Triển khai nội dung nghiên cứu (Phần 1) – Conducting the project

Intro

- **Aims:**
  - To introduce different approaches for developing software systems, testing those systems and ensuring software quality.
- **Learning objectives:**
  - Understand what is meant by a software development process and describe a number of different development processes.
  - Choose an appropriate process for your own project.
  - Evaluate your chosen process.
  - Understand the differences between verification, validation and testing and apply these techniques to your own project.

Introduction

- At one extreme,
  - your course may require you to undertake a software development as the fundamental component of the project;
- At the other extreme
  - you may just decide to develop a small program to evaluate some ideas in a more research-oriented project.
- Note that the term *process* model and *life cycle* model are two terms that are used interchangeably.

1. The software development life cycle (SDLC)

- The SDLC represents a generic model for software development and consists of a number of stages.
- These stages, shown in Figure 6.1, are: *requirements capture, design, build, test* and *implement.*
  - All software developments follow this generic model in one way or another and yours will do the same

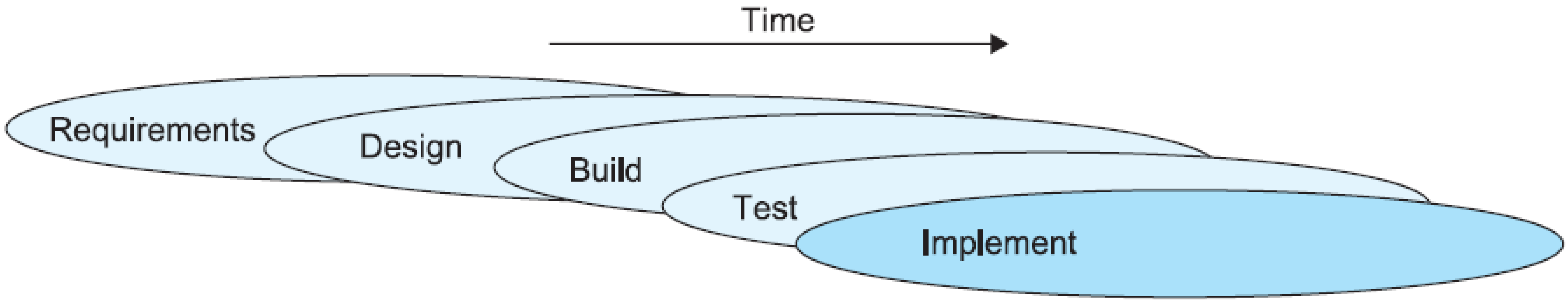

Figure 6.1 The software development life cycle

Requirements

- The outcome from this stage of the process is a series of documents that clearly define what the software system is required to do (but **not** how it should do it – that is the purpose of *design*).
- These documents should be produced in the following order:
- **1.** Requirements definition
- **2.** Requirements specification
- **3.** Functional specification

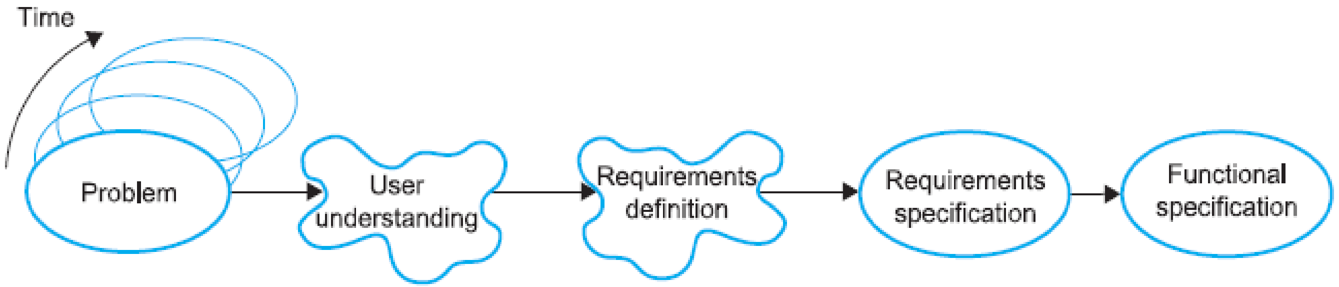

Figure 6.2 The conventional stages of requirements capture

Design

- Design works on a number of levels and a number of issues.
  - For example, systems can be designed as a series of modules (or objects) that are gradually brought together to produce a fully working system.
- Design can include flowcharts and pseudocode that plan how certain functions within the program will operate.
  - Object-oriented design techniques (for example, UML) can be used to build systems from a series of objects.

Design

- Design can also encompass interface issues –
  - human computer interaction (HCI), screen layouts, navigation between screens, and story boards.
- Design might include database design –
  - for example, structuring data tables using *Normalisation* techniques or *Entity Relationship* modelling.
- You should consult with your supervisor over the most appropriate design techniques to use for your project

Build

- Build simply represents the coding and construction (*bringing together the individual modules or objects*) of the software system.
- How this is achieved depends largely on
  - the programming language(s) used,
  - the design methods used and
  - any coding standards and
  - quality standards you might be following

Test

- This is the final testing of the system as it is brought together into a working whole.
- It might take a lot of time

Implementation

- The final stage of the SDLC is implementation.
  - This represents the final hand-over of the system to the user.
  - It can include acceptance testing by the user,
  - it will invariably involve training,
  - it might involve a formal handover, the setting up of data files, implementing new work procedures, documentation and so on.
- In this stage of the process *change management* is particularly important.
- This can also include overcoming resistance to change and data migration issues, etc

The earliest ’model’: build-and-fix

- As we saw earlier, in the pioneering days of software development there was no recognized process for developing software.
- Programmers merely attempted to grasp an understanding of the problem as best they could and ‘cobbled together’ some code to address this problem.
- The *build-and-fix* or *code-and-fix* ‘model’ represents this early approach to the development of software

The stage-wise and classical waterfall models (conventional models)

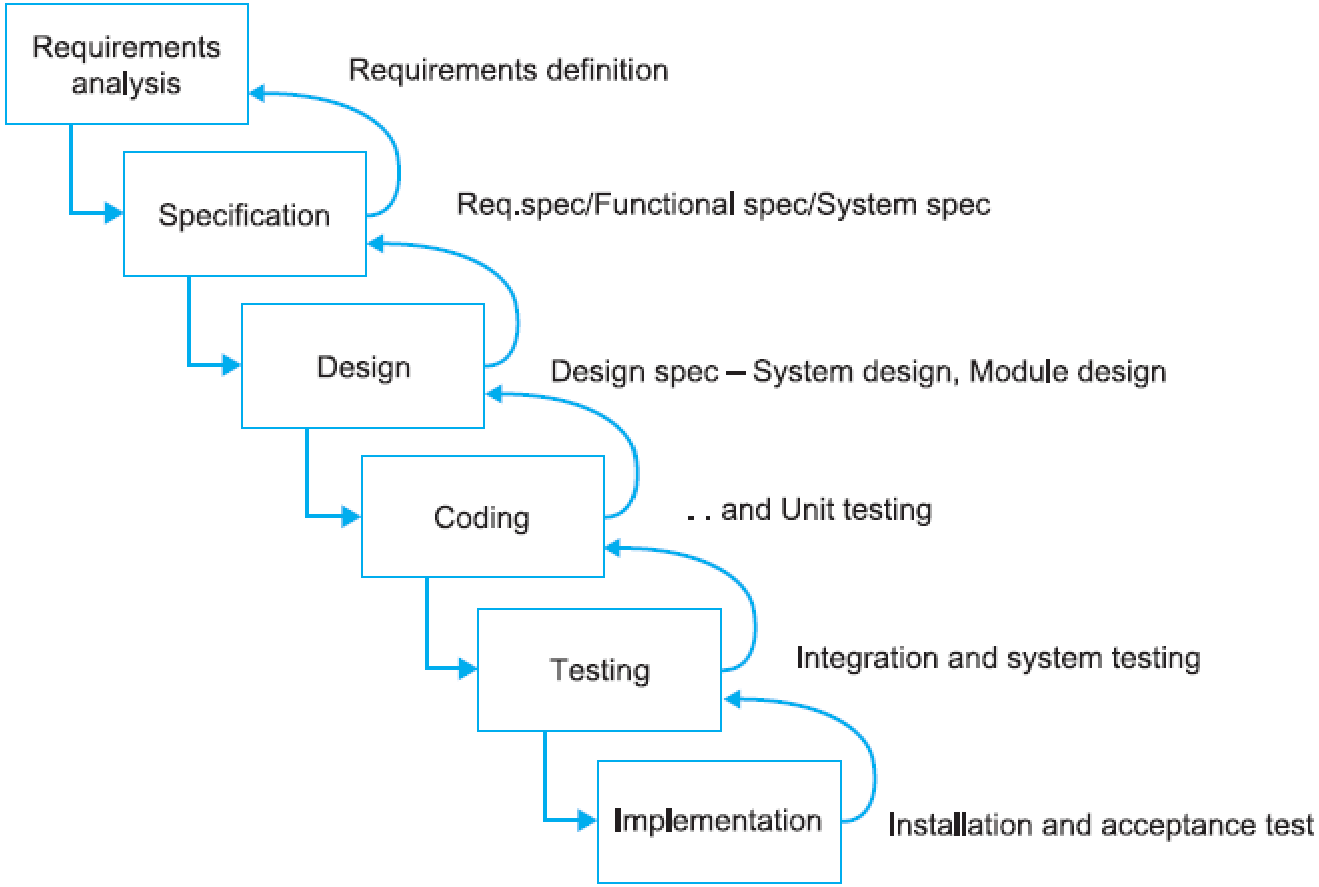
Requirements analysis
Requirements definition
Specification
Req.spec/Functional spec/System spec
Design
Design spec – System design, Module design
Coding
. . and Unit testing
Testing
Integration and system testing
Implementation
Installation and acceptance test

Figure 6.4 The classical waterfall model

The incremental model

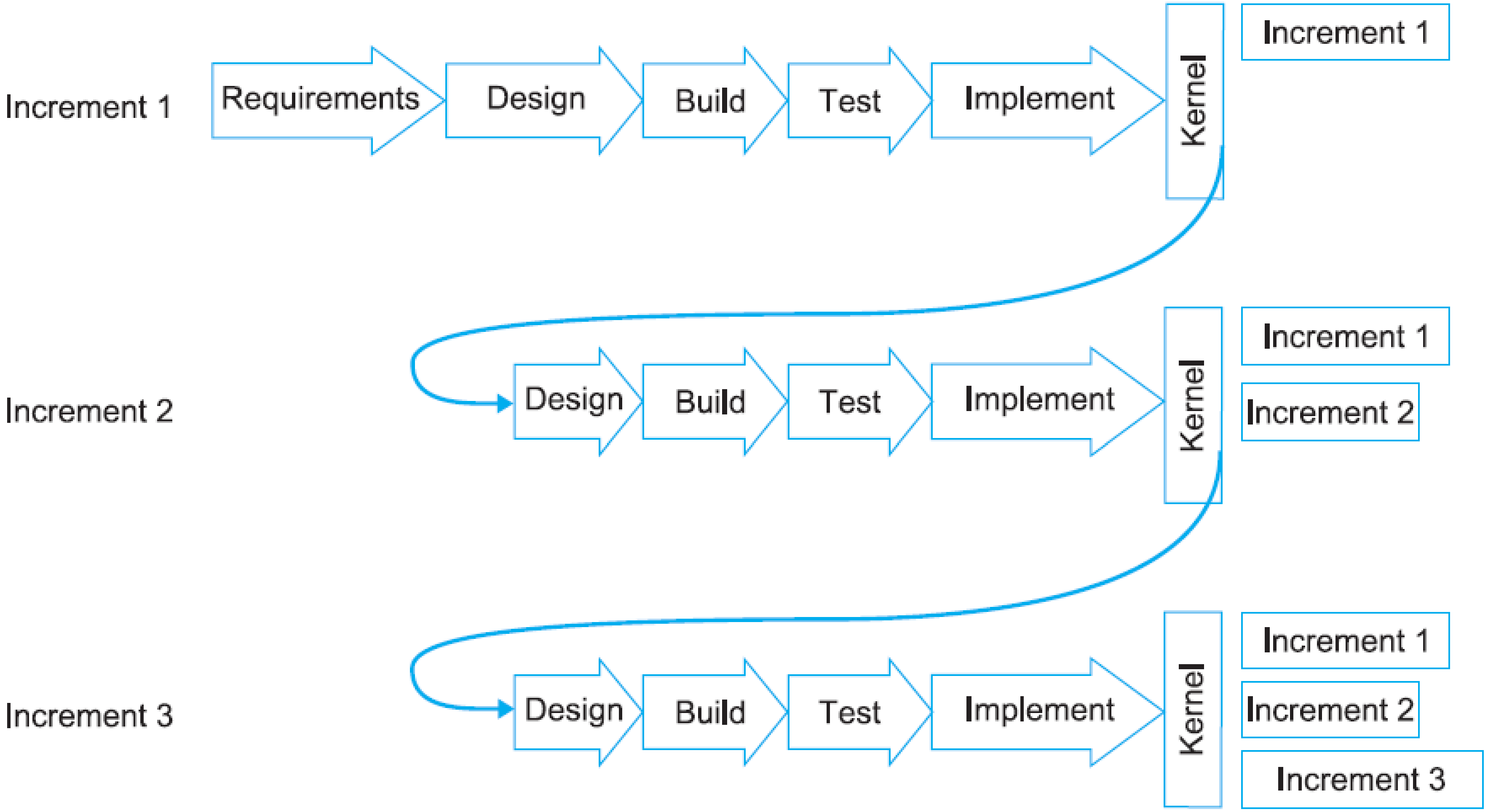

Figure 6.5 The incremental model

Prototyping

- The *conventional* and *incremental* models that have been introduced can be used in projects where
  - the problem is well defined,
  - the requirements can be clearly elicited and defined, and
  - the technical feasibility of a solution is understood.
- However, in many projects it is often difficult to pin down exactly what is required from a software system at the start of the project and/or we may not have a clear understanding of the technical issues surrounding that system.
- This is often the case in student projects where they are working with a supervisor or client for the first time, *perhaps in a new field or in a developing research area*. In these cases it is useful to produce a prototype

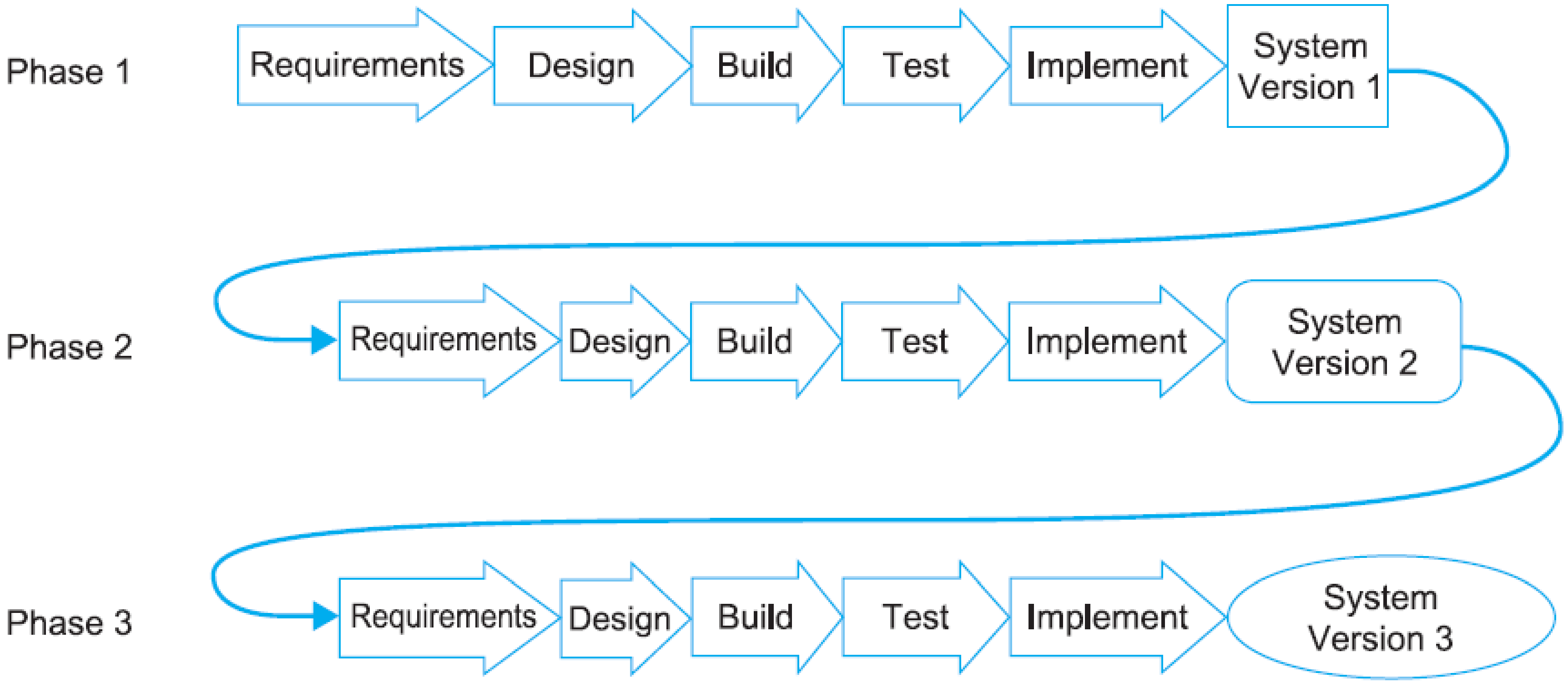

Figure 6.6 The evolutionary prototyping model

Agile methods

- It refers to approaches to software development that reduce risk by delivering software systems in short bursts or releases.
- The concept behind agile methods is to *release working systems to the user in a matter of weeks*.
- Although each iteration might not release a fully working system to the user at the end of each cycle, the aim is still to have an available release at the end of each iteration

Agile methods

- The other main characteristics of agile methods that differentiate them from older, more conventional models include:
  - their emphasis on smaller development teams (which are invariable working together in open plan offices); and
  - face-to-face communication with the users who are quite often based in the same working environment as the developers.
- Agile methods are well suited to projects that
  - have unclear or rapidly changing requirements;
  - the project team is fairly small but highly competent and can be trusted; and
  - close interaction with the user must be possible

Extreme programming (XP)

- Extreme programming is a software development approach that encompasses many of the ideals of agile methods.
  - It was introduced in the 1990s in an attempt to improve the way in which software is developed.
- It is designed for teams of between two and 12 members, so it is ideally suited for student projects
- Extreme programming is an approach that is well suited to projects in which the requirements are likely to change.
- It emphasizes teamwork and,
  - in the case of student projects, encourages the users, supervisor(s) and the project team to work together towards a common goal of developing quality software

Verification, validation and testing

- *Verification* is the process of checking that we are performing our development correctly
- *Validation* is checking to see if that system is really what the client/user needs at all
- *Testing* refers to the testing of the program itself to see if it works or has any bugs or errors within it

Questions