cleanroom software development

- 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



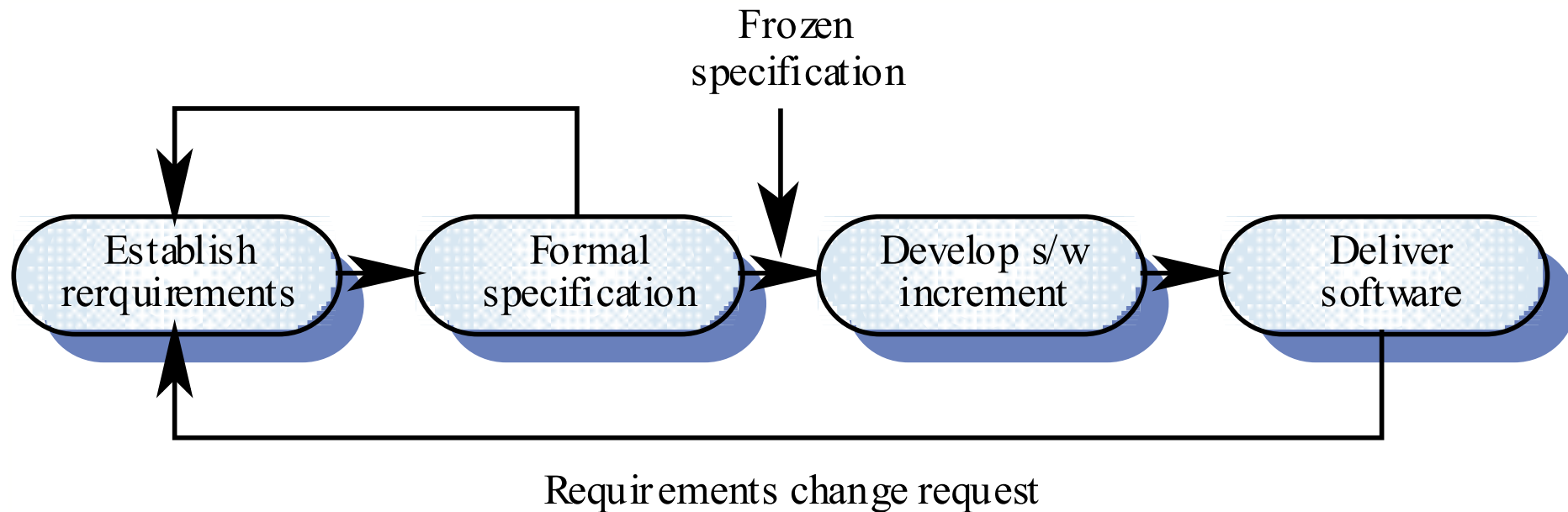


# Cleanroom process characteristics

- **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

- **Static verification techniques involve examination and analysis of the program for error detection**
- **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**