Software validation: testing I

- Basic terminology
- Phases in the testing process

CE202 Software Engineering, Autumn term
Dr Michael Gardner, School of Computer Science and Electronic Engineering, University of Essex

Software Testing

- Who tests?
  - Ideally specialist test teams with access to software to build and execute test scripts
  - Often the analysts who have carried out the initial requirements gathering and analysis
  - In eXtreme Programming (XP) programmers are expected to write test harnesses for classes before they write the code
  - Users of the system, who will test against requirements and do user acceptance testing

Basic testing terminology

- **Testing**: Executing a program, to detect the differences between specified and observed behaviour.
- **Validation vs. Defect Testing:**
  - **Validation testing**: The goal is to ensure the system meets the client's expectations
  - **Defect testing**: The goal is to detect defects (reveal problems).
- **Black vs. White Box Testing:**
  - **Black-box**: From "outside", just the public methods
  - **White-box**: From "inside", considering all aspects (including private methods and fields)

Examples of Faults and Errors

- Faults in the Interface specification
  - Mismatch between what the client needs and what the server offers
  - Mismatch between requirements and implementation
- Algorithmic Faults
  - Missing initialization
  - Incorrect branching condition
  - Missing test for null

- Mechanical Faults (very hard to find)
  - Operating temperature outside of equipment specification
- Errors
  - Null reference errors
  - Concurrency errors
  - Exceptions.

Purpose of Testing

- The purpose of testing is to try find errors, **not to prove the software is correct**
  - Test data should test the software at its limits and test business rules
    - extreme values (very large numbers, long strings)
    - borderline values (0, -1, 0.999)
    - invalid combinations of values (age = 3, marital status = married)
    - nonsensical values (negative order line quantities)
    - heavy loads (are performance requirements met?)
    - See week 10 class on selecting test conditions (equivalence classes) and choosing boundary values

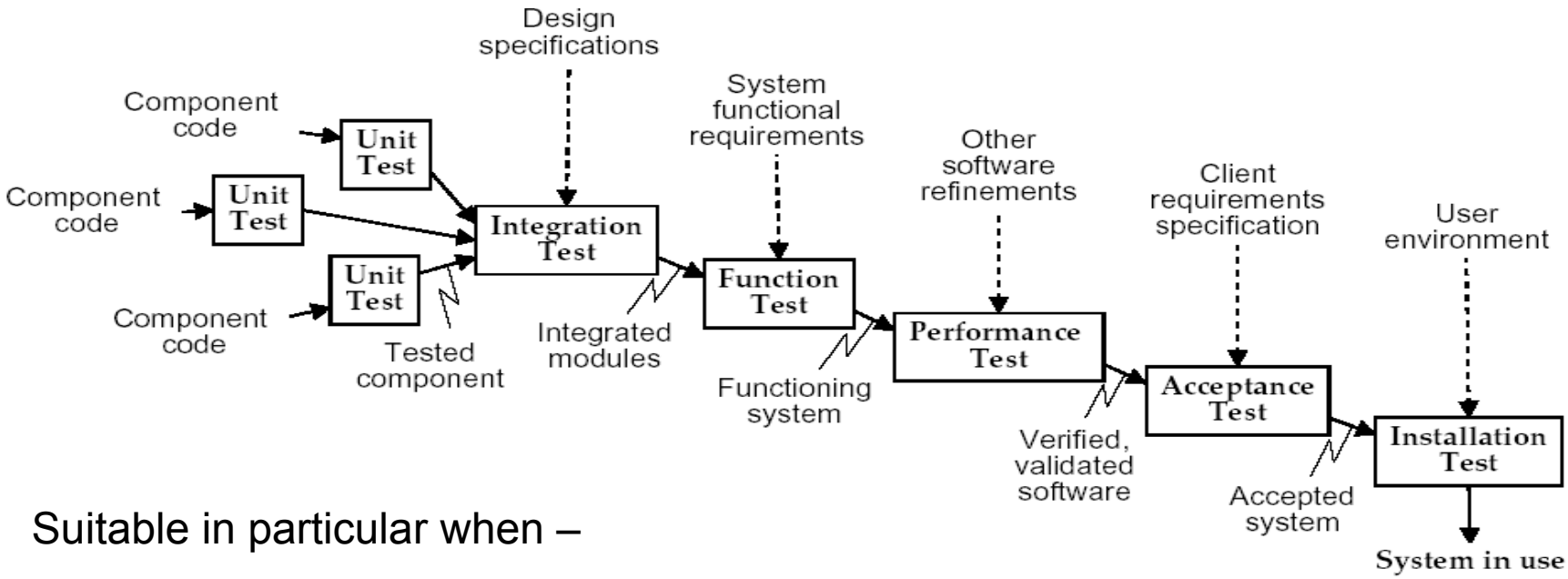
Caveat

- Testing can show the presence of bugs *but not their absence* [Dijkstra, 1972]

Levels of Testing

- Level 1
  - Test modules (classes), then programs (use cases) then suites (application)
- Level 2 (Alpha Testing or Verification)
  - Execute programs in a simulated environment and test inputs and outputs
- Level 3 (Beta Testing or Validation)
  - Test in a live user environment and test for response times, performance under load and recovery from failure

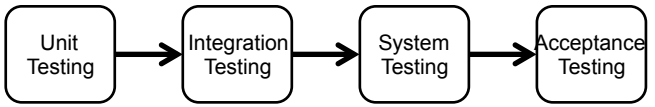
The testing hierarchy

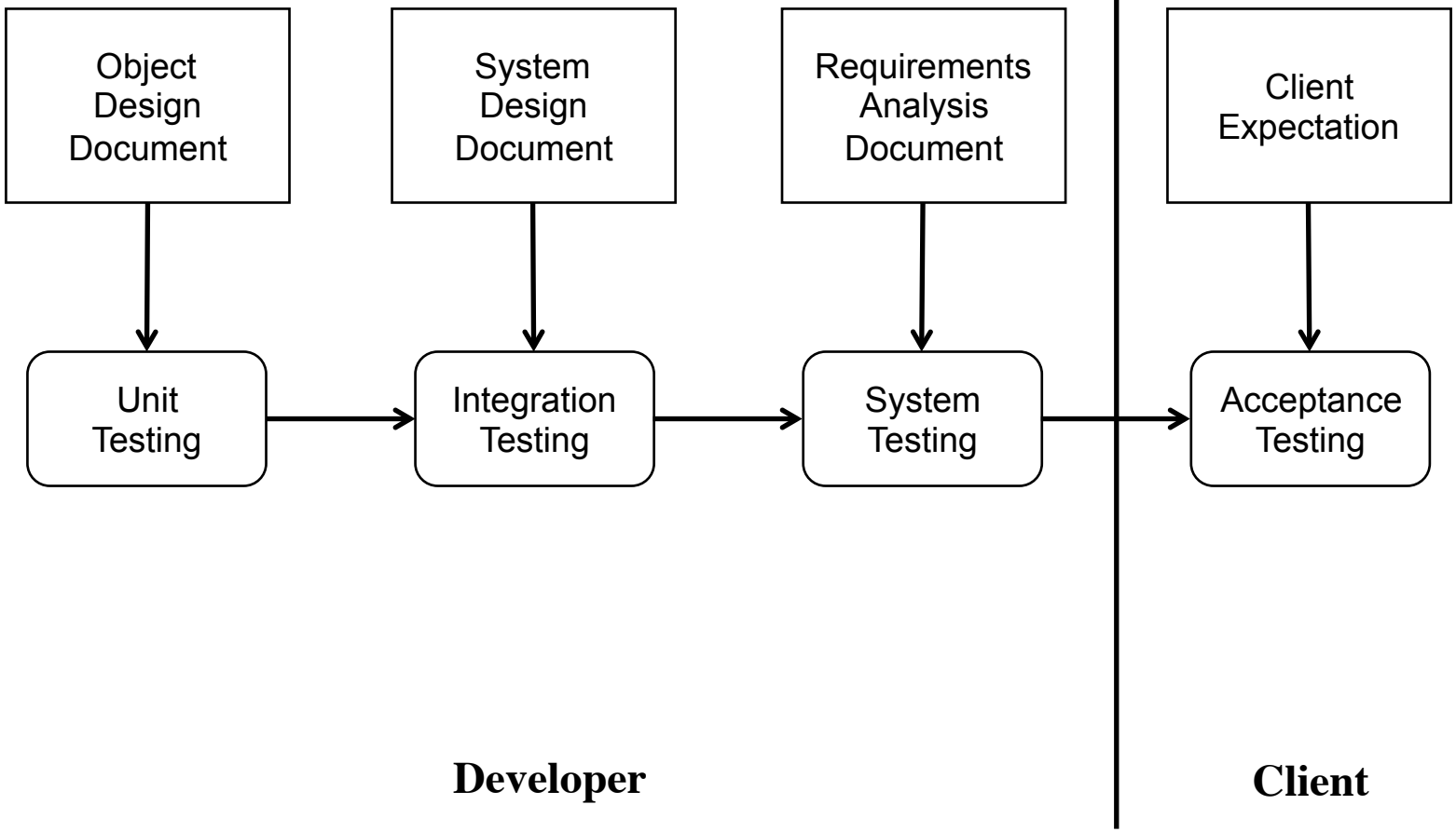


Suitable in particular when –

1. Project is large enough
2. Process model has progressively larger deliverables

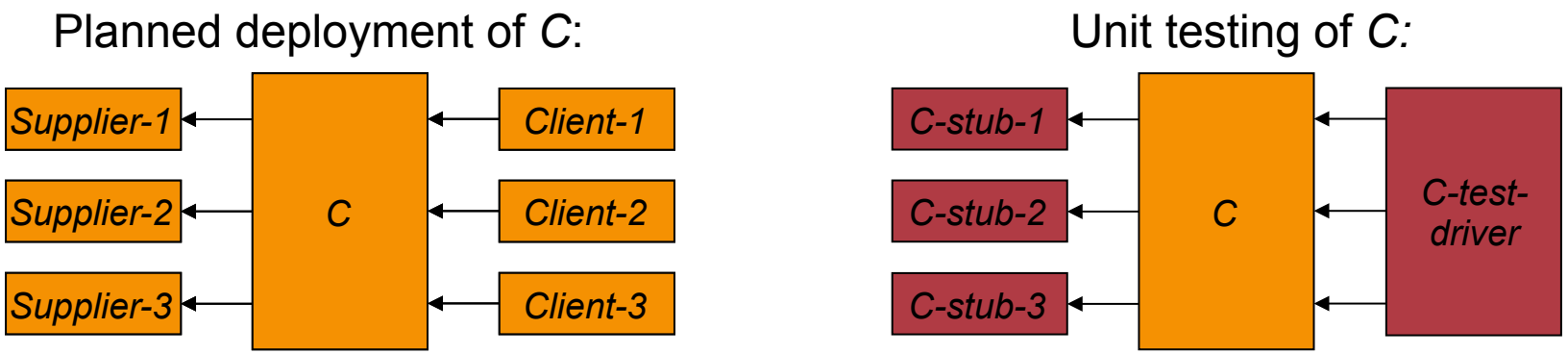
Testing Activities





Unit testing

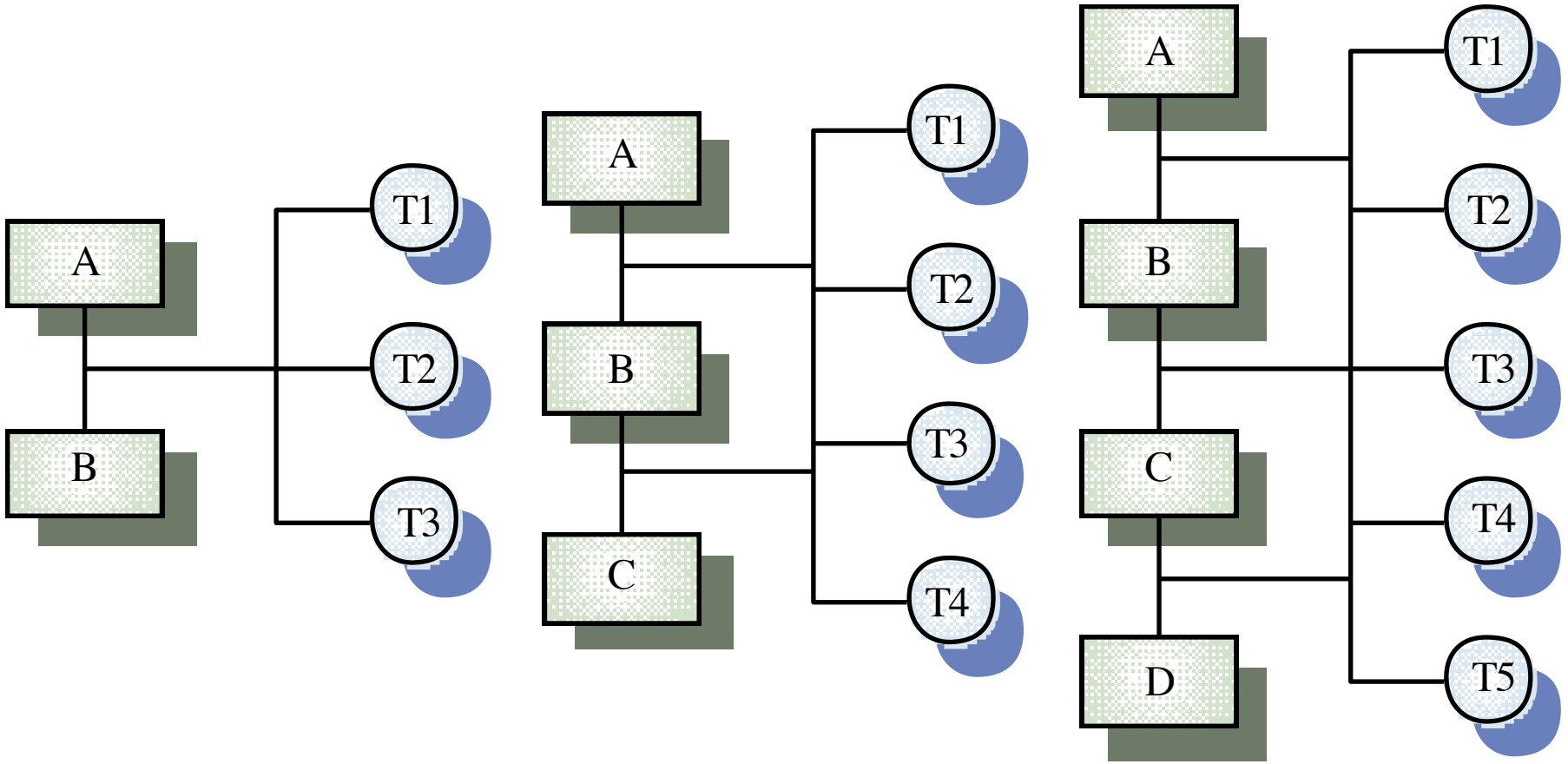
- Isolate the component under test *C* from the rest of the program:
  - *Test driver* calls *C*
  - *Test stubs* whenever *C* calls other components



Integration testing

- Tests a system consisting of integrated components
  - Can be any set of packages or components, up to the entire program
- Should be black-box testing
  - Tests derived from the specification
- **Problem**: Difficult to establish where the fault occurred
  - Not the problem with Unit testing
- **Solution**: Incremental integration testing

Incremental integration testing



Test sequence 1

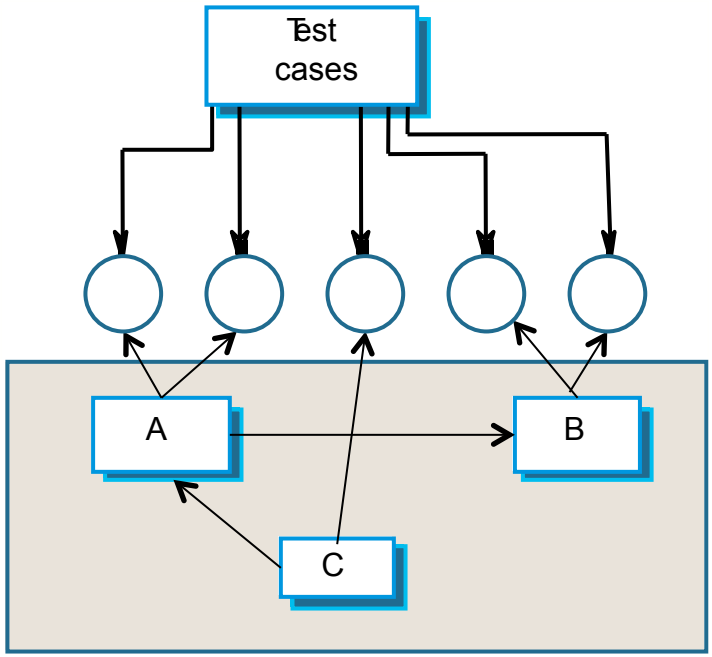
Test sequence 2

Test sequence 3

Approaches to integration testing

- Top-down testing
  - Start with high-level system and integrate from the top-down replacing individual components by stubs where appropriate
- Bottom-up testing
  - Integrate individual components in levels until the complete system is created
- In practice, most integration involves a combination of these strategies

Functional/System testing



- Conducted against the functional specifications
- Test function, not structure or performance
  - Test through interface
  - Ignore implementation detail
- Examples:
  - Test that a functional requirement is implemented properly

Performance testing

- Testing performance requirements
  - Usually: Test the programs use of resources such as time + space
- In contrast with: Functional or black-box testing
- Very important phase for real-time and critical systems!
- Test against the non-functional requirements

Example: Stress testing

- Exercises the system beyond its maximum design load
  - Often causes defects to come to light
- Testing the system's recovery
  - Systems should not fail catastrophically
    - Must not crash
    - Must not corrupt the data (e.g., leave 'dangling pointers')
    - Recovery should be swift

Regression testing

- Re-run all tests after applying changes
  - Tests lead to the discovery of faults
  - After changing the program: Must be re-tested!
- Applicable for every "version" of the program

Acceptance testing

- One step before last
  - Test against client's requirements
  - Involve the client in the testing
  - Use a test facility (not the client's environment)

Installation (a.k.a. release) testing

- Last step in testing
  - Execute in the target environment
  - Configure the target environment
  - Re-run all previous tests (unit, acceptance, etc.)
- usually black-box or functional testing
  - Based on the system specification only;
  - Ideally the testers do not have knowledge of the system implementation.

Summary

- Testing terminology eg. black/white box, validation & defect testing
- Purpose of testing – can't prove there are no bugs
- Levels of testing – classes/modules/applications, alpha & beta testing
- The testing hierachy
  - Unit testing
  - Integration/regression testing
  - Functional testing
  - Performance/stress testing
  - Acceptance testing
  - Installation testing

Further reading

- 1972 Turing Award lecture The Humble Programmer by Edsger Dijkstra
- Chapter 11 – Bruegge – Testing
- Chapter 8 – Pfleeger – Testing the Programs
- Chapter 9 – Pfleeger – Testing the System