LAU

Lebanese American University

Chartered in the State of New York

www.lau.edu.lb

SOE

School of Engineering

Department of Electrical & Computer Engineering

www.lau.edu.lb/ece

BYBLOS CAMPUS	BEIRUT CAMPUS	NEW YORK HEADQUARTERS
P.O. Box: 36	P.O. Box: 13-5053 Chouran	211 E. 46th Street
Byblos, Lebanon	Beirut 1102 2801, Lebanon	New York, NY 10017-2935, USA
Tel: +961 9 547 262	Tel: + 961 1 786 456	Tel: +1 212 203 4333
Fax: +961 9 546 262	Fax: +961 1 867 098	Fax: +1 212 784 6597

Software Engineering

COE 416

Chapter 2 – Software Processes

Dany Ishak, Ph.D.

Computer Software Engineering

Ch. 2 - Software Processes

Chapter Contents

Ch. 2 - Software Processes

1. Introduction

- **Software Process:** Set of related activities leading to the production of a software
  - Ranging over software development from scratch to reusing existing components

- Many different software processes exist
  - They mainly consist of four main **activities**:
    - **Software specification**: defining the functional/non-functional constraints and requirements of the software to be produced
      - Identified by the engineer and the customer
    - **Software development**: design and programming
    - **Software validation:** checked and tested to verify functional/non-functional requirements
    - **Software evolution:** maintenance and extensibility to satisfy customer needs

1. Introduction

- Software process **activities** are themselves complex
  - Including **sub-activities** e.g., requirements validation, architectural design, etc.

- In addition to **activities**, software processes include:
  - **Products**: the outcomes of process activities (e.g., reports, diagrams, code, etc.)
  - **Roles:** Responsibilities of people involved in the process (manager, programmer, etc.)
  - **Pre-** and **Post- conditions:** statements that have to be verified before/after a process activity is executed, for instance:
    - All requirements have to be met before the activity is executed
    - After activity is executed, we verify that the result conforms to the objectives

In this chapter, we mainly focus on **activities** and **products**

⇨ **Roles** and **conditions** will be covered in later chapters

1. Introduction

- Software processes are intellectual and creative processes
  - There is no predefined (set of) solution(s)
    - Most organization adopt/develop their own processes
      - Taking advantage of the organization’s recourses and project constraints

- For instance:
  - For **critical systems**: a very structured process is required
    - Activities are pre-planned and progress is measured against the plan

      ⮡ Plan-driven processes

  - For **online business systems** with rapidly changing requirements
    - A incremental, flexible process, where activities are interlaced: is likely more effective

      ⮡ Agile processes

1. Introduction

- While there is no “ideal” software process for all situations
  - Standardizing a set of processes is extremely beneficial:
    - **Improved communication** across an organization and between organizations
    - **More economical** automated software support
    - **Common basis** for introducing new software processes and techniques

↳ This chapter introduces and discusses some standard SPs

⇨ Developing underlying activities

Ch. 2 - Software Processes

Chapter Contents

2. Software Process (SP) Models

- **SP Model:** Simplified representation of a software process
  - Represents a software process from a particular perspective
    - Provides only partial information about the process
      - It's like a multi-dimensional graph where each model describes one dimension

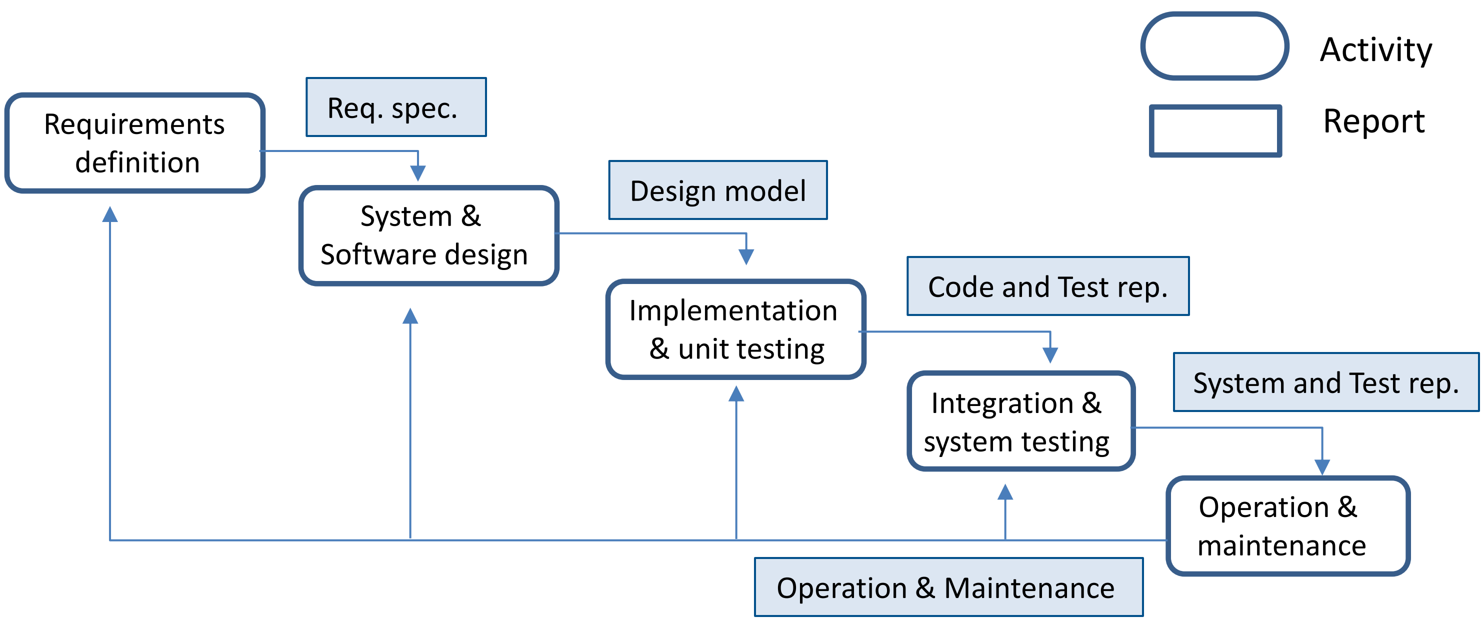
- Different general SP models exist, among which:
  - The **waterfall** model
  - The **incremental development** model
  - **Reuse-oriented (component-based) SE** model
  - **Spiral** model
  - **Chaos** model

2. Software Process Models

Does this below graph come to exam?

2.1. The Waterfall Model

- One of earliest models (Royce, 1970)
  - Considers the basic SP activities: specification, development, validation, and evolution, **separately** and **sequentially**



2. Software Process Models

2.1. The Waterfall Model

- Waterfall model activities:
  - **Requirements analysis & definition:** Establishing system services, constraints, & goals : serving as formal system specification
  - **System and software design:** Establishing an overall system architecture, allocating requirements to either software/hardware systems
    - Defining fundamental software system architecture, abstractions and relationships
  - **Implementation & unit testing**: software design in a set of programs and program units, verifying that each unit meets its specifications
  - **Integration & system testing:** Program units are interrelated and tested as a whole
    - After system testing, the software is released to the customer
  - **Operation & maintenance:** Correcting errors missed earlier, enhancing implementation and system services as new requirements are discovered

2. Software Process Models

2.1. The Waterfall Model

- Waterfall model: typical **plan-driven** model
  - All activities must be planned and scheduled in advance before working on them
  - The result of each phase is a (set of) document(s) to be approved
  - The next phase doesn't start until the previous phase has been approved
  - Feedback is limited between activities:
    - A document produced at a previous phase cannot be changed in a subsequent phase
      - Even though it could be done in practice, yet it would prove to be very costly
  - Problems found at one iteration are left for later resolution
    - Ignored or programmed around
      - ⇨ Premature freezing of requirements: so that process flow is not disturbed

2. Software Process Models

2.1. The Waterfall Model

- **Benefits:**
  - Model consistent with general engineering process model
  - Documentation is produced at each activity phase
    - Enhanced visibility for managers to facilitate monitoring processes against plan

- **Problems:**
  - Inflexibility: partitioning the process into separate and sequential phases
    - Commitments must be made early in the process:
      - The system might not finally answer all requirements
      - Difficult to respond to changing customer requirements
      - Producing a badly structured system
        - ⇨ As problems are circumvented by implementation

2. Software Process Models

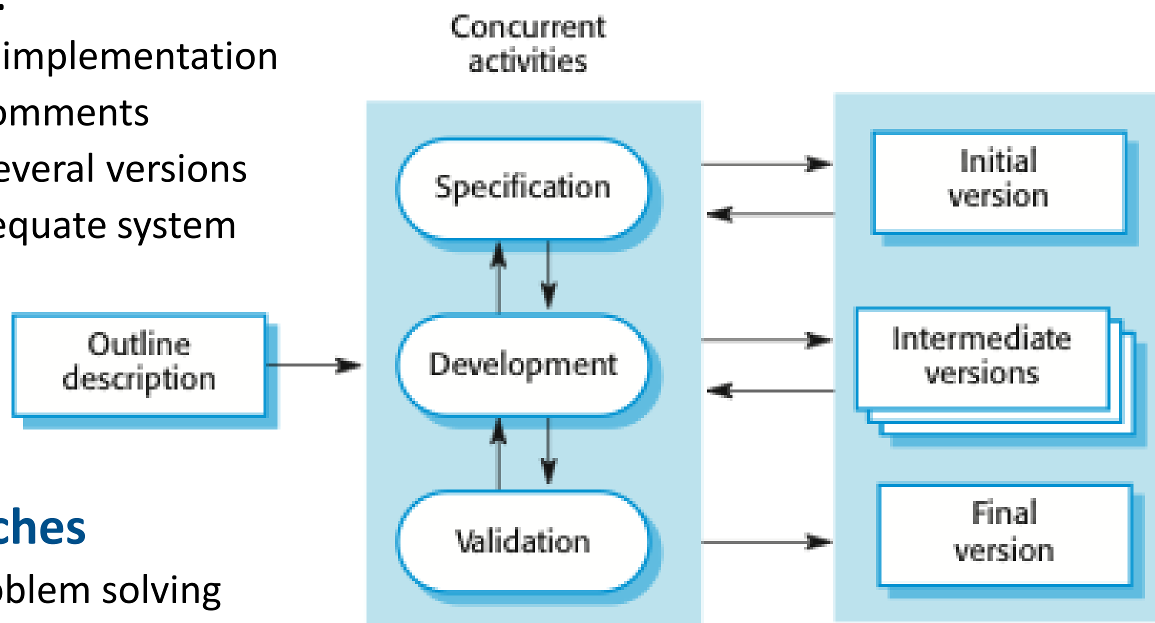
2.1. The Waterfall Model

- **Usability:**
  - Still commonly used because of its ease of management and tractability
    - Especially with large systems engineering projects developed on different sites
      - Helps in better coordinating the work
  - Suited when system requirements are well understood and unlikely to change during system development
    - Such as **critical systems**: with strict safety, reliability, & security requirements
      - **Safety-critical** systems: whose failure may result in injury, loss of life, or serious environmental damage (e.g., control system for a manufacturing plant)
      - **Mission-critical** systems: whose failure may result in the failure of goal-directed activity (e.g., spacecraft navigation system)
      - **Business-critical** systems: whose failure may result in high costs for the business (e.g., accounting system in a bank)

2. Software Process Models

2.2. Incremental Development

- Main process stages:
  - Developing an initial implementation
  - Exposing it to user comments
  - Evolving it through several versions
  - Until reaching an adequate system



- Part of **agile approaches**
  - Similar to human problem solving
    - Rarely producing a complete solution

      ⇨Series of incremental solutions backtracking when needed

2. Software Process Models

2.2. Incremental Development

Benefits:

- **Reducing cost** of accommodating <u>changing</u> customer <u>requirements</u>
  - The amount of analysis and documentation to be redone is much less
    - In comparison with the waterfall model
- Easier to get **customer feedback** on the development work already done
  - Customers can comment on demonstrations of the software
- **More rapid delivery** and deployment of useful software to the customer
  - Customers can use and gain value from the software throughout development
    - Earlier than is possible with a waterfall process: delivery upon completion
- **Lower risk** of overall project failure
- The **highest priority** system **services** tend to **receive** the **most testing**

2. Software Process Models

2.2. Incremental Development

Problems:

- The process is **not completely visible**
  - Managers need regular deliverables to measure progress
    - Since the system is developed quickly following this model
      - ⇨ Costly to produce documents that reflect every version of the system
- System **structure tends to degrade** as new increments are added
  - Regular change tends to corrupt its structure
    - Unless time and money is spent on refactoring to improve the software
      - ⇨ Incorporating further software changes is increasingly difficult and costly
- System **specification** is **developed in conjunction with** the **software**
  - Conflicts with intuition and the procurement model of many organization

2. Software Process Models

2.2. Incremental Development

Usability:

- Effective when developing systems with **rapidly changing requirements**
  - Such as with Web-based software: Web applications and Web services
- Developing systems with **fuzzy** and/or **incomplete requirements**
  - Such as with scientific prototypes and research project systems
    - An early example is NASA's space shuttle software:
      - The primary avionics software system, built from 1977 to 1980
      - Applying incremental development in a series of 17 iterations over 31 months
        - ⇨ Averaging around eight weeks per iteration
- Difficult to use with **large** and **distributed systems**
  - Where different teams develop different parts of the system

2. Software Process Models

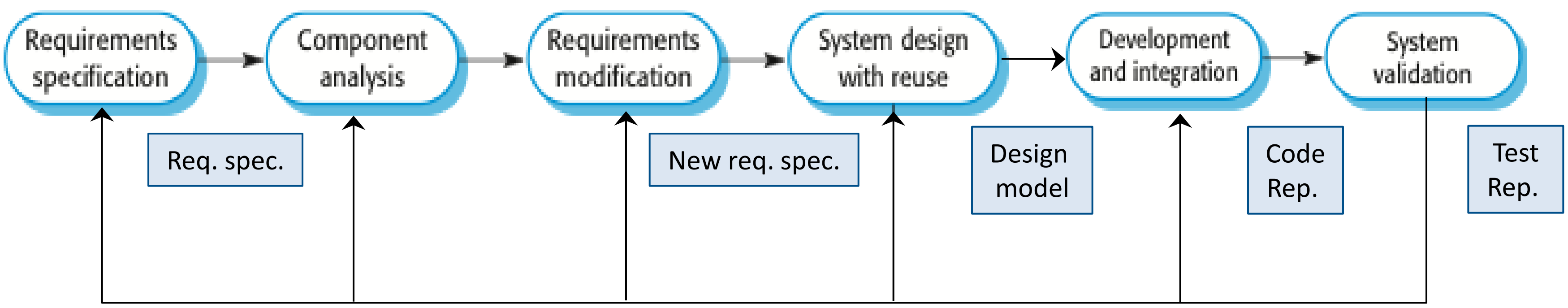
2.3. Reuse-oriented (Component-based) SE

- Based on systematic reuse where systems are integrated
  - From existing components or COTS (Commercial-off-the-shelf) systems
- Main process stages:
  - **Component analysis**
  - **Requirements modification**
  - **System design with reuse**
  - **Development and integration**
- Reuse is now a standard approach for building many types of business systems
  - Especially with the advent of the Web and increasing availability of software

2. Software Process Models

2.3. Reuse-oriented (Component-based) SE

- Based on systematic reuse where systems are integrated
  - From existing components or COTS (Commercial-off-the-shelf) systems



2. Software Process Models

2.3. Reuse-oriented (Component-based) SE

- Types of Components that can be used:
  - **Web services** developed according to service standards
    - Those which are available for remote invocation
  - **Libraries**: Collections of objects that are developed as a package
    - To be integrated within a component framework such as .NET or J2EE
    - External DLL (Dynamic Link Libraries)
  - **Stand-alone software** systems (COTS) that are configured for use in a particular environment
    - Open source software

2. Software Process Models

2.3. Reuse-oriented (Component-based) SE

- **Benefits:**
  - Time and resource saving
  - Making use of existing high quality software instead of reinventing the wheel
  - Faster development and deployment of systems
- **Problems:**
  - Dependability issues: components are not always safe, secure, reliable
  - Maintainability issues: not always easy to update/maintain components
    - Loss of some control over system evolution
  - Risks producing badly structured systems
    - Since we are gluing together prefabricated components
    - Instead of designing and modeling the system from scratch

2. Software Process Models

2.3. Reuse-oriented (Component-based) SE

Usability:

- Increasingly used for its **practicality**
  - Especially in object-oriented and web-based software development
    - With the increasing availability of support libraries and packages

- Corresponds to **Web-based** software development **paradigms**
  - Web services composition
    - Identifying and combining different existing services to perform a certain task
  - Cloud computing
    - Using a certain existing software for a certain period of time

- Difficult to use with **critical systems**
  - Unless components are thoroughly tested

2. Software Process Models

2.4. Boehm's Spiral Model

- Process is **represented as a spiral**
  - Rather than as a sequence of activities with backtracking

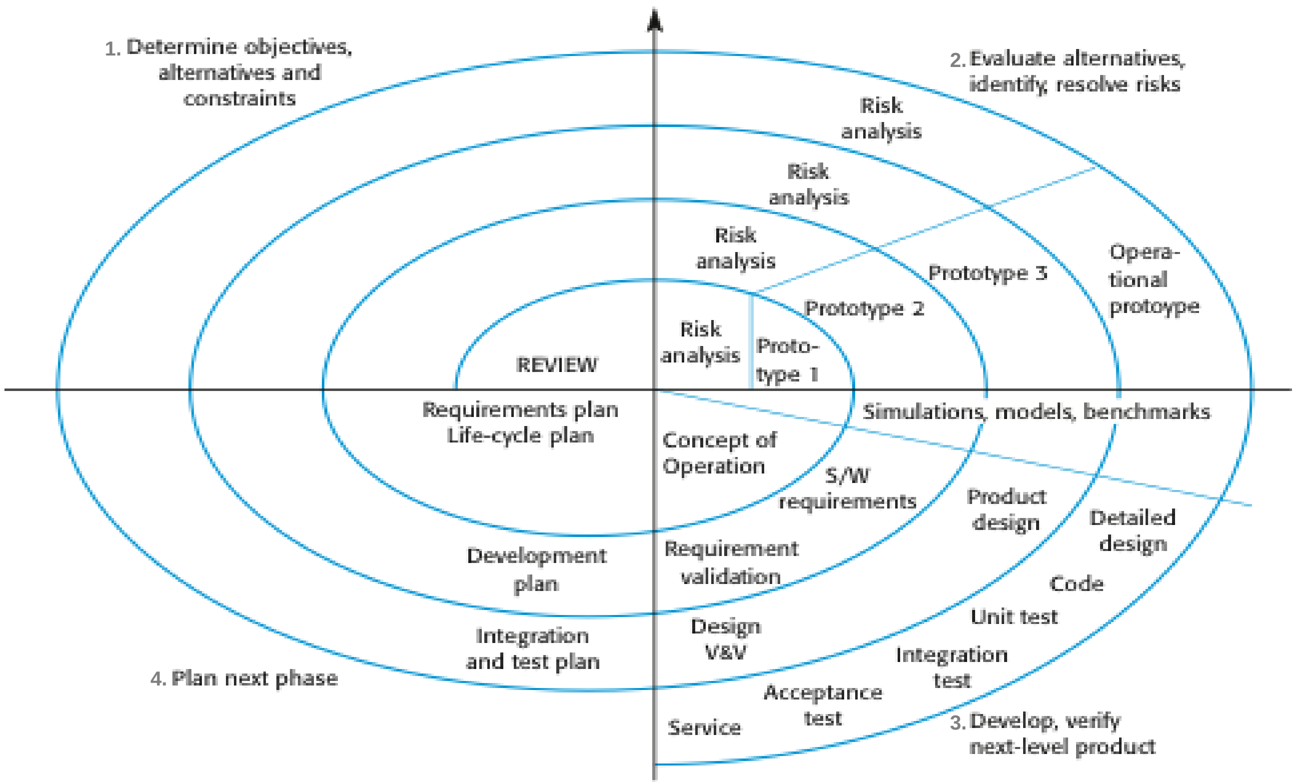
- **Each loop** in the spiral **represents a phase** in the process

- No **fixed phases** such as: specification, or design
  - Loops in the spiral are chosen depending on what is required

- **Risks are explicitly assessed** and resolved throughout the process
  - Risk assessment remains implicit in previous models

2. Software Process Models

2.4. Boehm's Spiral Model



2. Software Process Models

2.4. Boehm's Spiral Model

Spiral Model Sectors:

- **Objective setting**
  - Specific objectives for the phase are identified
- **Risk assessment and reduction**
  - Risks are assessed and activities put in place to reduce the key risks
- **Development and validation**
  - A development model for the system is chosen: It can be the generic models
- **Planning**
  - The project is reviewed and the next phase of the spiral is planned

2. Software Process Models

2.4. Boehm's Spiral Model

Risk Assessment

- **Determining level of effort**
  - Deciding how much effort is enough for each activity (e.g.: requirements, design, etc.)
    - For example, additional testing time can reduce risk due to the marketplace rejecting the product
    - However, additional testing time might increase risk due to a competitor's early market entry

- **Determining degree of detail**
  - Deciding how much detail is enough for each activity
    - For instance, considering the **requirements specification** activity:
      - Specify in detail those features helping reduce risk, e.g., hardware/software interfaces, etc.
      - No need to precisely specify those features not increasing risk, e.g., graphical layouts, etc.

From a spiral model perspective, the objective is to **minimize total risk**

2. Software Process Models

2.4. Boehm's Spiral Model

Usability:

- Influential in helping people think about iteration in software processes
  - Similar to **incremental-driven** approach
    - While introducing the **risk-driven approach** to development
- Suitable for systems with:
  - **Fuzzy or incomplete requirements**
  - **Changing requirements**
- However, the model is rarely used in practice
  - Seems quite complicated for practical software development
    - Even though it shares almost the same benefits as **incremental development**

2. Software Process Models

2.5. Chaos Model

- **Previous SP models** are generally good at managing schedules and staff
  - Not so good in providing methods to fix bugs or solve other technical problems
- **Raw programming skills** are effective at fixing bugs and solving technical problems
  - Not so good in managing deadlines or responding to customer requests

Chaos model attempts to bridge this gap

2. Software Process Models

2.5. Chaos Model

Main process phases: no predefined organization

⇨ Phases apply to all levels of the project

- **Lines of code** are defined, implemented and integrated
- **Functions** must be defined, implemented, and integrated
- **Modules** must be defined, implemented, and integrated
- **System** must be defined, implemented, and integrated

General tendency:

System emerges from the combined behavior of the smaller building blocks

2. Software Process Models

2.5. Chaos Model

Main Premise: Always resolve the most important issue first

- An **issue** is an incomplete programming task
- The **most important issue** can be any or a combination of:
  - **Big issues:** provide value to users as in working functionality
  - **Urgent issues:** timely in that they would otherwise hold up other work
  - **Robust issues:** trusted after being thoroughly tested
    - ⇨ So that developers can then safely focus their attention elsewhere
- **Approach resembling the way programmers work toward the end of a project**
  - When they have a list of bugs to fix and features to create
    - Usually <u>someone prioritizes remaining tasks</u>, and programmers fix them one at a time

3. Process Activities

- Four basic process activities:
  - **Specification**
  - **Development**
  - **Validation**
  - **Evolution**

- Organized differently in different development processes
  - For instance: In the waterfall model, they are organized in a sequence
  - Whereas in incremental development they are inter-leaved

- Each activity consists of different sub-activities organized in a **process**
  - Actives can be viewed as sub-processes of the overall SE process

3. Process Activities

3.1. Software Specification

- It is the process of establishing what services are **required**
  - And the **constraints** on the system's operation and development

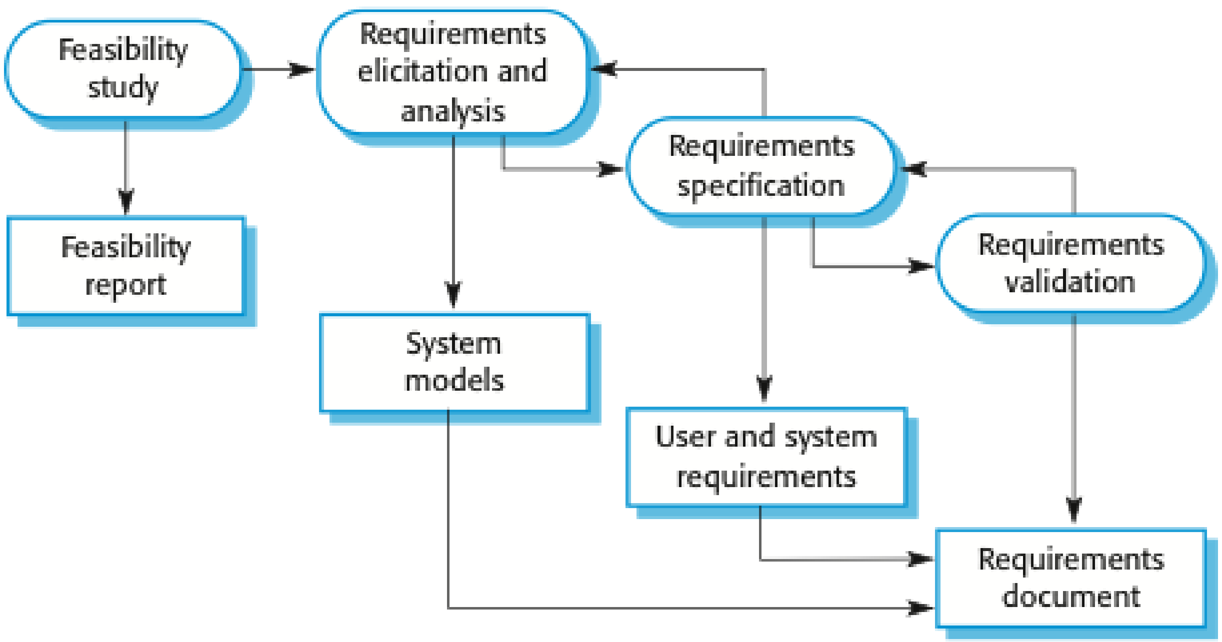
- **Requirements engineering process:**
  - **Feasibility study**
    - Is it technically and financially feasible to build the system?
  - **Requirements elicitation and analysis**
    - What do the system stakeholders (customers) require or expect from the system?
  - **Requirements specification**
    - Defining the requirements in detail
  - **Requirements validation**
    - Checking the validity of the requirements

Ch. 2 - Software Processes

3. Process Activities

3.1. Software Specification

- **Requirements Engineering process:**



3. Process Activities

3.2. Software Design and Specification

- Process of converting the system specification into an executable system
- **Software design:**
  - Design a software structure that realises the specification
    - Design diagrams, visual and/or text-based pseudo algorithms, etc.
- **Implementation:**
  - Translate this structure into an executable program
    - Writing the source code

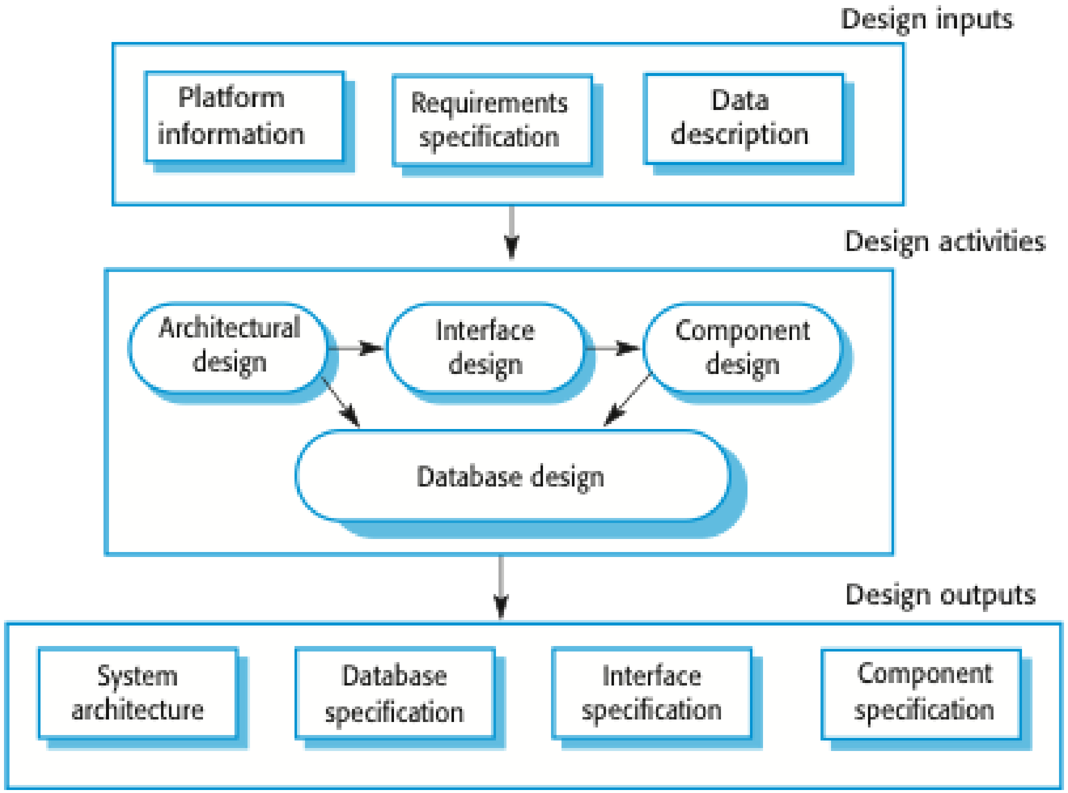
↳ The activities of design and implementation are closely related

⇨ and may be inter-leaved

3. Process Activities

3.2. Software Design and Specification

General software design process



3. Process Activities

3.2. Software Design and Specification

Software design process activities:

- **Architectural design:** Identifying overall system structure
  - Initial system components (sometimes called sub-systems or modules)
  - Their relationships and how they are distributed
- **Interface design:** Defining the interfaces between system components
  - As well as interfaces between system components and the user
- **Component design:** Defining how each system component operates
- **Database design:** Defining system data structures
  - And how these are to be represented in a database

3. Process Activities

3.3. Software Validation

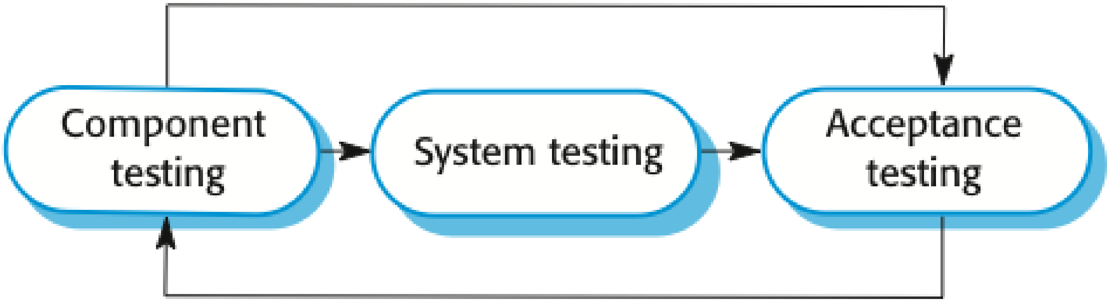
- Verification and validation (V&V):
  - Verification: intended to show that a system conforms to its specification
  - Validation: whether it meets the requirements of the customer
- Involves checking and **reviewing processes** and **system testing**
  - **Reviewing processes:** verifying process design validity
  - **System testing:** running the system with test cases
    - Derived from the specification of real data to be processed by the system
    - Simulating real data using synthetic data generators
      - Especially for large-scale data testing

Testing is the most commonly used V&V activity

3. Process Activities

3.3. Software Validation

- Different stages of testing:



3. Process Activities

3.3. Software Validation

Different stages of testing:

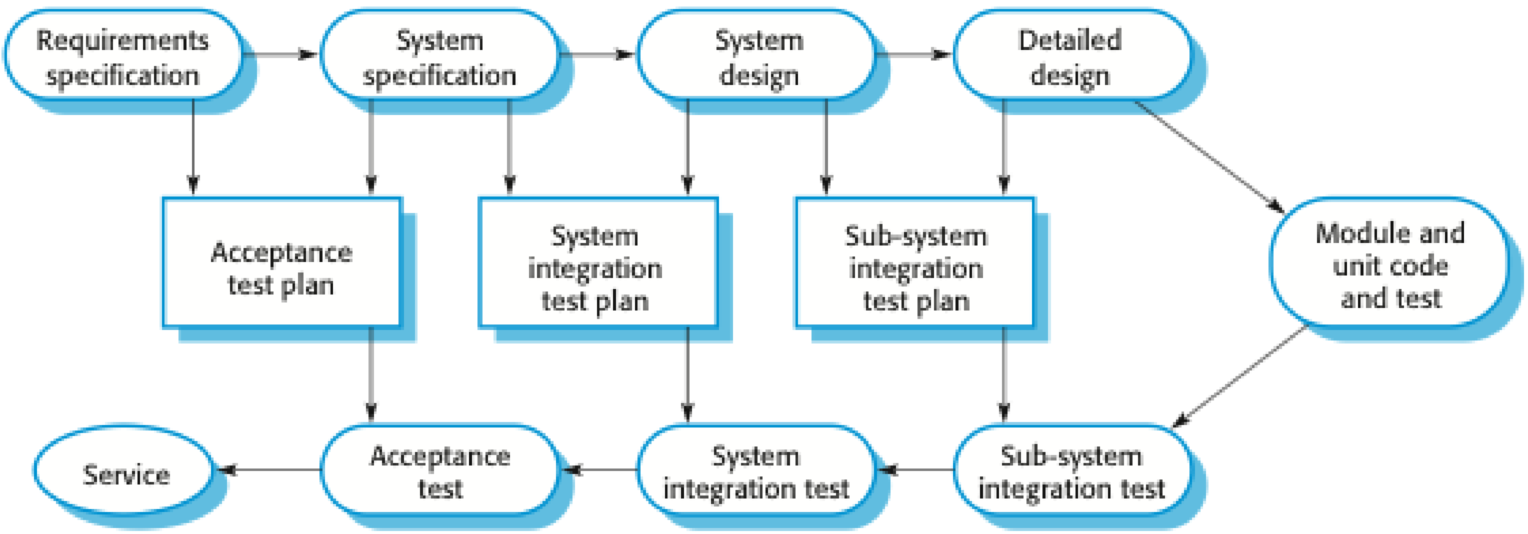
- **Component testing**
  - Individual components are tested independently
  - Components may be functions or objects or coherent groupings of both
- **System testing**
  - Testing of the system as a whole
  - Testing of emergent system properties is particularly important
- **Acceptance testing** (or **alpha** testing)
  - Testing with customer data to check that the system meets the customer's needs
- **Beta testing**
  - Delivering system to potential customers agreeing to use the system & report problems

3. Process Activities

3.3. Software Validation

Testing phases in a plan-driven process:

- **V-model:** Extension of the **waterfall model**



Ch. 2 - Software Processes

3. Process Activities

3.4. Software Evolution

- Software is inherently flexible and can change

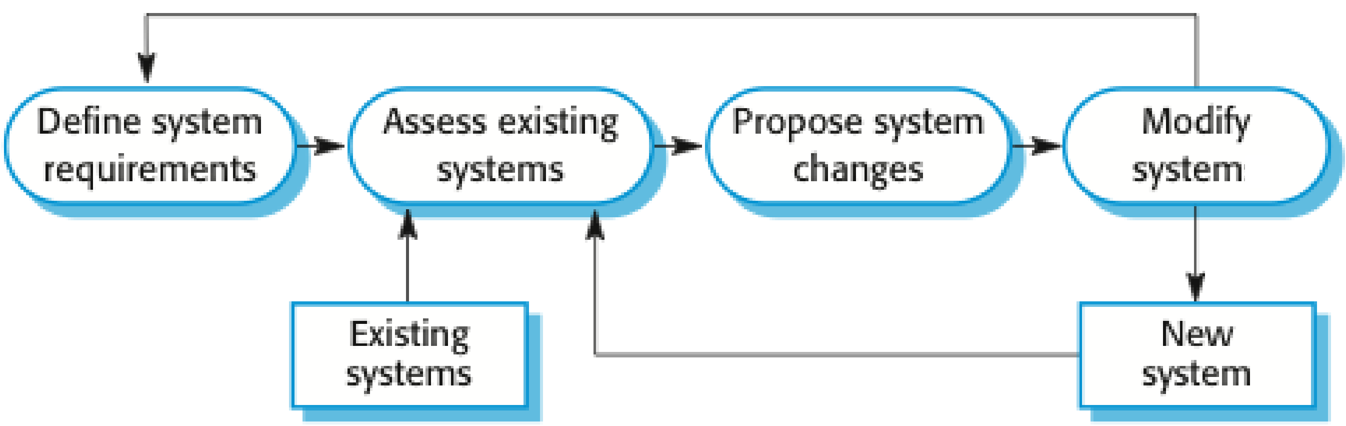
- As requirements change through changing business circumstances
  - The software that supports the business must also evolve and change
    - ⇨ **Maintainability**

- Even though **development** and **evolution** are considered separate activities
  - This demarcation is becoming increasingly irrelevant
    - ⇨ As fewer and fewer systems are completely new

3. Process Activities

3.4. Software Evolution

- **General software evolution (maintenance) process:**



  - **Extensibility:** Change existing software system to meet new requirements
    - The software must evolve to remain useful

4. Coping with Change

- Change is inevitable in all large software projects
  - **Business changes** lead to new and changed system requirements
  - **New technologies** open up new possibilities for improving implementations
  - **Changing platforms** require application changes

- Change leads to additional costs
  - The costs of change include:
    - **Rework** (e.g. re-analysing requirements, re-validation)
    - **Introducing new functionality**

4. Coping with Change

4.1. Reducing Change Costs

- **Change avoidance:** including activities that can anticipate possible changes
  - Before significant rework is required
    - For example: developing a prototype system to show key features to customers

- **Change tolerance:** accommodating changes at relatively low cost
  - This normally involves some form of incremental development
    - Proposed changes may be implemented in increments
    - Then only a single increment (a small part of the system) is altered to incorporate change

4. Coping with Change

4.2. Software Prototyping

- A prototype is an initial version of a system
  - Used to demonstrate concepts and try out design options

- A prototype can be used in:
  - The **requirements engineering process** to help with requirements elicitation and validation
  - In the **design process** to explore options and develop a UI design
    - Tools for GUI prototyping: Adobe XD (payed), Pencil Project (open source)
  - In the **testing process** to run back-to-back tests

4. Coping with Change

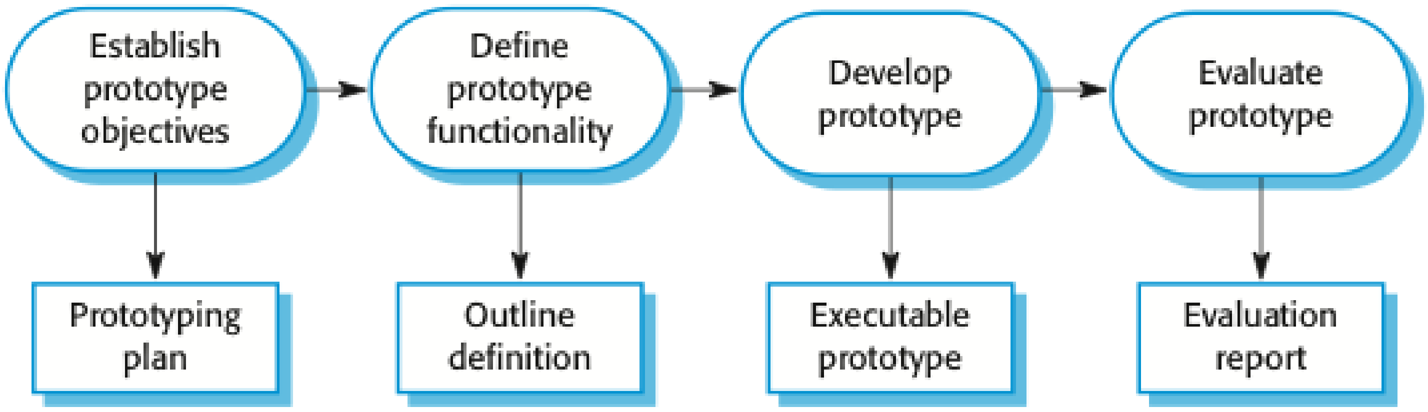
4.2. Software Prototyping

- **Main benefits of prototyping:**
  - Improved system usability
  - A closer match to users' real needs
  - Improved design quality
  - Improved maintainability
  - Reduced development effort

4. Coping with Change

4.2. Software Prototyping

Process of prototype development:



4. Coping with Change

4.2. Software Prototyping

Process of prototype development:

- May be based on rapid prototyping languages or tools
- May involve leaving out functionality:
  - Focusing on areas of the product that are **not well-understood**
  - Error checking and **recovery may not be included** in the prototype
  - **Focus on functional** rather than non-functional **requirements** (such as reliability and security)
- **Test automation tools** can also be used:
  - Special software to control the execution of prototypes and tests
    - Comparing actual outcomes with predicted outcomes

4. Coping with Change

4.2. Software Prototyping

Throw-away prototypes

- Discarded after development: not a good basis for system production
  - It may be **impossible** to tune the prototype **to meet non-functional requirements**
  - Prototypes are normally **undocumented**
  - The prototype **structure is usually degraded** through rapid change
  - The prototype probably will not meet normal organizational quality **standards**

5. Tips for Good Practice

- Try to **develop software iteratively** (incrementally) when possible
  - Plan increments based on customer priorities
  - Deliver highest priority increments first
    - ⇨ So that changes may be made without disrupting the system as a whole
- **Manage requirements**
  - Explicitly document customer requirements
  - Keep track of changes to these requirements
- **Use component-based architectures** when possible
  - Organize system architecture as a set of reusable components
    - ⇨ Easier to design, model, test, validate, maintain and extend later on

5. Tips for Good Practice

- **Model software visually**
  - Use graphical UML models to present static and dynamic views of the software
- **Verify software quality**
  - Ensure that the software meet’s organizational quality standards
- **Control changes to software**
  - Keep track and organize software changes
    - Using some kind of change management system (versioning)
      - E.g., storing different software versions, or storing changes
- **Prototyping** is essential in coping with change
  - Helps avoid poor decisions on requirements and design

Ch. 2 - Software Processes

Support Material and Exercises

- Available in the textbook - Chapter 2
- Available with solution keys on **Blackboard!**