Software Engineering COMP 201

Lecturer: **Sebastian Coope**

Ashton Building, Room G.18

E-mail: ***coopes@liverpool.ac.uk***

COMP 201 web-page:

http://www.csc.liv.ac.uk/~coopes/comp201

Lecture 3 – Software Processes

This lecture will look at

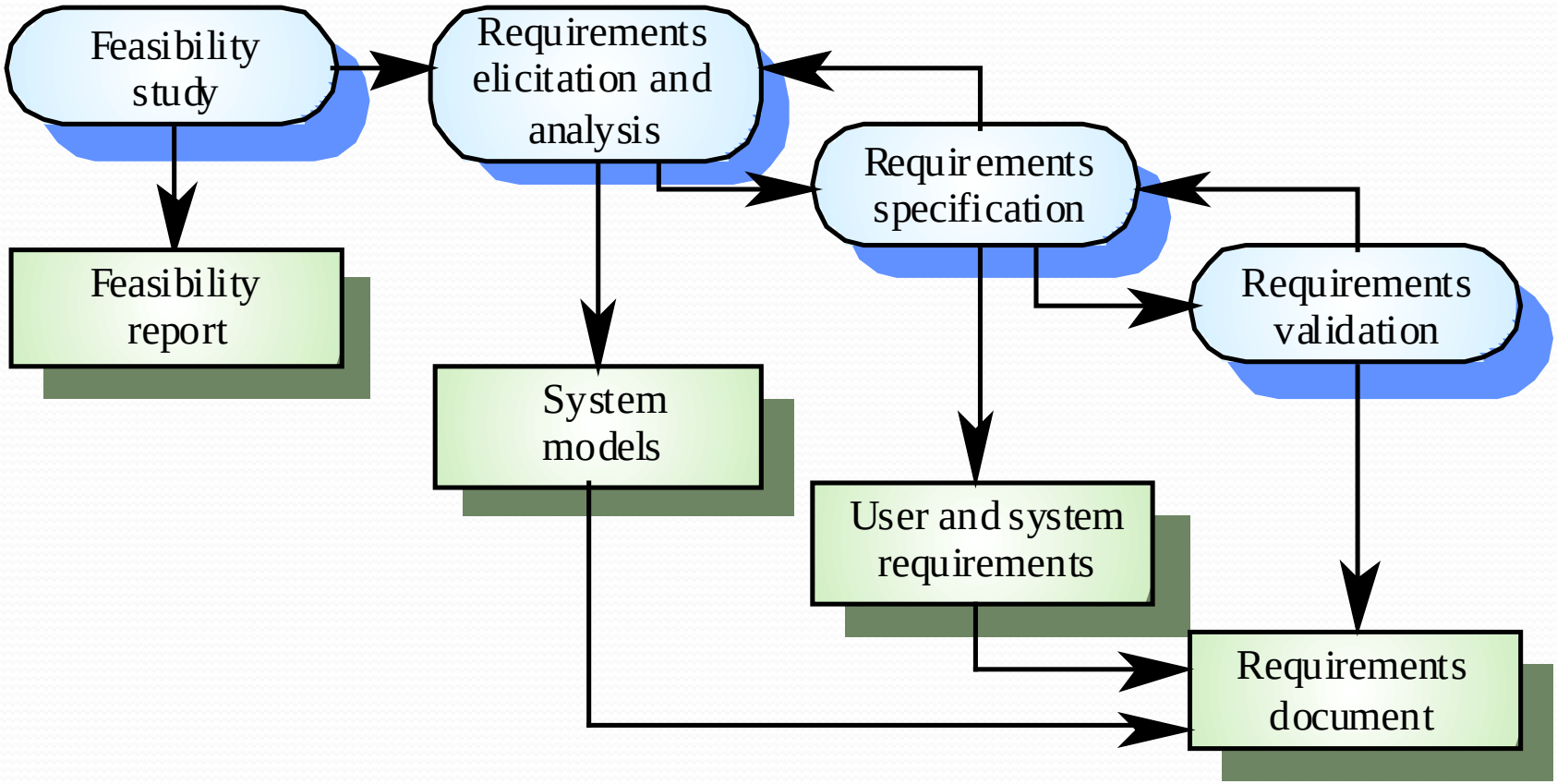
- Requirements engineering and specification
- Software design
- Programming, testing and debugging
- Software Evolution

Software Specification

- **Software Specification: The process of establishing** what services are required and the constraints on the system's operation and development

- **Requirements Engineering Process**
  - Feasibility study
  - Requirements elicitation and analysis
  - Requirements specification
  - Requirements validation

The Requirements Engineering Process



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

Extremes of design philosophy

- Data driven design
  - Always start be looking at all the data the system must handle
  - Describe the relationships of the data and how it can be manipulated
  - For each data item describe how it can be operated upon (functions)
- Responsibility driven design
  - Think of the functions (responsibilities of the system)
  - Break complex functions into simpler functional parts
  - Each responsibility may require data to support its action

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?

Cash handling

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
  - Works with coins from many different countries

Design Methods

- **Design (structured) methods** are systematic approaches to developing a software design
- The design is usually documented as a set of graphical models
- **Possible models** (we study these in detail in later lectures)
  - Data-flow model (data flows between processes)
  - Entity-relation-attribute model (data base or class design)
  - Structural model (shows major sub-systems)
  - Object models (objects have state and behaviour)
  - A state transition model showing system states and triggers

Programming and Debugging

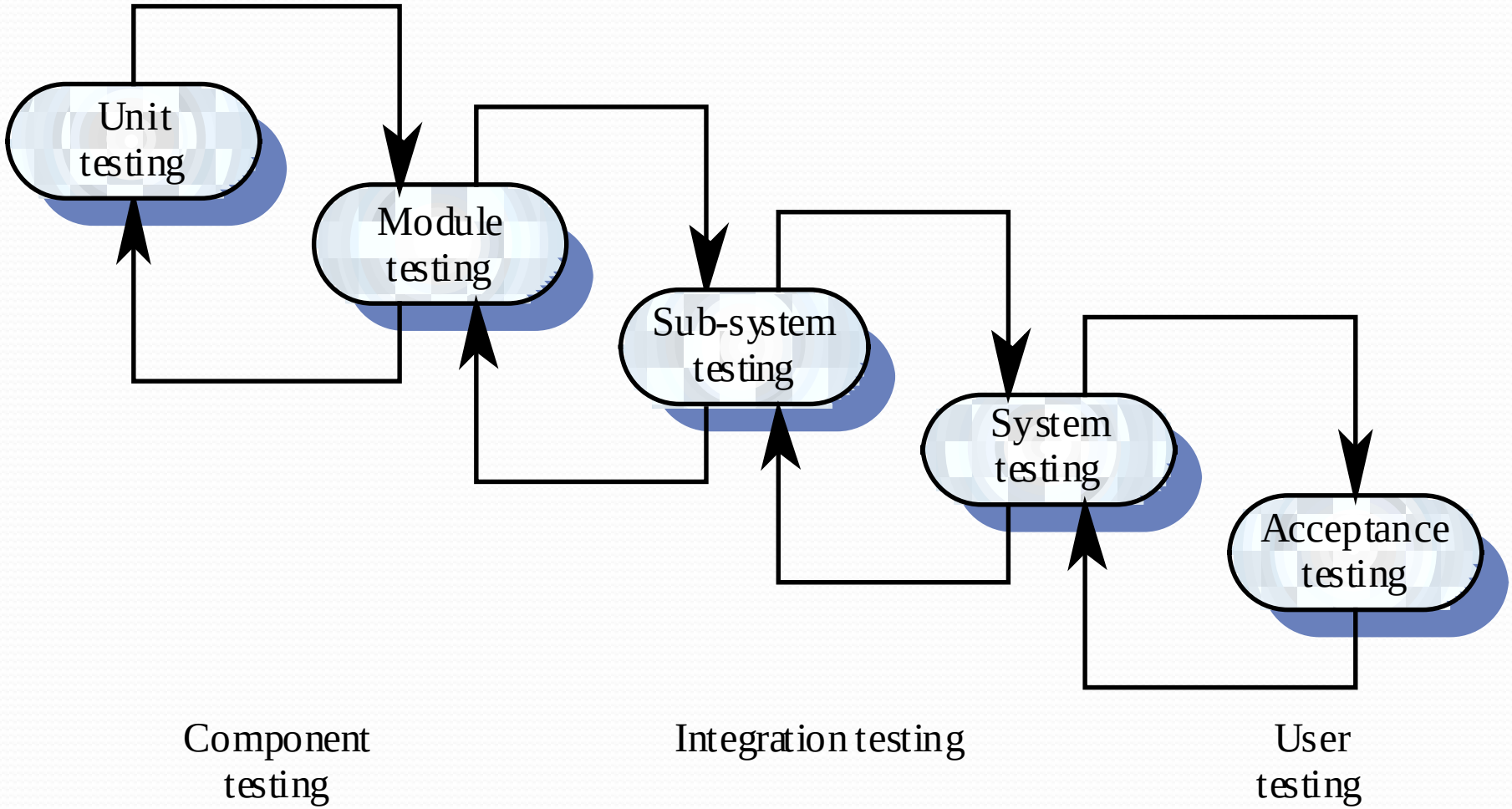
- Programming and Debugging consist of translating a design into a program and removing errors from that program
- **Programming is usually personal activity** - there is no generic programming process, but there are good programming practices and **organisational standards** to be followed.
- **Programmers carry out some program testing** to discover faults in the program and remove these faults in the debugging process

Good programming is iterative

- Write a very small piece of code
- Determine it works (test it)
- Archive it (git or svn commit)
- Add little bit to code, test it
- Archive it
- Add little bit to code, test it

- Etc. etc.

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 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

Next Lecture

- Requirements engineering is the process of developing a software specification
- In the next lecture we will be looking at
- REQUIREMENTS
- This will be part of coursework 1