COMP201
Software Engineering 1

Lecture 3 – Software Processes

Lecturer: **T. Carroll**

Ashton Building, Room G.14

E-mail: ***Thomas.Carroll2@liverpool.ac.uk***

See Vital for all notes

Recap

Recap From Lecture 2 – Software Process Models

- **The Waterfall Model**
  - Separate and distinct phases of specification and development
- **Evolutionary Development**
  - Specification and development are interleaved
- **Formal Systems Development**
  - A mathematical system model is formally transformed to an implementation
- **Iterative development (most widely used)**
  - The system is built up in a series of steps

This Lecture…

Outline

- A look at the Reuse-Oriented Model
- More in-depth look at the **Software Design Process**

Reuse-Oriented Model

Why Reinvent The Wheel?

- The Reuse-oriented model of software development incorporates the **reuse of existing code**, incorporating it into your development process
- Examples of code reuse:
  - User Authentication
  - Encryption
  - GUI Elements
  - Libraries
  - Copy/paste from the web

Code Reuse-Oriented Model

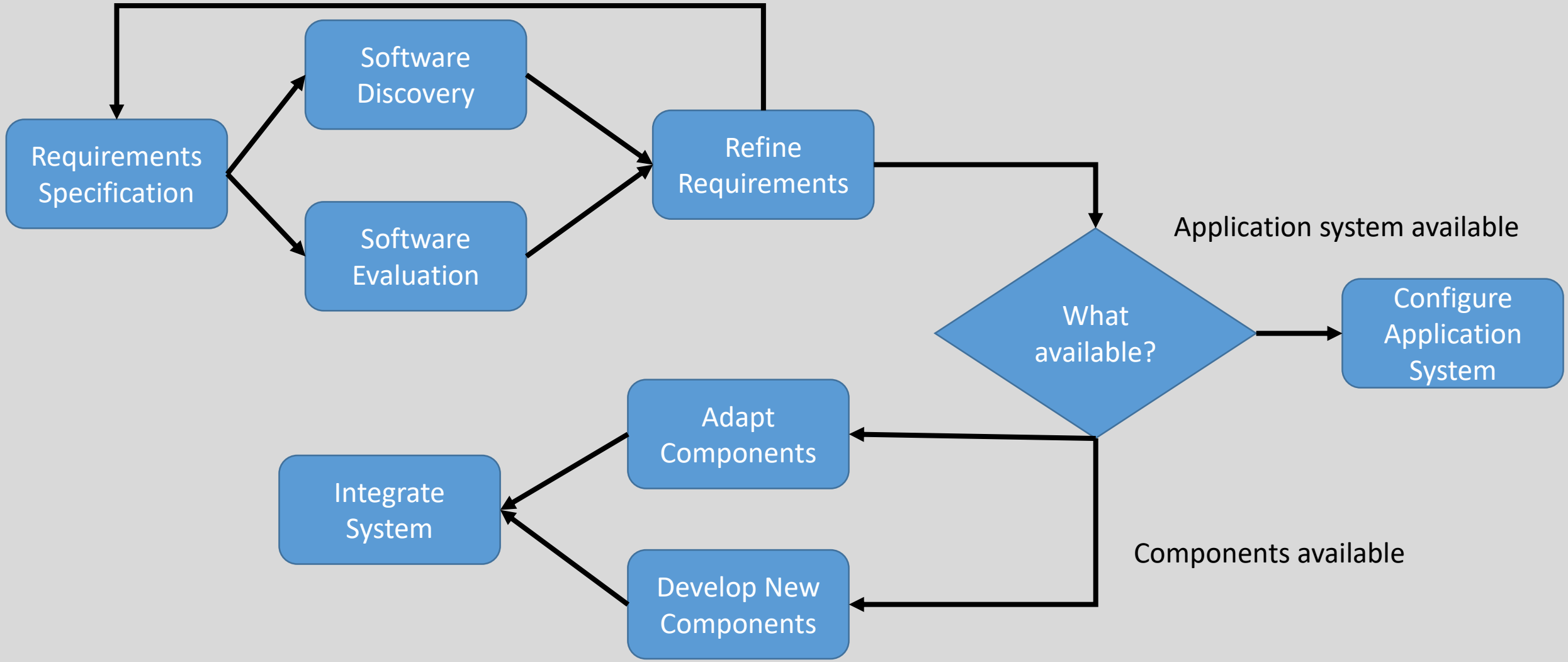


Figure 2.3 (Sommervilel, 10th Edition)

Why Reuse Code?

Advantages:

- It is often **not practical, nor realistic,** to design and implement everything from scratch
- An element with high code reuse suggests **high quality** code
- Code reuse can **reduce your costs**
- Code reuse **increase product quality**
- Code reuse can **lower the risk of project failure**

Disadvantages:

- Can lead to **requirements compromise**
- Can lead to **loss of control** over some parts of the system

Software Design Process

Software Design Process

- Remember (from previous lectures) the Software Design Process has several stages:
  - Specify
  - Design
  - Implement
  - Test
  - Integrate
  - Evolve
- The *process model* which we choose will affect how these stages interact with each other
- Regardless of the chosen process model, the stages have *similar characteristics*

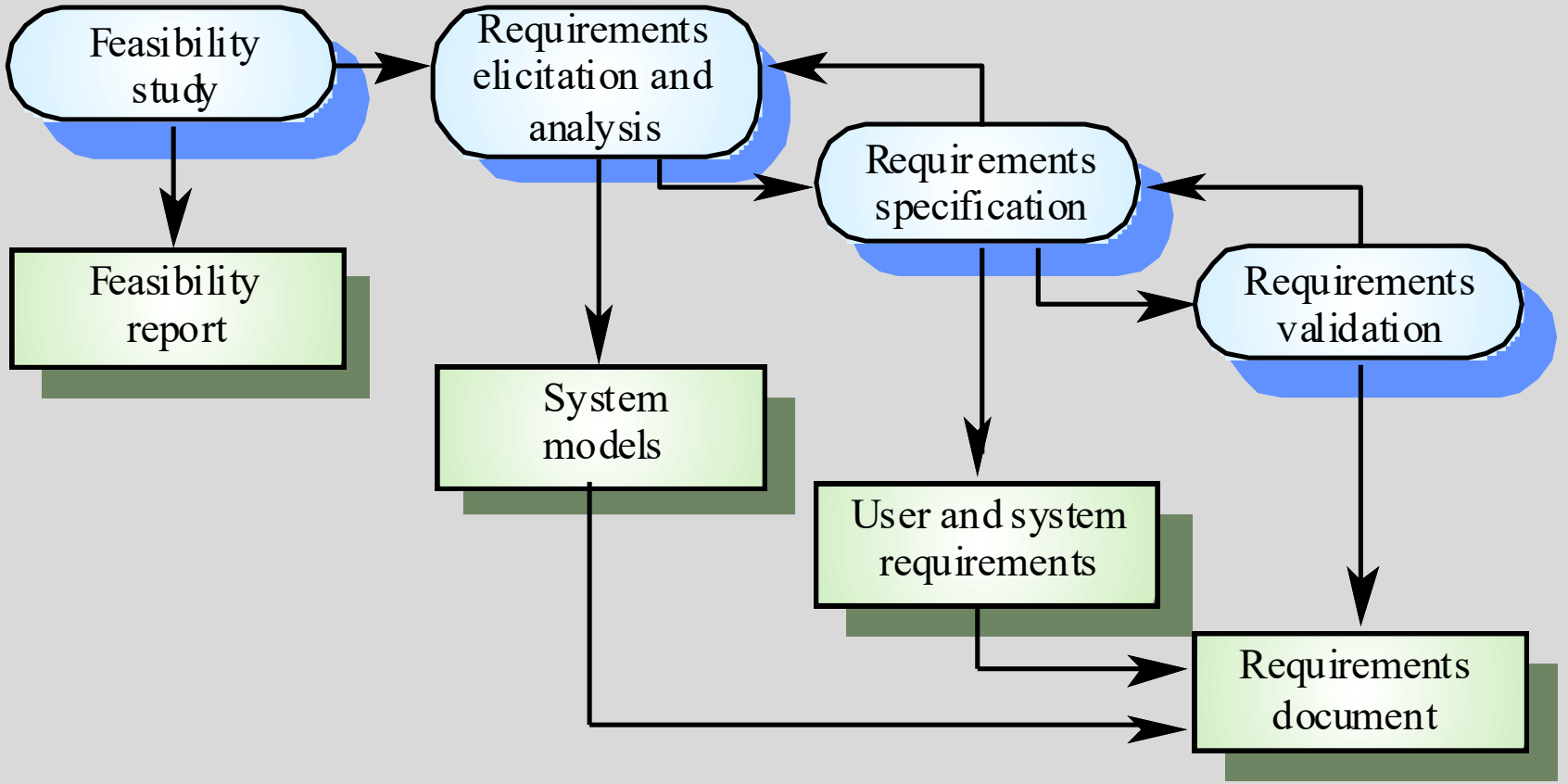
Last lecture, we asked….

“The specification is the MOST critical phase of any software engineering project.”

Specify

Specify the Requirements

- Establish what services are required and the constraints on the system's operation and development using the **Requirements Engineering Process**



Design

Software Design and Implementation

The process of converting the system specification into an executable system

- **Software design**
  - Design a software structure that realises the specification
  - Tasks .. Design database, website design, data structures, communications protocols
- **Implementation**
  - Translate this structure into an executable program

- The activities of **design** and **implementation** are closely related and **may be inter-leaved**

Design Process Activities

- **Architectural design (separate web service modules)**
  - The sub-systems making up the system and their relationships are identified and documented.
- **Abstract specification**
  - For each sub-system, an abstract specification of its operational constraints and services is produced.
- **Interface design**
  - For each sub-system, an unambiguous interface with other sub-systems is designed and documented
    - Formal specification may be used in this stage (we study this later)

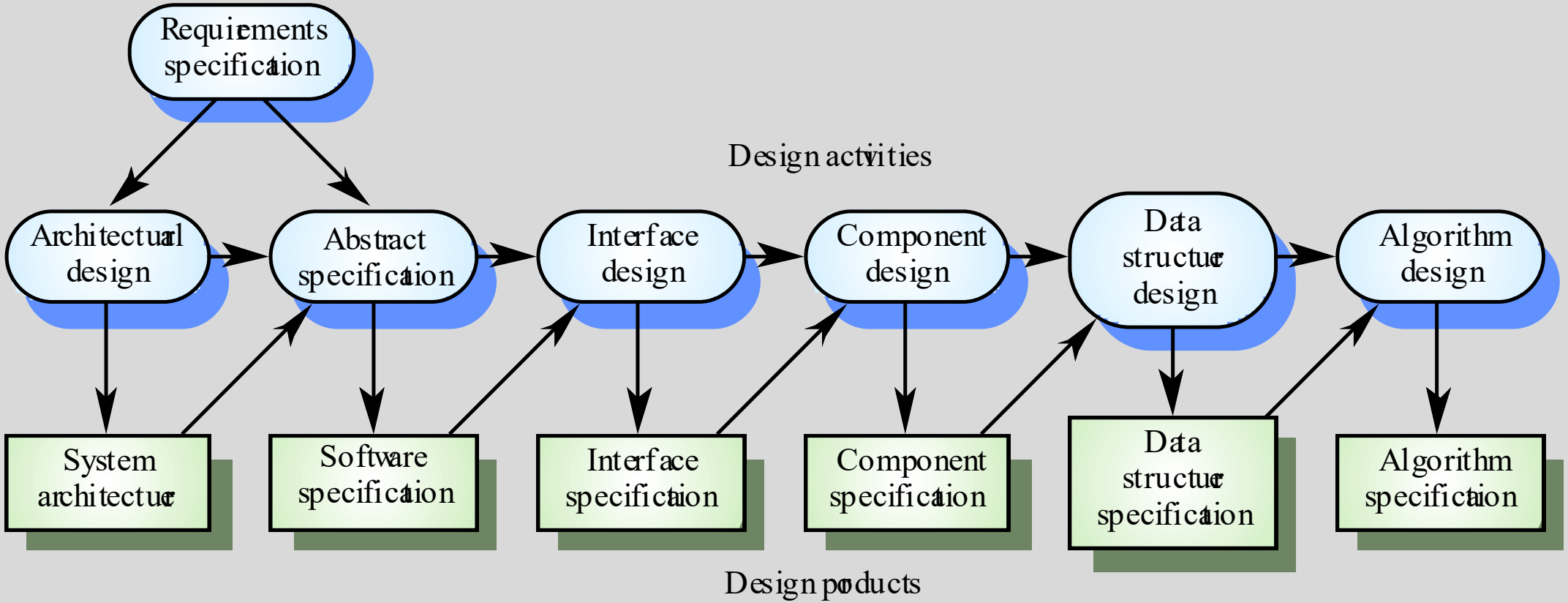
Design Process Activities

- **Component design**
  - Services are allocated to components and the interfaces of these components are designed
- **Data structure design**
  - The data structures used in the system implementation are designed in detail and specified
- **Algorithm design**
  - The algorithms used in components to provide services are designed and specified

An Example System

- Consider the scenario of developing a Coffee/drinks machine software
- What are the major sub-systems?
  - Graphical display
  - Cash handling
  - Accounting
  - Safety system
  - Recipe handling
  - Stock control
- How may we define an abstract specification for each? How do the different sub-systems interact?
- Can you define specifications for components/data structures and algorithms for one of the sub-systems?

The Software Design Process



Cash handling

- Abstract specification
  - Registers entry of new coins with updated balance
  - Handles return of change
  - Can be interfaced to wide range of coin handling mechanisms
  - Interfaces with note acceptor hardware
  - Locks coin mechanism when machine is out of order
  - Rejects incorrect currency

Models

- Graphical views of the operation/structure of the system
- Can be dynamic or static
- Why have models
  - Formalizes the type and format of required information
  - Easier to get the big picture than text documents
  - Do not rely heavily on natural language to be understood
  - Some, can be translated automatically to software implementation
  - Can be tested for validity automatically

Design Methods

- **Design (structured) methods** are systematic approaches to developing a software design

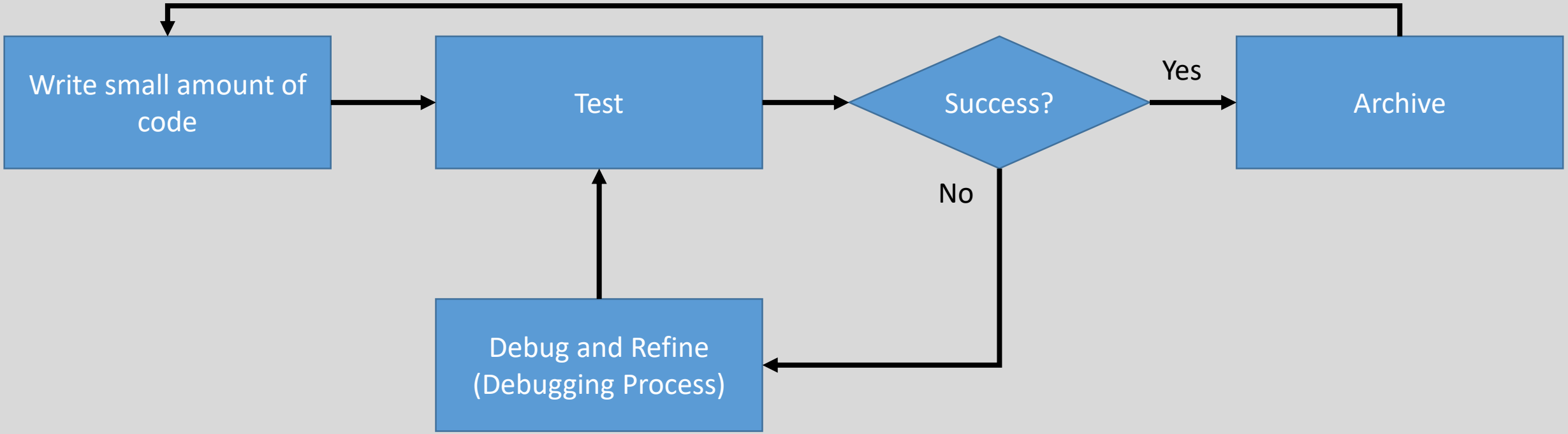
- The design is usually documented as a set of graphical models
  - Data-flow model
  - Entity-relation-attribute model (database or class design)
  - Structural model
  - Object models
  - A state transition model (showing system states and triggers)

Implement

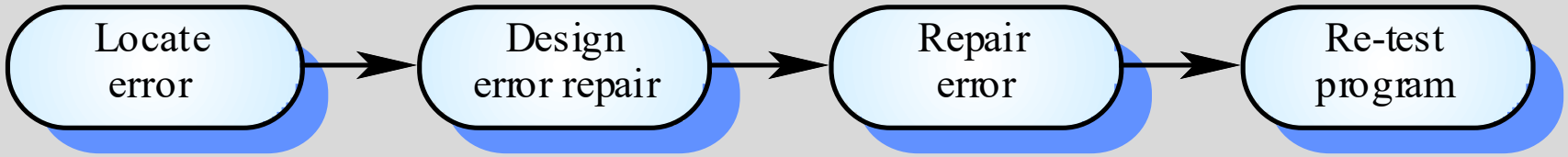
Programming and Debugging

- Translating a design into a program
- Removing errors from that program
- Programming is usually personal activity
  - no generic programming process
  - good programming practices
  - organisational standards
- Programmers carry out some program testing to discover faults in the program and remove these faults in the debugging process

Good programming is iterative



The Debugging Process



Debugging in real world

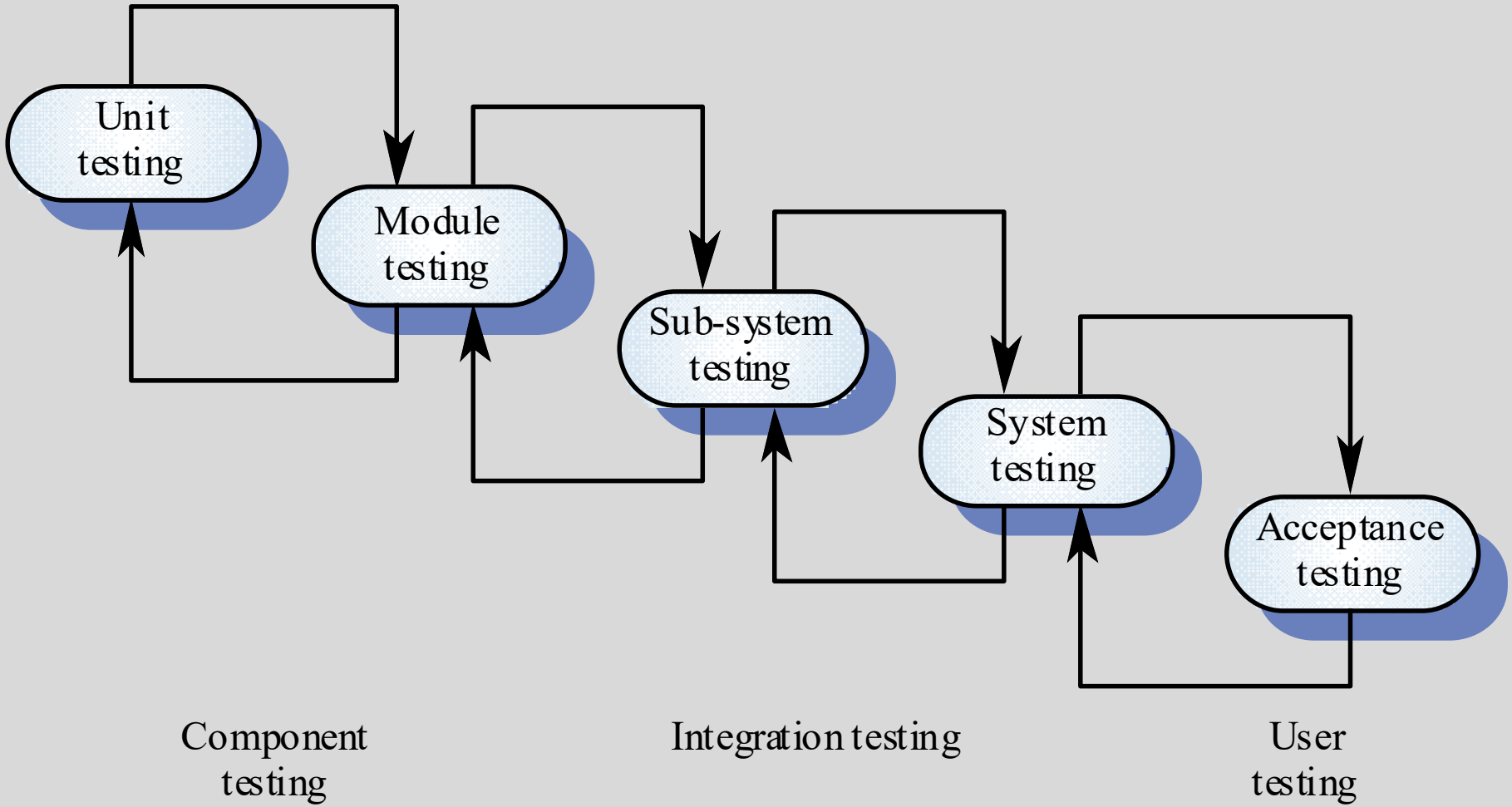
- Ideally the software fault can be re-produced **at will**
- Some software faults indicate problems with overall software design and require application re-design
  - e.g. lack of thread safety
- If bugs are **hard or impossible** to re-produce in test conditions
  - Insert debug/test code embedded into product to log and alert in fault conditions
  - Trace statements
  - Add patch code, which will help recover in fault conditions
    - Example catching exceptions and logging

Test

Software Validation

- **Verification and validation** is intended to show that a system conforms to its specification (**verification**) and meets the requirements of the system customer (**validation**)
- Involves **checking** and **review processes** and **system testing**
- **System testing involves executing the system with test cases** that are derived from the specification of the real data to be processed by the system

The Testing Process



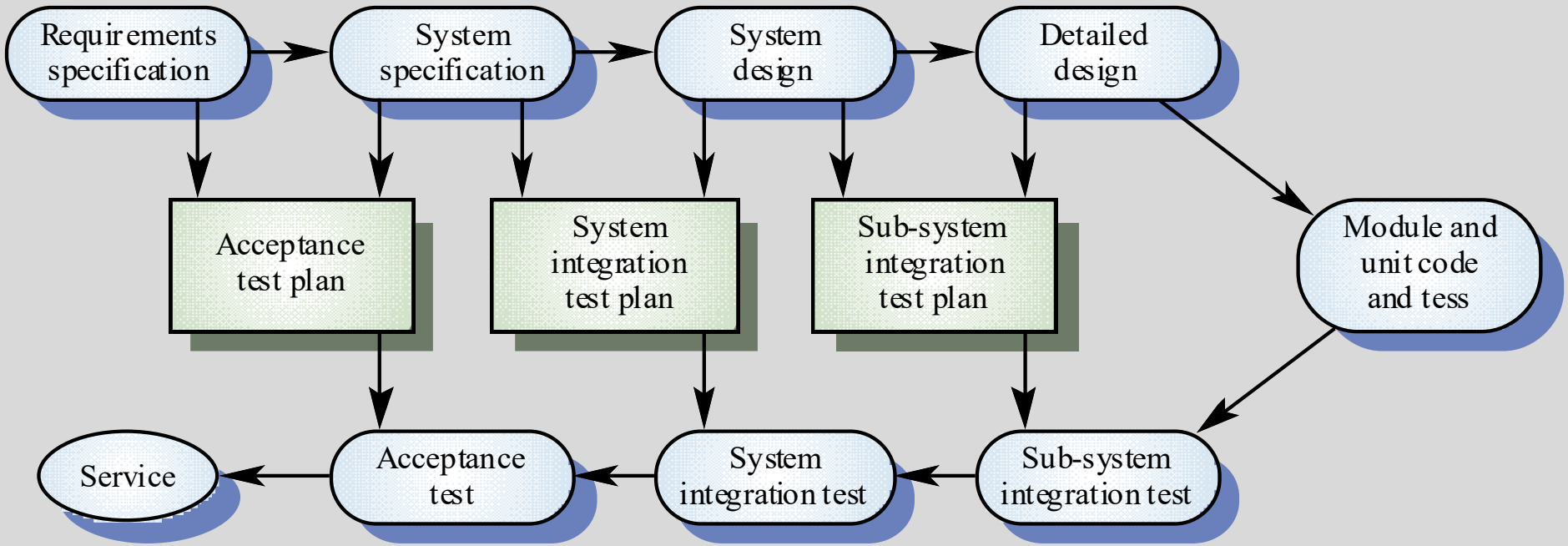
Testing Stages

- **Unit testing**
  - Individual components are tested
- **Module testing**
  - Related collections of dependent components are tested
- **Sub-system testing (merges with system testing)**
  - Modules are integrated into sub-systems and tested. The focus here should be on interface testing
- **System testing**
  - Testing of the system as a whole. Testing of emergent properties
- **Acceptance testing**
  - Testing with customer data to check that it is acceptable

Testing mapped to OO programming

- Unit testing (class/method level)
  - Testing an individual classes methods
- Module testing (interleaved with unit testing)
  - Testing classes which integrate with other classes
- Sub-system testing
  - A number of classes tested which produce a given service (example card payment services, SMS sending services)
  - Organised as package or JAR library
- System test
  - Test whole system

Testing Phases



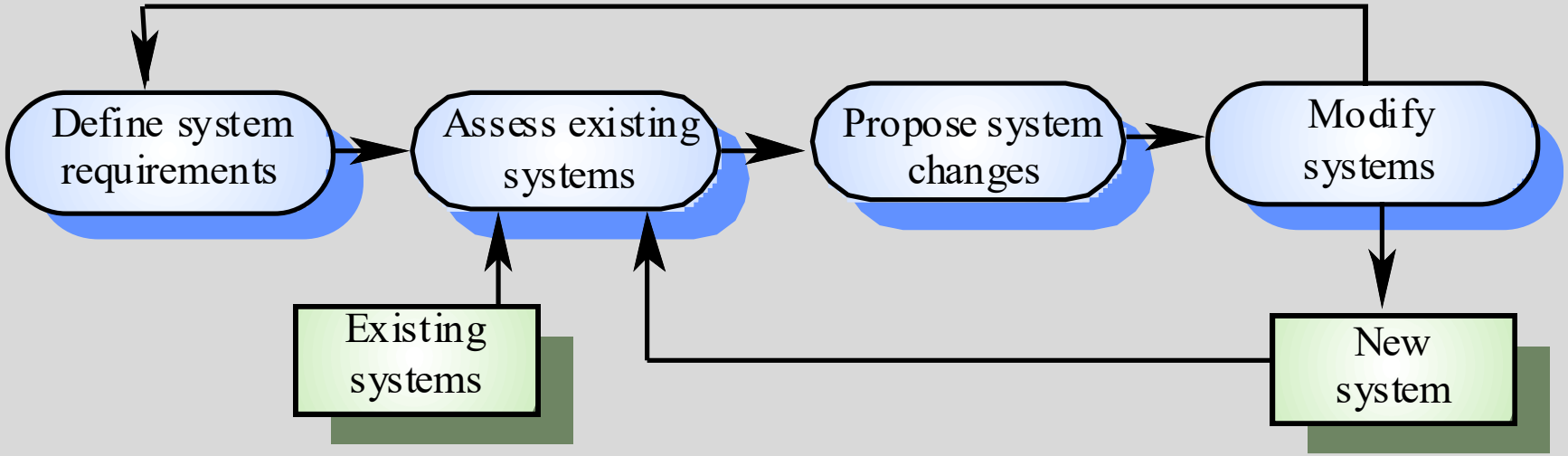
Evolve

Software Evolution

Software is inherently flexible and can change.

- As requirements change through changing business circumstances, the software that supports the business must also evolve and change
- Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new
- It is important to realise that maintenance costs are sometimes several times the initial development costs of the system.

System Evolution



Recap

Lecture Key Points

- Code reuse can be incorporated into the software engineering lifecycle using the Reuse-oriented model
- Requirements engineering is the process of developing a software specification
- Design and implementation processes transform the specification to an executable program
- Validation involves checking that the system meets its specification and user needs
- Evolution is concerned with modifying the system after it is in use