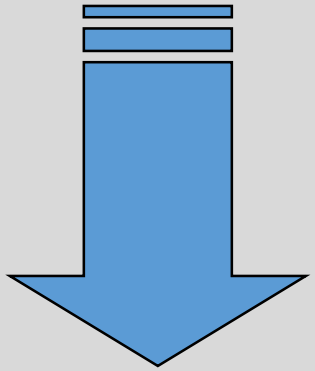
COMP201 – Software Engineering I
Lecture 24 – Validation and Verification

Lecturer: Dr T Carroll

Email: Thomas.Carroll2@Liverpool.ac.uk
Office: G.14
See Vital for all notes

Verification and Validation



Ensuring that a software system meets a user's needs

Objectives

- To introduce software verification and validation
  - 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

Verification vs Validation

- **Verification**:
  "Are we building the product correctly?"
- The software should conform to its specification

- **Validation**:
  "Are we building the right product"
- The software should do what the user really requires

Verification vs Validation

- Verification should check the program meets its specification as written in the requirements document.
  - This may involve checking that it meets it functional and non-functional requirements
- Validation ensures that the product meets the customers expectations
  - System specifications don’t always accurately reflect the real needs of users

The Verification & Validation Process

- As 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.

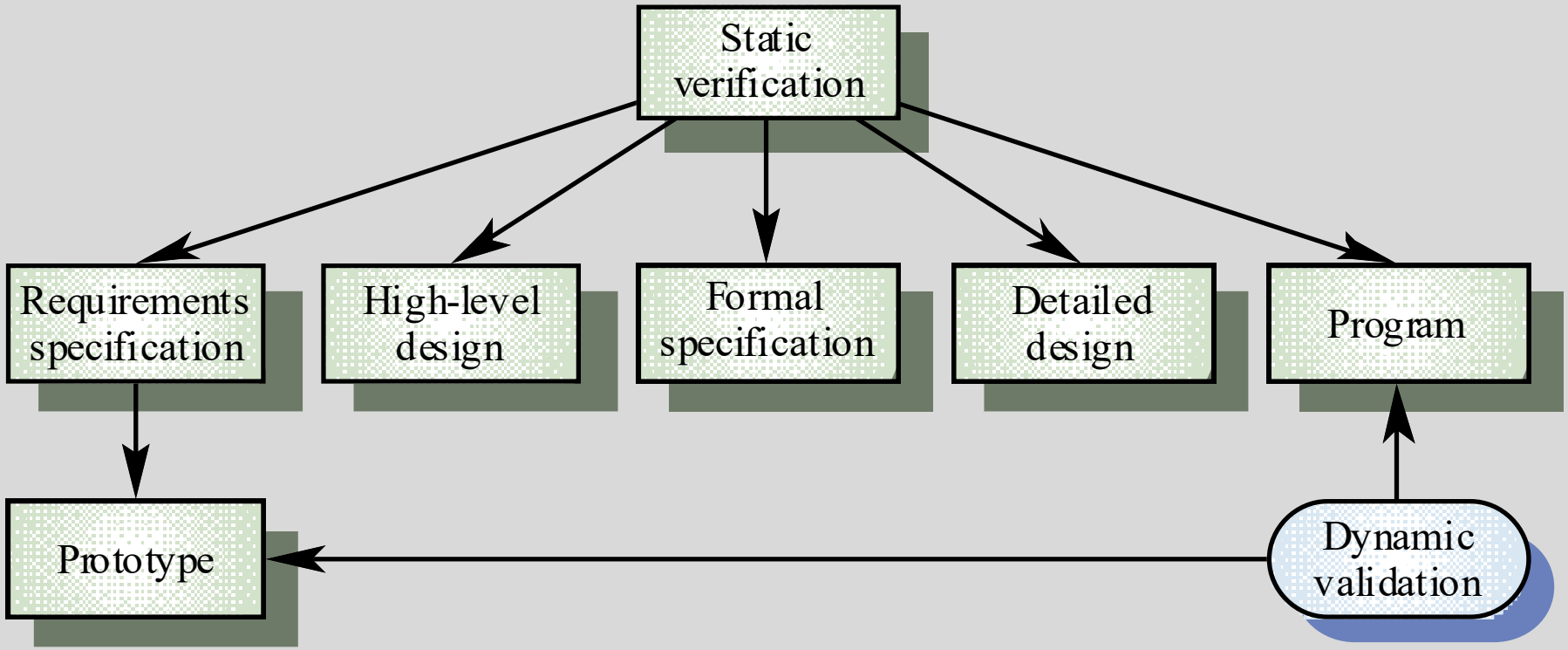
Static and Dynamic Verification

- ***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 Verification

- **System testing** is only possible when an executable version of the program is available
- This is therefore an advantage of incremental development
  - a testable version of the system is available at a fairly early stage
- New functionality can be checked as it is added
  - we can perform **regression testing**
- Real data can be used as input to the system and we try to observe any anomalies in the output

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
- Program testing is the only validation technique for non-functional requirements
- Should be used in conjunction with static verification to provide full V&V coverage

Testing with Agile (XP and SCRUM)

- Development is test driven
- Test developed before target code
- Target code developed to pass test
- Benefits
  - Test is more valid, based purely on specification
  - Code is always tested
  - Code is often simpler and closer to specification
- TDD: see comp220 comp285 for more details…

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.

- **Statistical testing**
  - Tests designed to reflect the frequency of user inputs.
  - Used for reliability estimation.

Verification & Validation Goals

Verification and validation should establish a degree of confidence that the software is fit for purpose

- **This does NOT mean completely free of defects**
- The degree of confidence required depends upon:
  - Software function – safety critical systems need *higher confidence* than prototype systems
  - User expectations – Users sometimes have a low expectation of software and are willing to tolerate some system failures (although this is decreasing)
  - Marketing environment – Competing programs must be taken into account and the required schedule for introducing the product to market.
  - Cheaper products may be expected to have more faults.

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 the defects
- Debugging involves
  - formulating a hypothesis about program behaviour
  - then testing these hypotheses to find the system error

Testing and Debugging

- There is no simple process for debugging
- It often involves **looking for patterns** in test outputs with defects and using a **programmers skill** to locate the error
- Question: Recall the programs you have written so far.
  - Were there errors in early versions?
  - How did you discover them and fix them?
  - Were they syntactic or semantic errors?
- Interactive debuggers provide a special run-time environment with access to the symbol table and program variables to aid error location.
  - You can also “step-through” the program line by line

Example Incorrect Code

```
public class Temperature {
 // calcTGrd function to calc. the value of a T gradient
 public double calcTGrd(float ZVAL) {
     int a = (int) x * x
     if(a = 1)
          x = ZVAL * 3.8883;
     return a;
 }
 public double x;
}
```

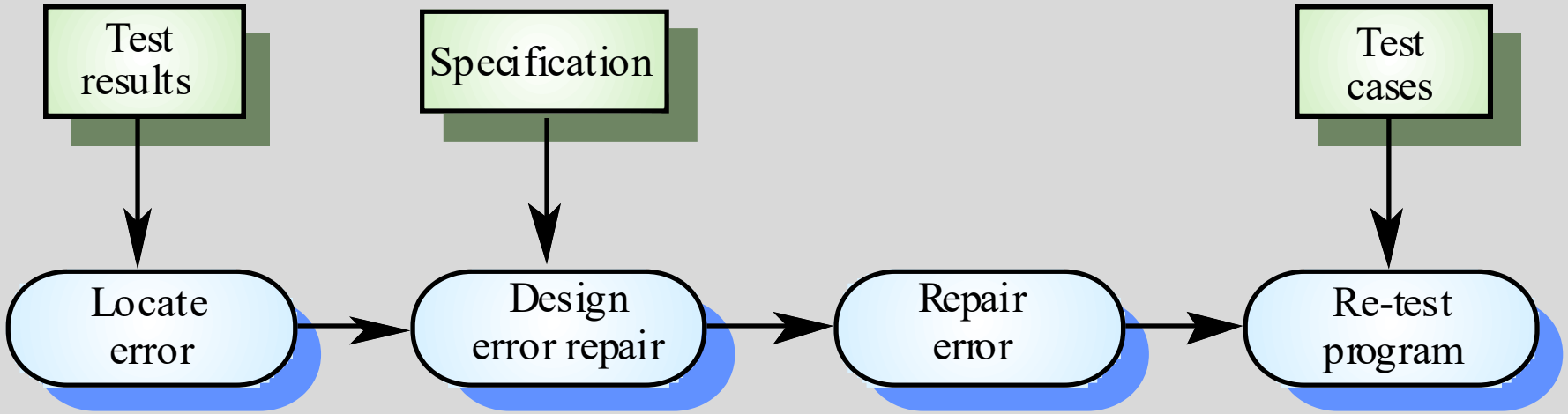
Syntax and Semantic Errors

- A syntax error should be caught by the compiler
  - indicate the **location** and the **type** of error.
- A semantic error (also called a logical error) can occur in a program which compiles and runs, but produces incorrect output on some (or all) input
  - (e.g. An incorrect algorithm or mistake in a formulae etc.)
- Semantic errors are often harder to detect since the compiler may not be able to indicate where/what the problem is.

Testing and Debugging

- Once errors are located, it is necessary to:
  - correct the program code
  - re-test the program
- **Regression testing** – after fixing a defect, it is advisable to retest the program with all previous test data
  - This tries to ensure the "fix" has not introduced new problems
  - This is not always feasible due to costs
- Experience has shown that a large proportion of fault repairs introduce **new errors** or are **incomplete**

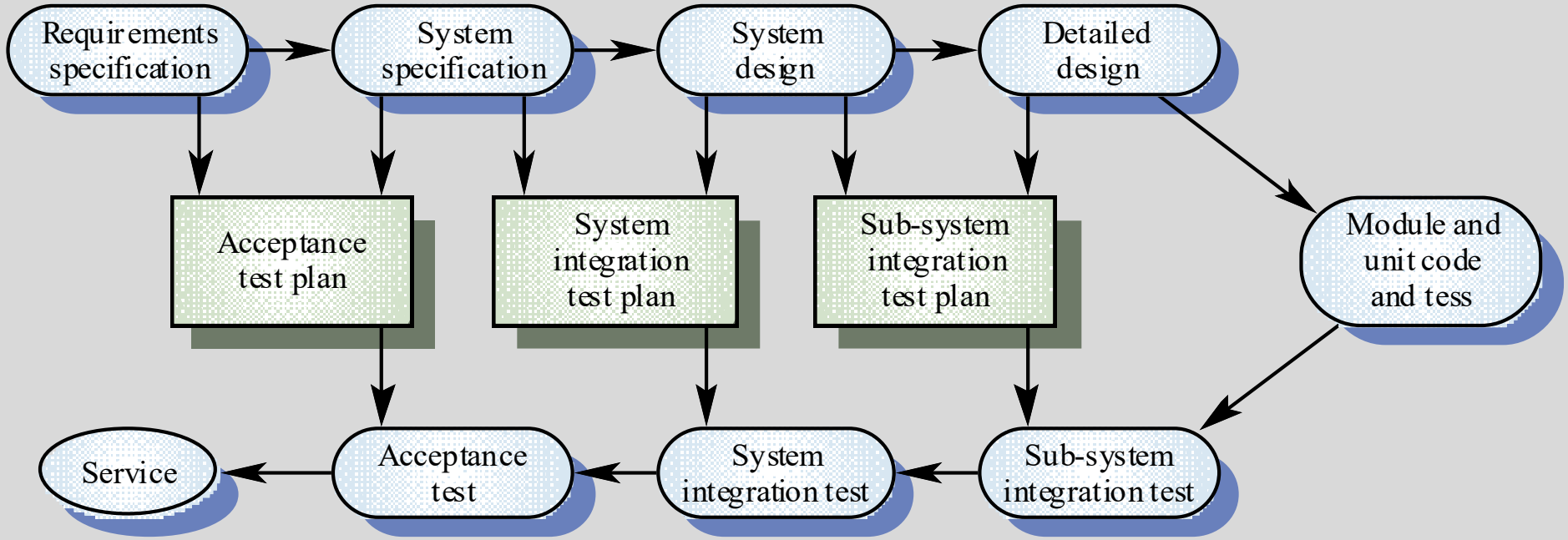
The Debugging Process



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



This diagram shows how test plans should be derived from the system specification and design.

The Structure of a Software Test Plan

- The testing process - a description of the major phases of the testing process
- Requirements traceability – testing should ensure that all requirements are individually tested
- Tested items – Specify the products of the software process to be tested
- Testing schedule – An overall schedule for the testing of the software is required
  - resources (time and personnel) must be allocated as part of the general project schedule

The Structure of a Software Test Plan

- Test recording procedures – The results of tests must be systematically recorded
  - it is not enough to simply run the tests.
  - This allows an audit of the testing process
- Hardware and software requirements –A list of software tools required and the estimated hardware utilisation
- Constraints – Any constraints affecting the testing process should be anticipated in this section

Software Inspections

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

Software Inspections

- Incomplete versions of the system can be inspected without additional costs
  - specialised test harnesses that work on only a part of the program are not required
- As well as program defects, inspections can consider broader quality attributes
  - compliance with standards
  - portability and maintainability
- Poor programming style and inefficiencies can be found and corrected
  - make the system easier to maintain and update

Inspection Success

- Many different defects may be discovered in a single inspection
- There is no “interaction” between errors to be concerned with
  - In testing, one defect, may mask another, so several executions would be required
- Reviewers 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
- Inspections can't check:
  - conformance with the customer's real requirements
  - non-functional characteristics
    - performance, usability, etc

Lecture 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.
- **Program inspections** are very effective in discovering errors
- Different types of systems and software development processes require different levels of verification and validation