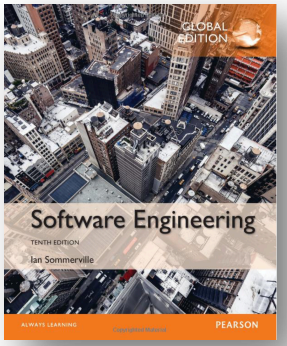
Software Processes

FEUP-MIEIC-ESOF-2019-20

Ademar Aguiar

Adapted from:

Software Engineering, Ian Sommerville, 10th Edition Chapter 2



U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

What is software engineering?

Process

- (1) The application of a **systematic, disciplined, quantifiable** approach to the development, operation, and maintenance of software; that is, the application of engineering to software.
  (2) The study of approaches as in (1).

  Source: IEEE Std 610.12: 1990 - Standard Glossary of Software Engineering Terminology

Performance measures

- Software engineering is concerned with the **cost-effective** and **timely** development of high-**quality** software in a **predictable** way, particularly for large scale systems.

The software process

- A **software process** is a structured set of activities required to develop a software system.
  - Examples: RUP, XP, Scrum, etc.
- Many different software processes but all involve the following **process activities**:
  - **Specification** – defining what the system should do;
  - **Design and implementation** – defining the organization of the system and implementing the system;
  - **Validation** – checking that it does what the customer wants;
  - **Evolution** – changing the system in response to changing customer needs.

Why do we need (defined) processes?

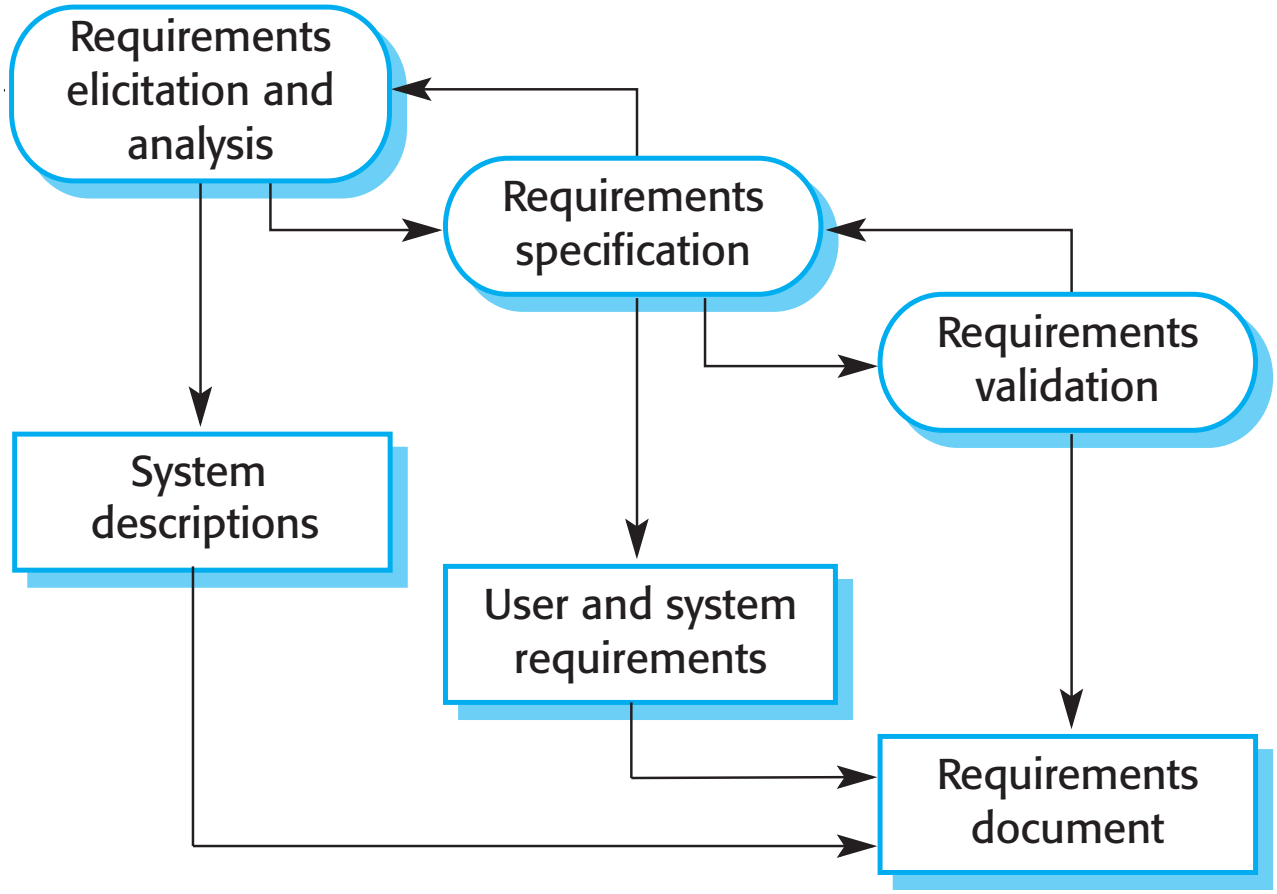
- Efficiency
  - incorporates best practices
  - structures and guides your work
  - keeps you focused on what needs to be done now
- Consistency
  - results likely to be similar
  - work likely to become predictable
- Basis for improvement
  - gathering data on your work helps determine which steps are the most time consuming, ineffective, or troublesome
  - this is useful to determine opportunities for improvement

Plan-driven and agile processes

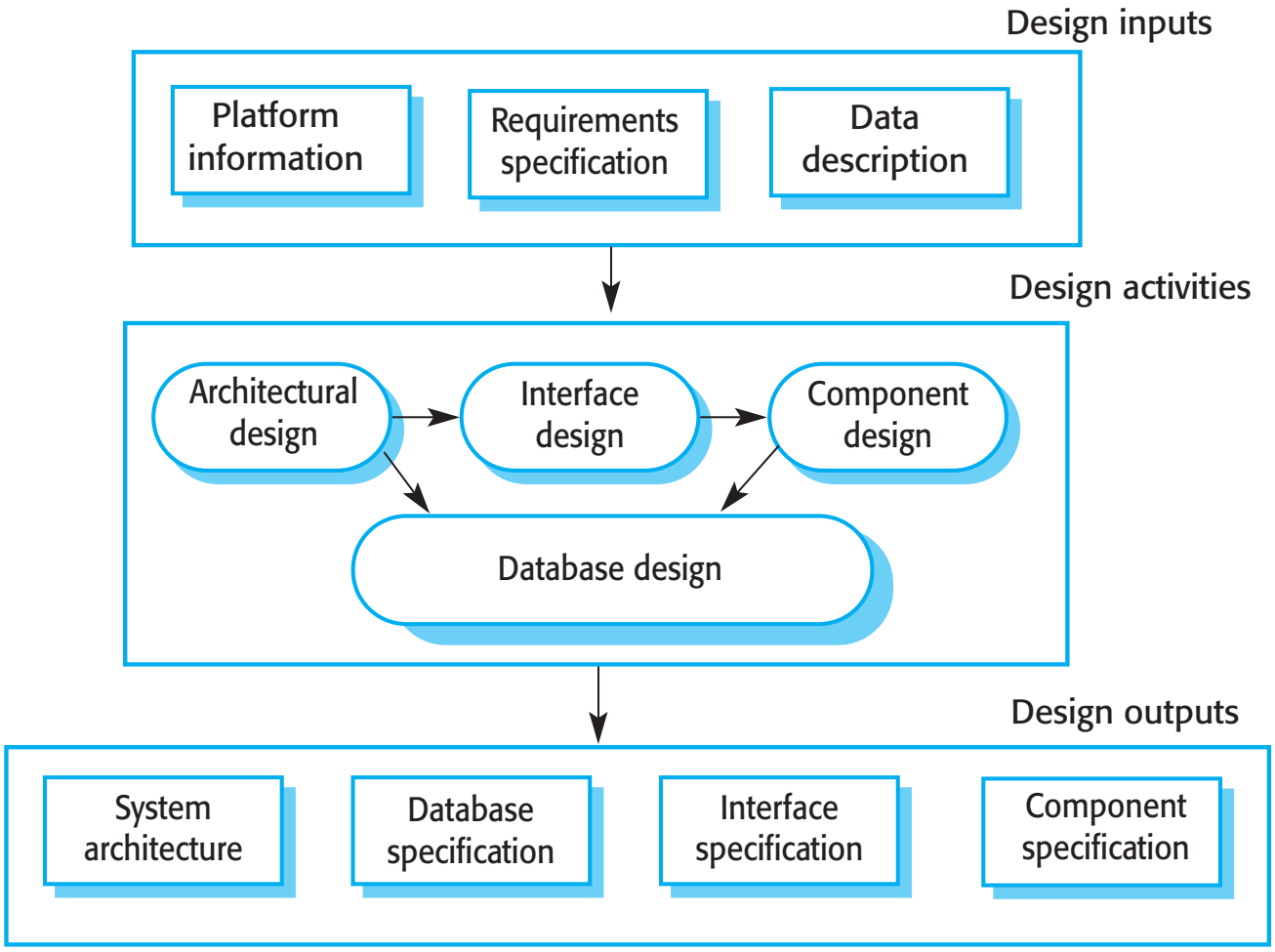
- **Plan-driven** processes are processes where **all** of the process activities are **planned in advance** and progress is measured against this plan.
- In **agile** processes, **planning is incremental** and it is easier to change the process to reflect changing customer requirements.
- In practice, most practical processes include elements of both plan-driven and agile approaches.
- There are no right or wrong software processes.

Process Activities

Requirements engineering



Software design and implementation



Software (verification &) validation

- Verification and validation (V & V) is intended to show that
  - the system conforms to its specification (verification), and
  - meets the requirements and customer needs (validation).
- Performed mainly through testing, reviews & inspections.
- Typical testing stages (or levels) are:
  - Unit (or component) testing
    - Individual components are tested usually by their developers
  - Integration testing
    - Performed as components are integrated, to find integration problems
  - System testing
    - The system as a whole is tested usually by an independent test team, with a focus on emergent properties (performance, usability, etc.).
  - Acceptance testing
    - The system is tested (under the customer responsibility) with customer data to check that customer's needs are met.

Software evolution (or maintenance)

- 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, this is increasingly irrelevant as fewer and fewer systems are completely new.
- Types of maintenance activities:
  - Corrective – bug fixing
  - Adaptive – adapt to new platforms, technologies
  - Perfective – new functionalities

Software Process Models

Software process models

- The waterfall model
  - Plan-driven model. Separate and distinct phases of specification and development.
- Incremental development (& delivery)
  - Specification, development and validation are interleaved. May be plan-driven or agile.
- Integration and configuration
  - The system is assembled from existing configurable components. May be plan-driven or agile.
- Software prototyping
  - Not actually a model but an approach to cope with uncertainty.
- In practice, most large systems are developed using a process that incorporates elements from all of these models.

Waterfall

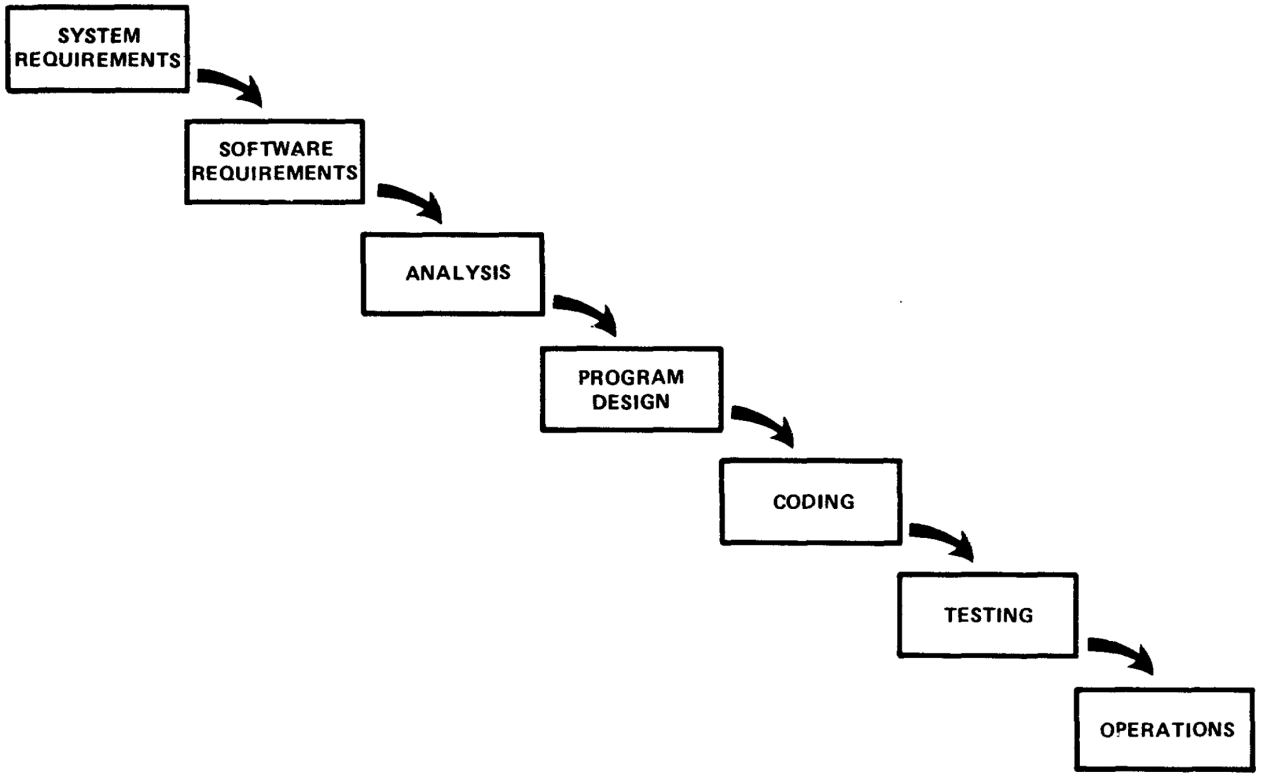
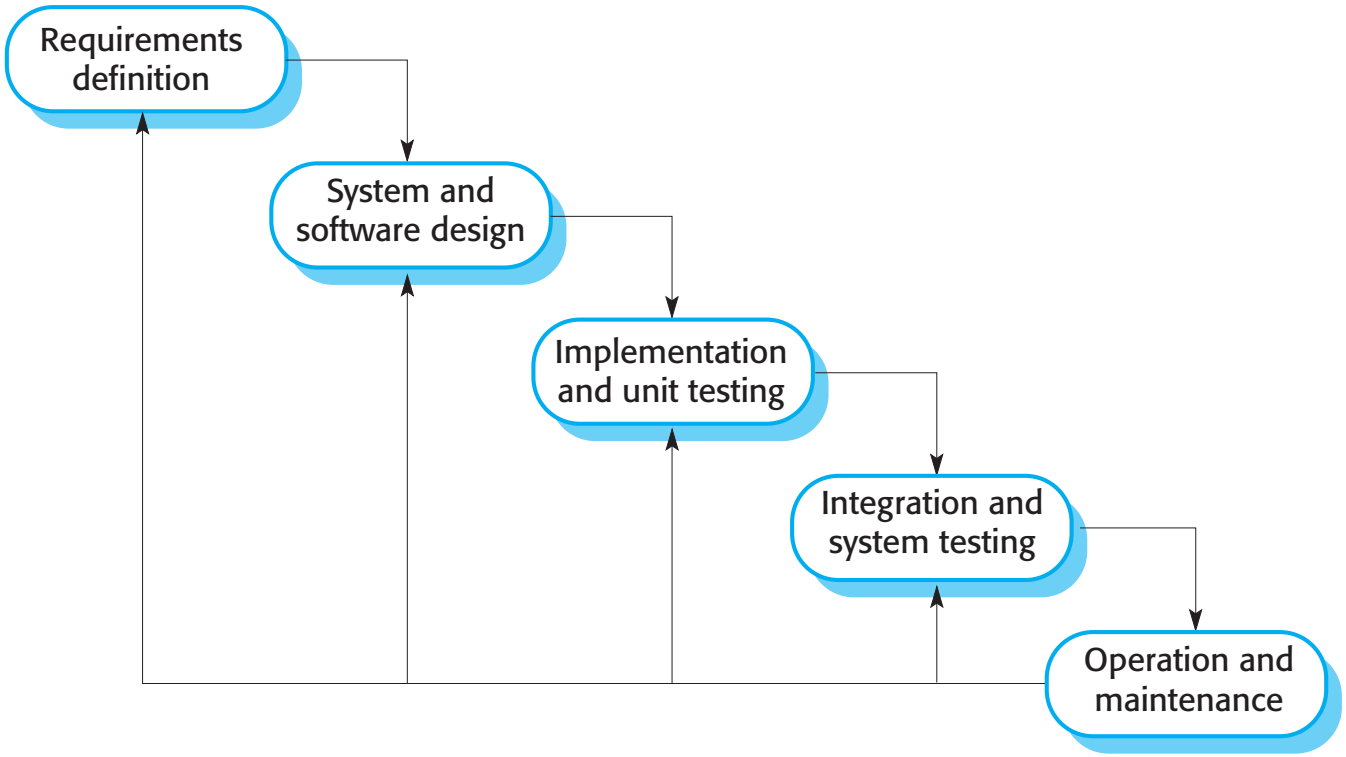


Figure 2. Implementation steps to develop a large computer program for delivery to a customer.

The waterfall model

- Plan-driven model. Separate and distinct phases of specification, development and validation.
- In principle, a phase has to be complete before moving to the next phase.



Waterfall model applicability

- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
  - Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
  - Few business systems have stable requirements.
- The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.
  - In those circumstances, the plan-driven (predictive) nature of the waterfall model helps coordinate the work.

Iterative

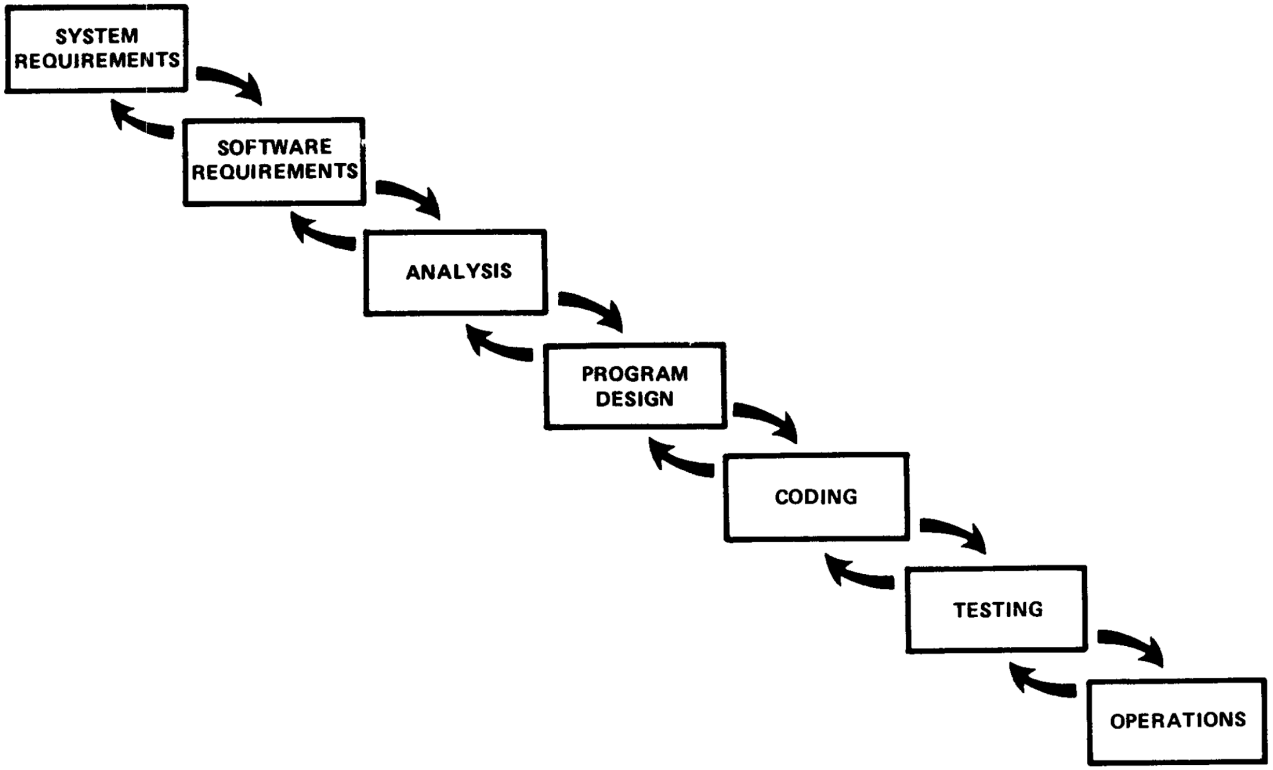


Figure 3. Hopefully, the iterative interaction between the various phases is confined to successive steps.

Spiral

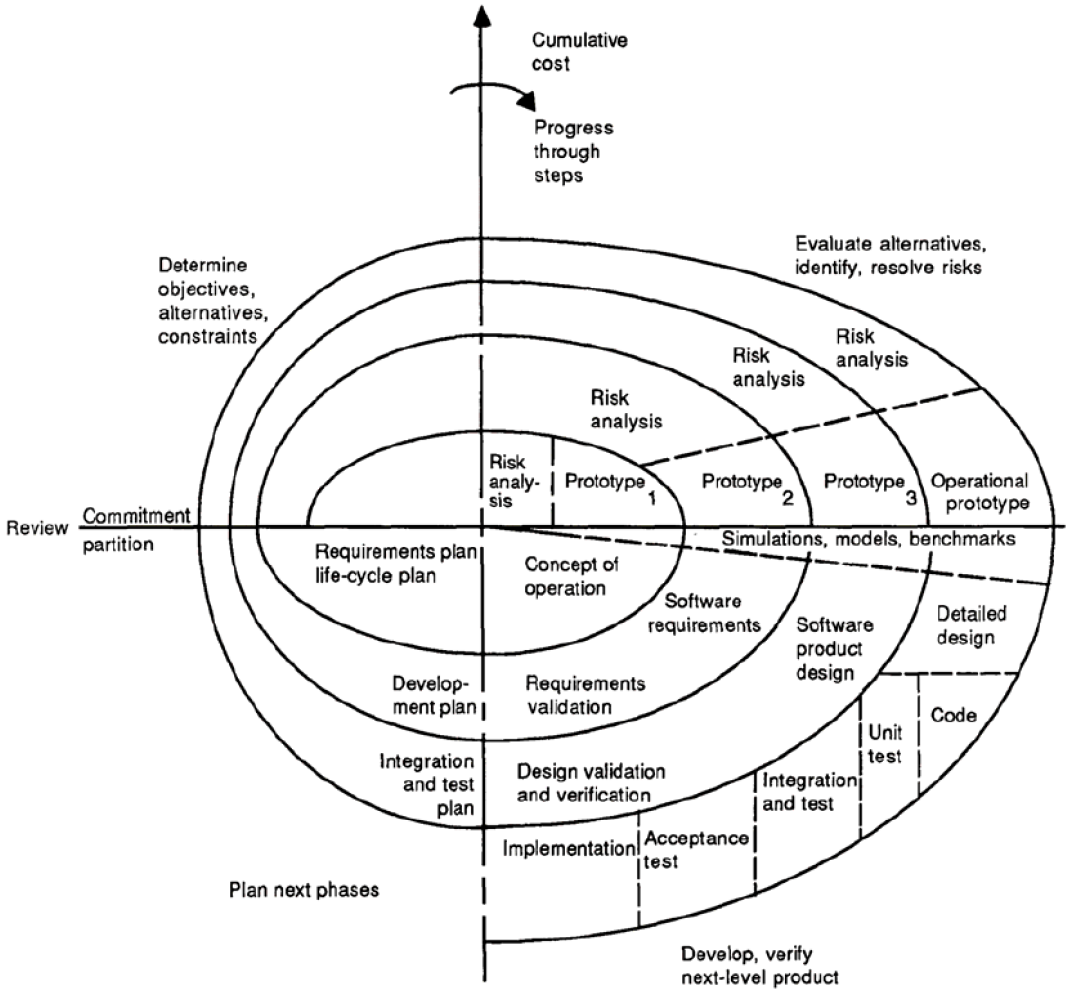
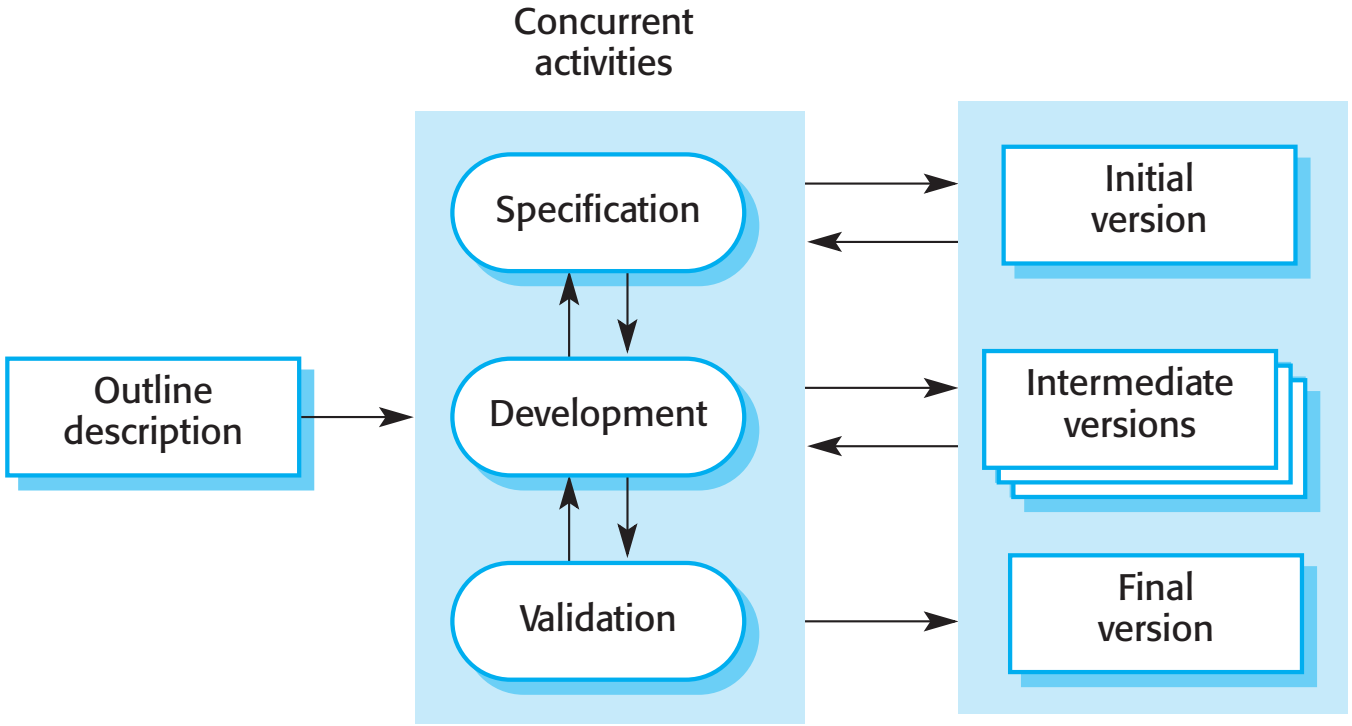


Figure 2. Spiral model of the software process.

Incremental development (& delivery)

- The system is developed in increments and each increment is evaluated (or even delivered to customers) before proceeding to the development of the next increment.
- Specification, development and validation may be interleaved.
- May be plan-driven or agile.



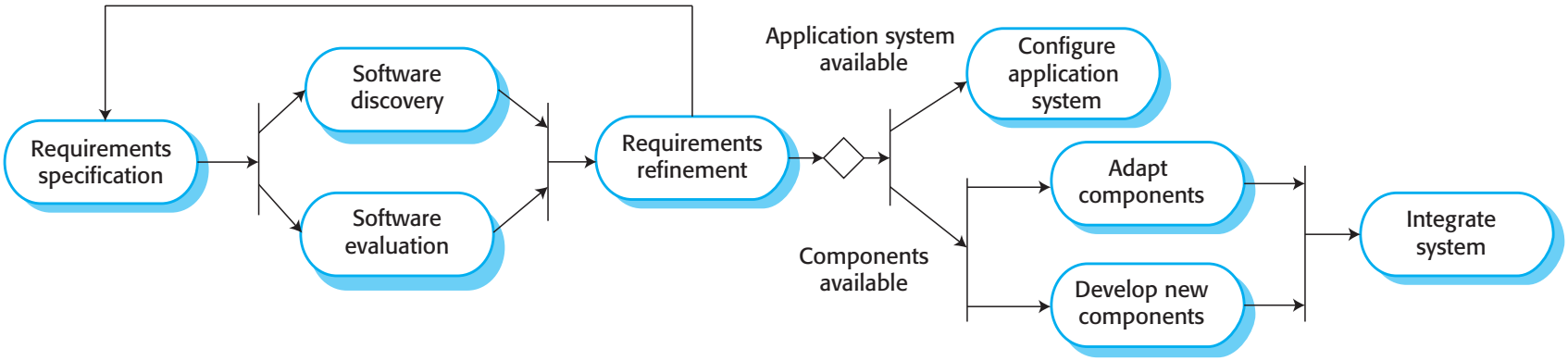
Incremental dev. (& delivery) benefits

- The cost of accommodating changing customer requirements is reduced.
  - Less documentation to change
  - Unstable requirements can be left for later stages of development
- More frequent and early customer feedback reduces risk of failure
- Customer value can be delivered with each increment so system functionality is available earlier.
- Early increments act as a prototype to help elicit requirements for later increments.

Incremental dev. (& delivery) problems

- System structure tends to degrade as new increments are added.
  - Unless time and money is spent on **refactoring** to improve the software, regular change tends to corrupt its structure and incorporating further changes becomes increasingly difficult and costly.
- It can be hard to identify upfront common facilities that are needed by all increments, so level of reuse **may be suboptimal.**
- Incremental delivery may not be possible for replacement systems as increments have less functionality than the system being replaced.
- The nature of incremental development of the specification together with the software may be not be adequate for establishing a development contract at the begin.

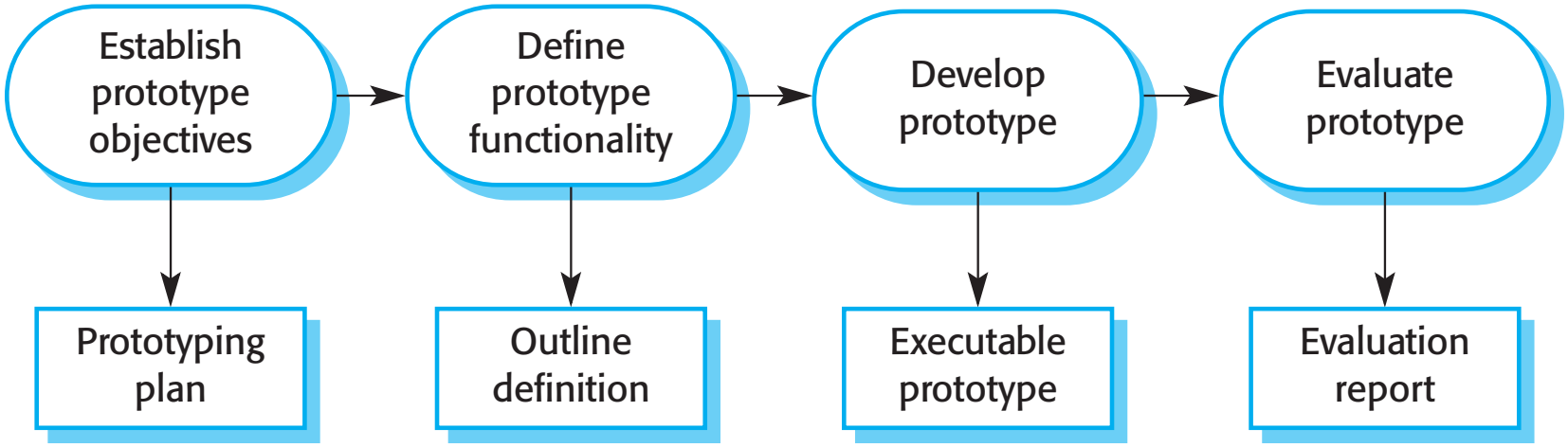
Integration and configuration



Integration and configuration: advantages and disadvantages

- Reduced costs and risks as less software is developed from scratch
- Faster delivery and deployment of system
- But requirements compromises are inevitable so system may not meet real needs of users
- Loss of control over evolution of reused system elements

Software prototyping



Key points

- There are many different software processes but all involve the following technical activities: specification, design and implementation, validation and evolution
- Generic ways of organizing the basic process activities are described by software process models.
- Examples include the ‘waterfall’ model, incremental development, and integration and configuration.
- Processes should be organized to better cope with change, by including prototyping activities, iterative development, etc.

jpf@fe.up.pt