

---

# Chapter 3

## Software Processes

# Software Processes

---

- *Software process* - a set of related activities that leads to the production of a software product.
- *Software process model* – A software process model is a simplified representation of a software process.

# Objectives

---

- To introduce software process models
- To describe a number of different process models and when they may be used
- To describe outline process models for requirements engineering, software development, testing and evolution
- To introduce CASE technology to support software process activities

# Topics covered

---

- Software process models
- Process iteration
- Software specification
- Software design and implementation
- Software validation
- Software evolution
- Automated process support

# The software process

---

- *Software process* - a set of related activities that leads to the production of a software product.
  - Specification
  - Design
  - Validation
  - Evolution
- A *software process model* is an abstract representation of a process. It presents a description of a process from some particular perspective

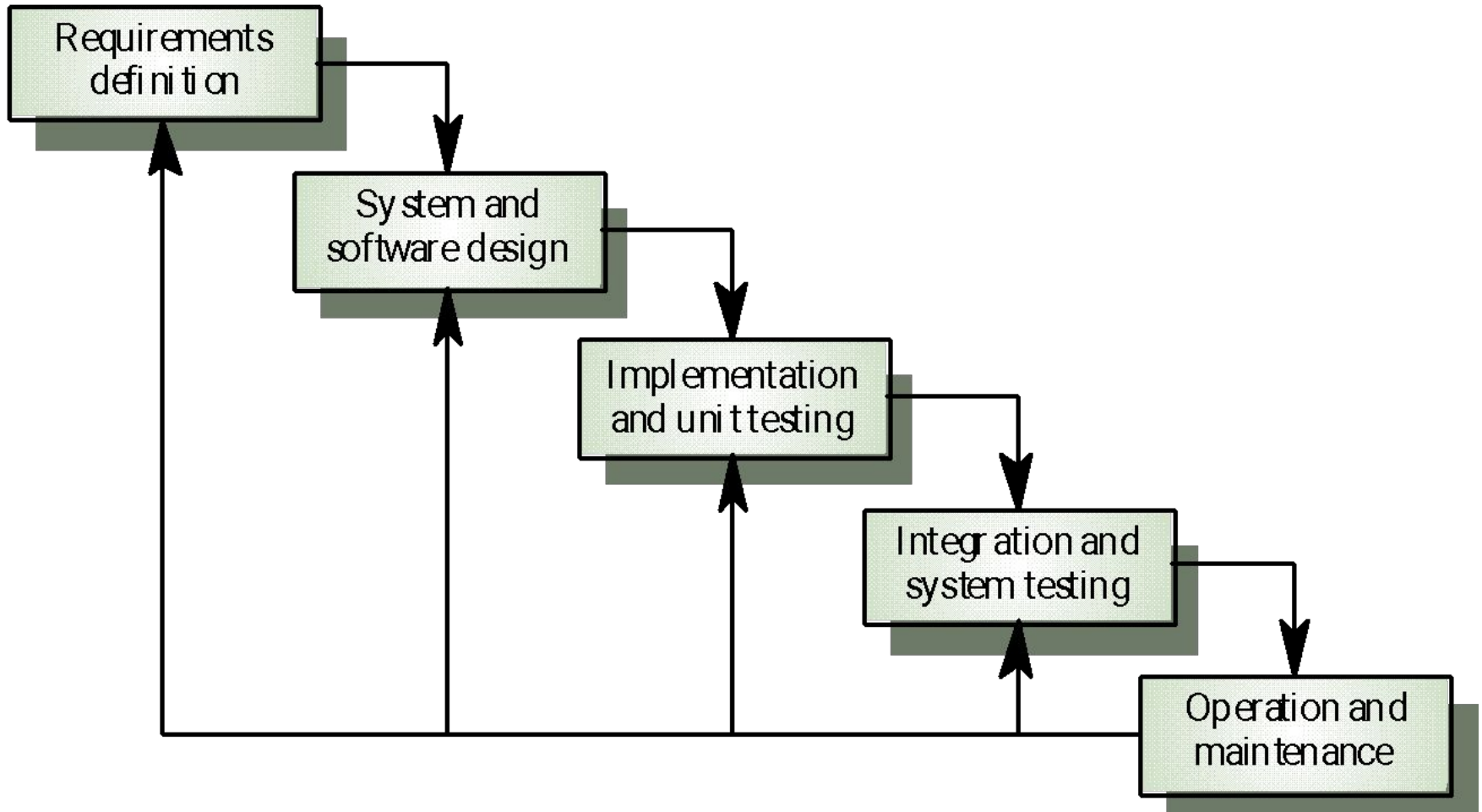
# Generic software process models

---

- The **waterfall** model
  - Separate and **distinct phases** of **specification** and **development**
- **Evolutionary** development
  - Specification and development are **interleaved**
- **Component Based**
  - The system is assembled from existing components

# Waterfall model

---



# Waterfall model phases

---

- Developed in early 1970
- It is also referred to as a **Linear-sequential model**.
- It is very simple to understand and use. In a **waterfall model**, each **phase** must be **completed before** the **next** phase can begin and there is **no overlapping** in the phases.
- The drawback of the waterfall model is the **difficulty** of accommodating **change** when the process is underway (Requirements are fixed)
- Suitable for **small or medium size project**.



# Waterfall model problems

---

- Advantages
  - Well documented
  - Process is visible
- Disadvantages
  - Premature freezing
  - Expensive
  - Inflexible
  - No Parallelism
- Therefore, this model is only appropriate when the requirements are clear and well-understood

# Evolutionary development

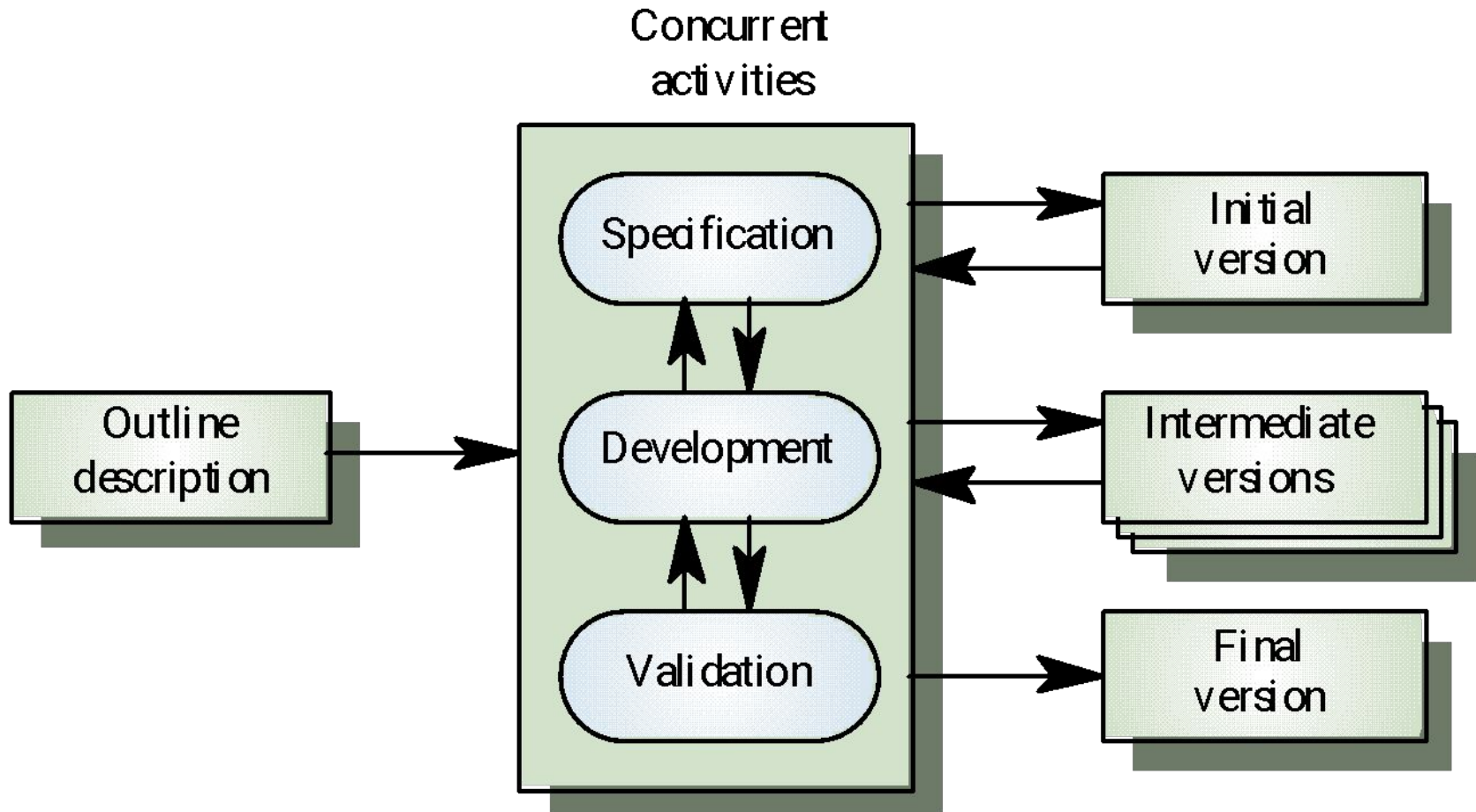
---

- Exploratory development
  - Objective is to work with customers and to **evolve** a **final system** from an **initial outline specification**. Should start with well-understood requirements
- Throw-away prototyping
  - Objective is to understand the system requirements. Should **start** with **poorly understood requirements**
  - In the evolutionary model, it is not the case. After an overall understanding of the system, the development team does not really capture all the **requirements** but **starts** with some **core** or **riskier modules**. Here the prescribed scope is not defined. Sometimes, the **project never converges** as the project manager does not know how much **time** and **effort** are required if the company signs this project or project contract for the first time.

- 
- The requirements are not clear, and it evolves during the development cycle.
  - At the initial stage, based on their understanding of the customer's requirements, they developed the core modules and functionalities and delivered them to the customer for feedback.
  - This is called an iteration. In each iteration, they release a working software after performing integration and level of testings.
  - The development team does not know how much has been completed and how much needs to be complete in the near future.
  - Picking criteria for the evolutionary model: when the requirements are not clear.

# Evolutionary development

---



# Evolutionary development

---

- Problems
  - Lack of process visibility
  - Systems are often poorly structured
  - Special skills (e.g. in languages for rapid prototyping) may be required
- Applicability
  - For small or medium-size interactive systems
  - For parts of large systems (e.g. the user interface)
  - For short-lifetime systems

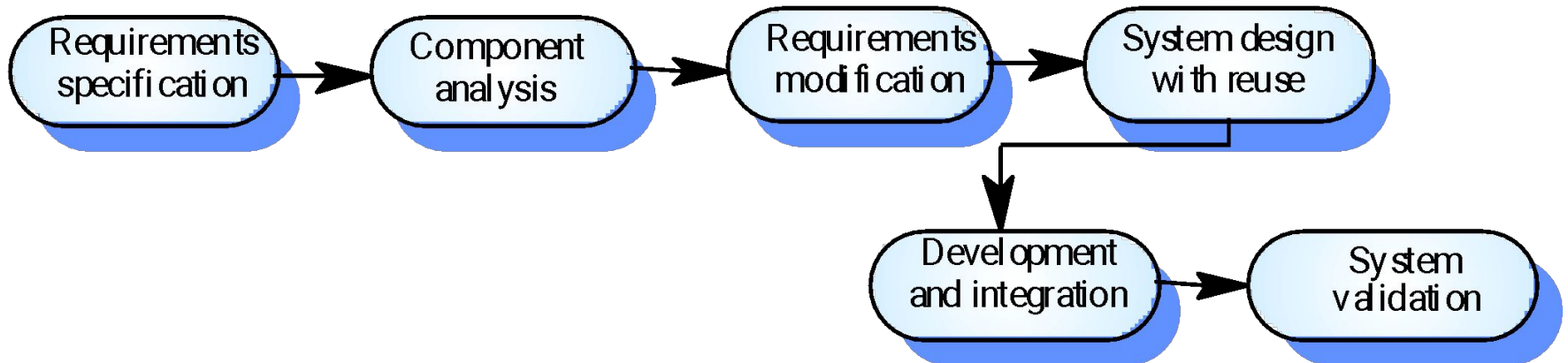
# Component Based Software Engineering

---

- Based on **systematic reuse** where **systems** are **integrated** from existing components or COTS (**Commercial-off-the-shelf**) systems
- Process stages
  - **Component** analysis
  - **Requirements** modification
  - **System design with reuse**
  - **Development** and **integration**
- This approach is becoming more important but still limited experience with it

# Component Based Software Engineering

---



# Process iteration

---

- System requirements ALWAYS evolve in the course of a project so process iteration where earlier stages are reworked is always part of the process for large systems
- Iteration can be applied to any of the generic process models
- Two (related) approaches
  - Incremental development
  - Spiral development



- 
- In an incremental model, all the requirements are gathered before starting the incremental development process.
  - The requirements are clear to the development team.
  - The requirements are split into slices, and one by one, slices are picked based on selection criteria.
  - The development of each slice goes into multiple iterations for the refinement of functionality, and in the end, a deliverable product is released.

- 
- It is called an increment, and in each increment, a deliverable product is given to the user.
  - In this model, requirements do change. But to a certain extent, like the addition and removal of certain functionalities.
  - The development team knows how much has been completed and how much needs to be complete soon.

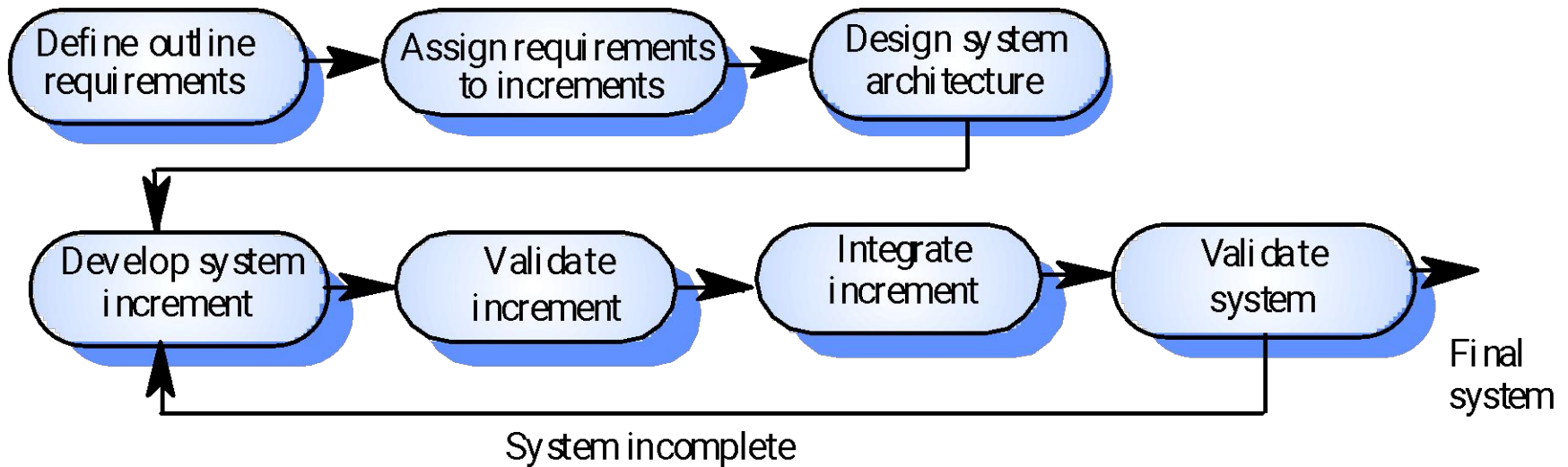
# Incremental development

---

- Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality
- User requirements are prioritised and the highest priority requirements are included in early increments
- Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve

# Incremental development

---



# Incremental development advantages

---

- Customer value can be delivered with each increment so system functionality is available earlier
- Early increments act as a prototype to help elicitate requirements for later increments
- Lower risk of overall project failure
- The highest priority system services tend to receive the most testing

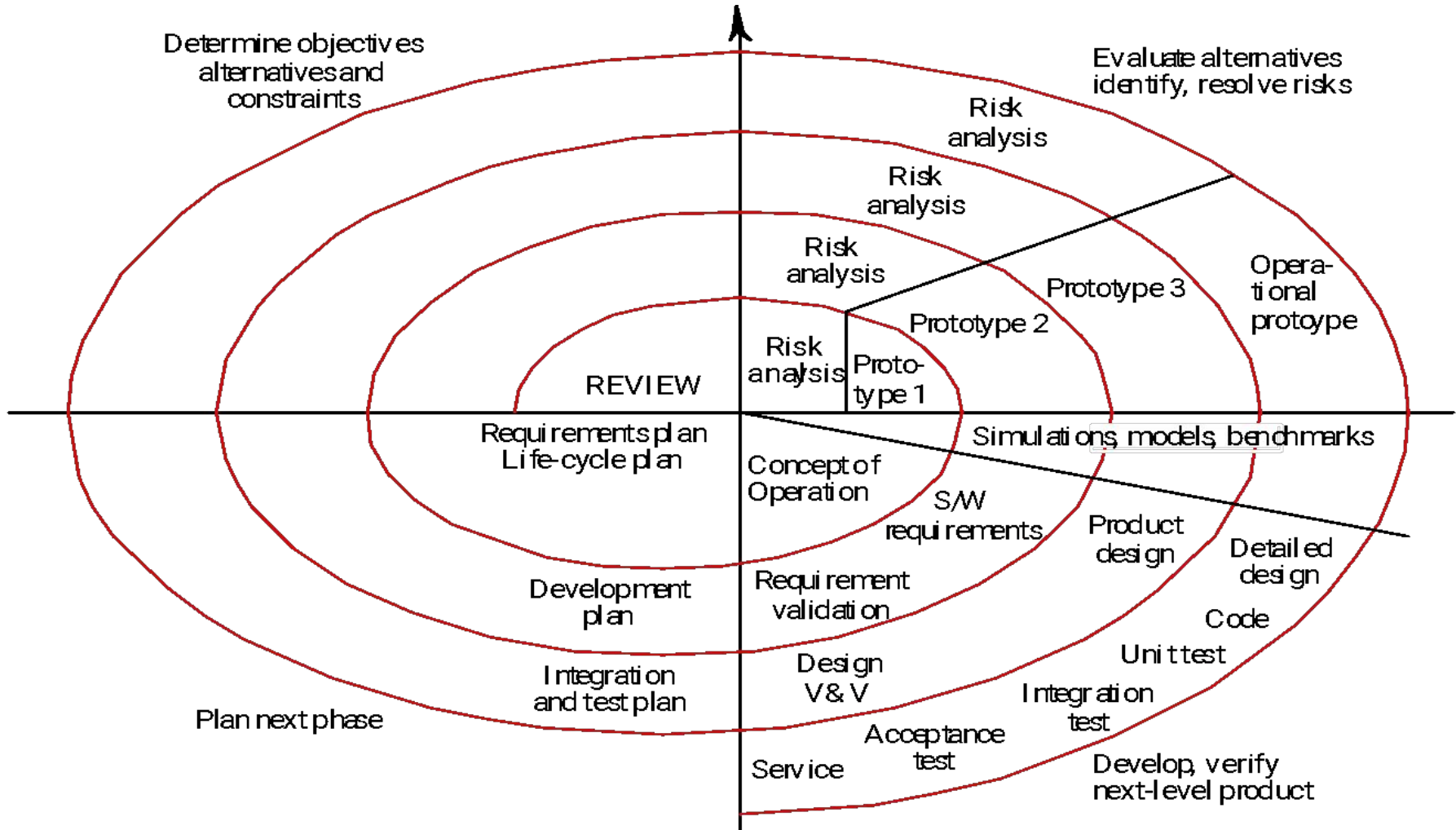
# Spiral development

---

- 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
- The spiral model is a risk-driven software development process model.

- 
- The spiral model is **a risk-driven software development process model**. Based on the unique risk patterns of a given project, the spiral model guides a team to adopt elements of one or more process models, such as incremental, waterfall, or evolutionary prototyping.

# Spiral model of the software process





# 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 which can be any of the generic models
- Planning
  - The project is reviewed and the next phase of the spiral is planned

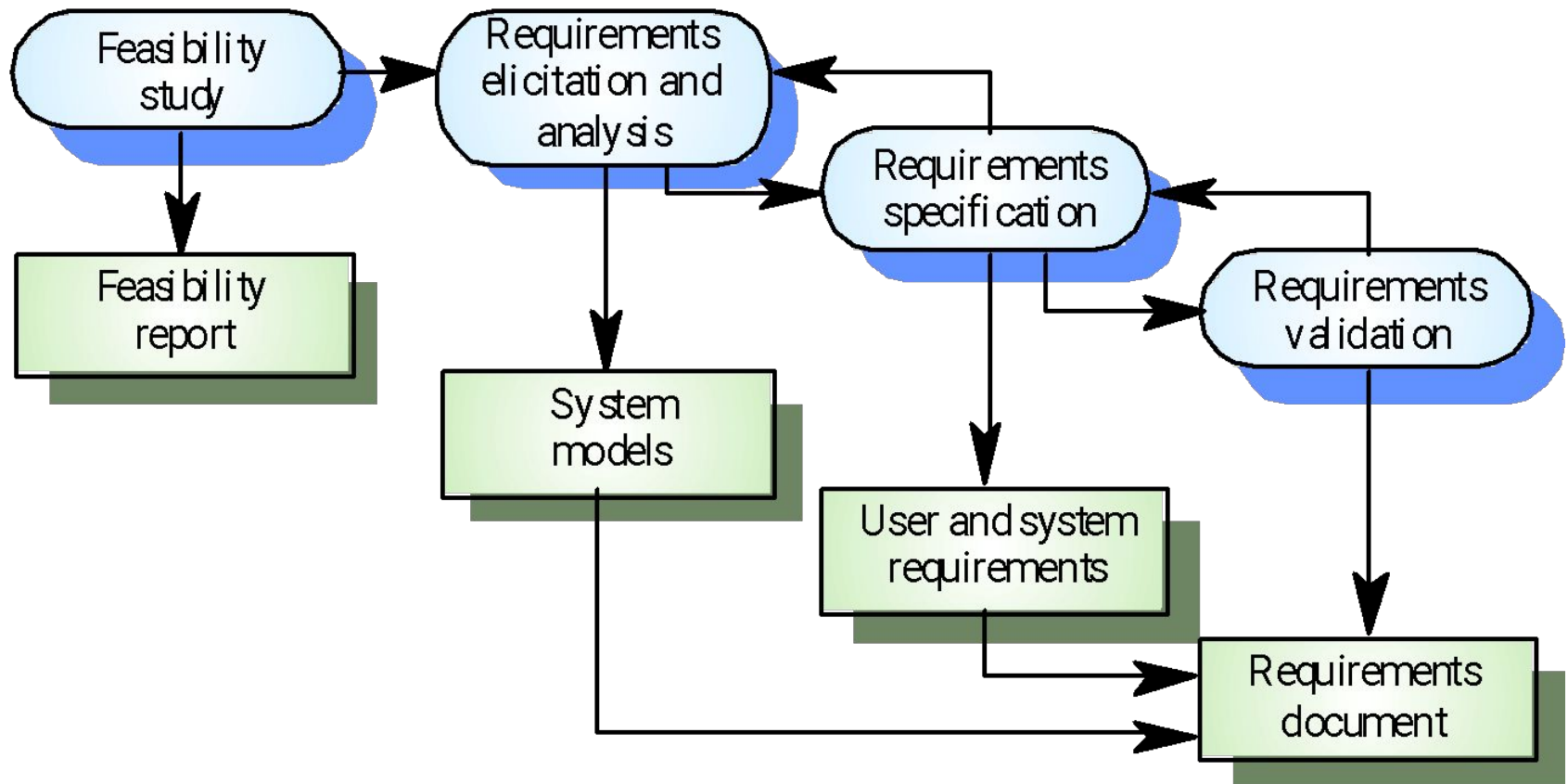
# Software specification

---

- 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

---

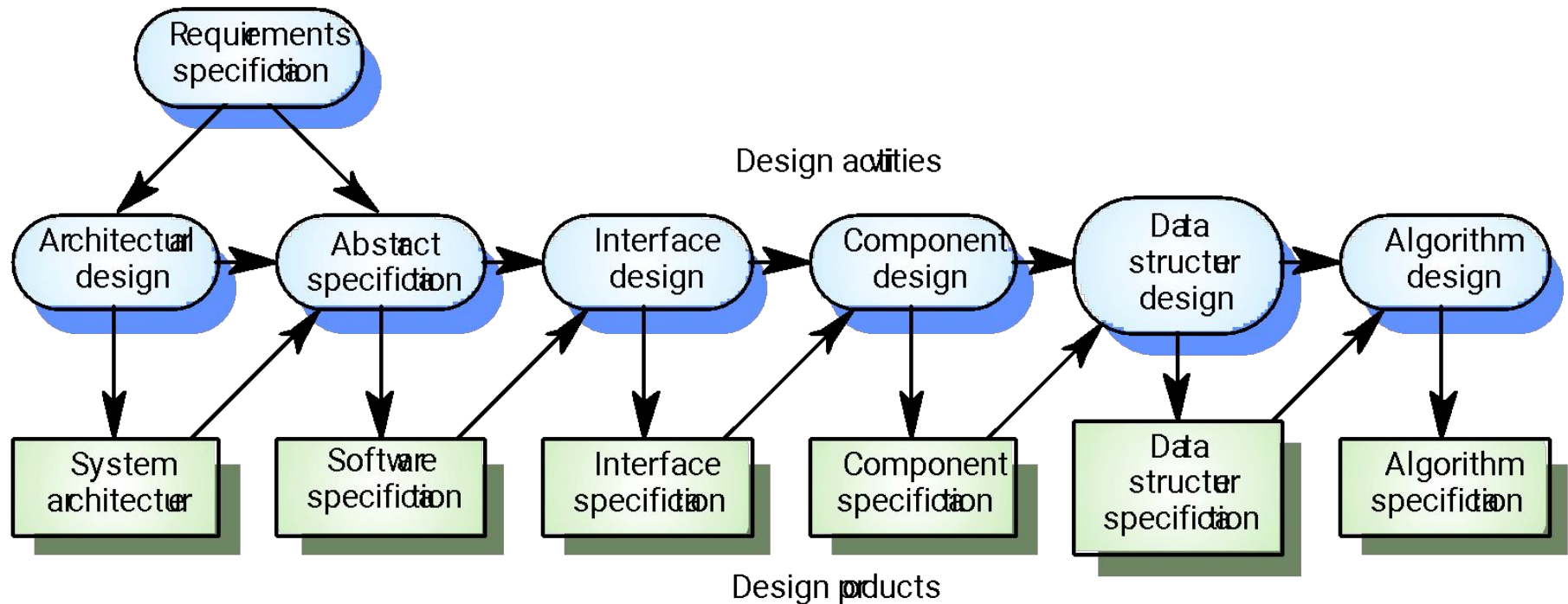
- The process of converting the **system specification** into an **executable system**
- Software design
  - Design a software structure that realises the specification
- Implementation
  - Translate this structure into an executable program
- The activities of design and implementation are closely related and may be inter-leaved

# Design process activities

---

- Architectural design
- Abstract specification
- Interface design
- Component design
- Data structure design
- Algorithm design

# The software design process



# Design methods

---

- Systematic approaches to developing a software design
- The design is usually documented as a set of graphical models
- Possible models
  - Object models
  - Sequence model
  - State transition model
  - Structural model

# Programming and debugging

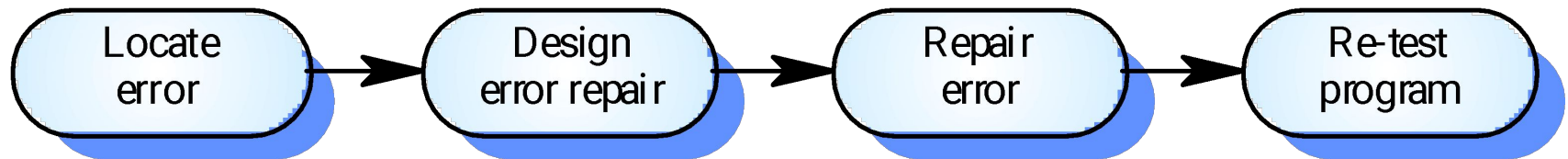
---

- Translating a design into a program and removing errors from that program
- Programming is a personal activity - there is no generic programming process
- Programmers carry out some program testing to discover faults in the program and remove these faults in the debugging process



# The debugging process

---



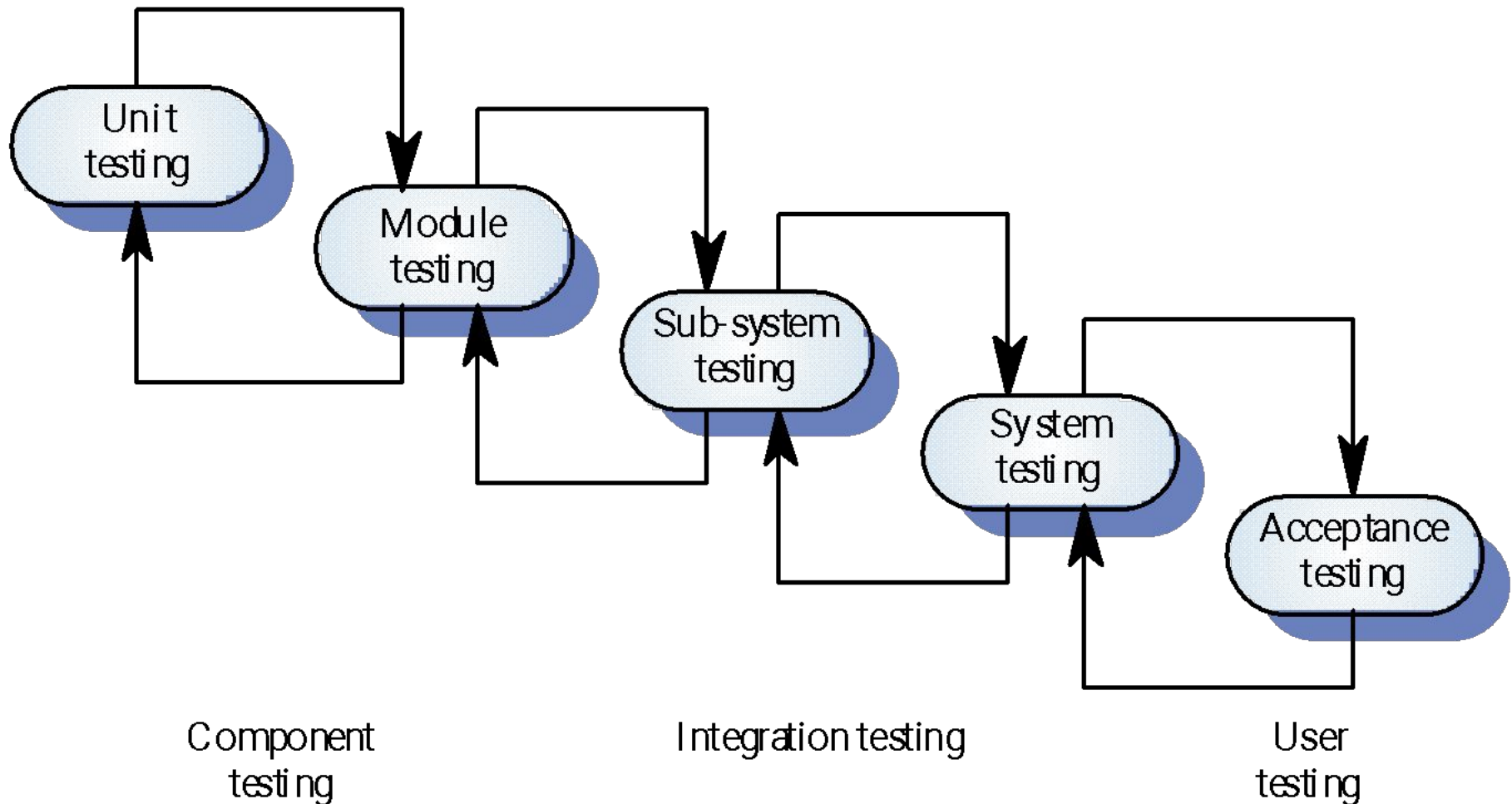
# Software validation

---

- Verification and validation is intended to show that a system conforms to its specification and meets the requirements of the system customer
- 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

---



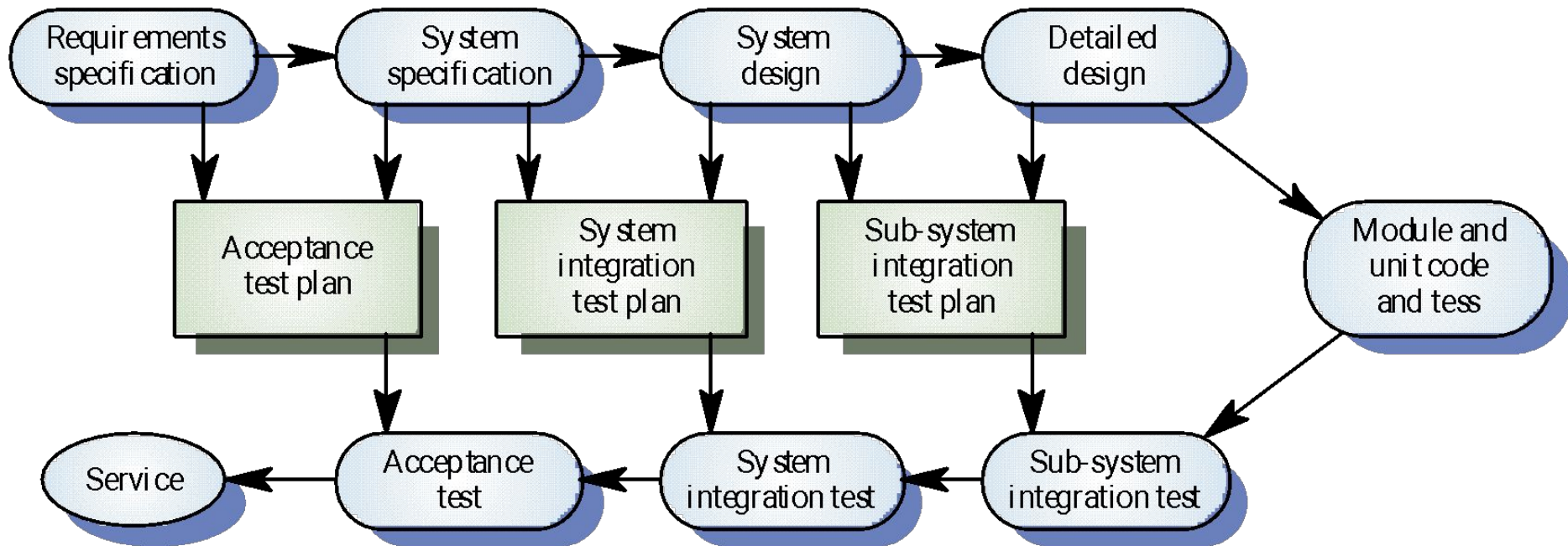
# Testing stages

---

- Unit testing
  - Individual components are tested
- Module testing
  - Related collections of dependent components are tested
- Sub-system testing
  - 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 phases

---



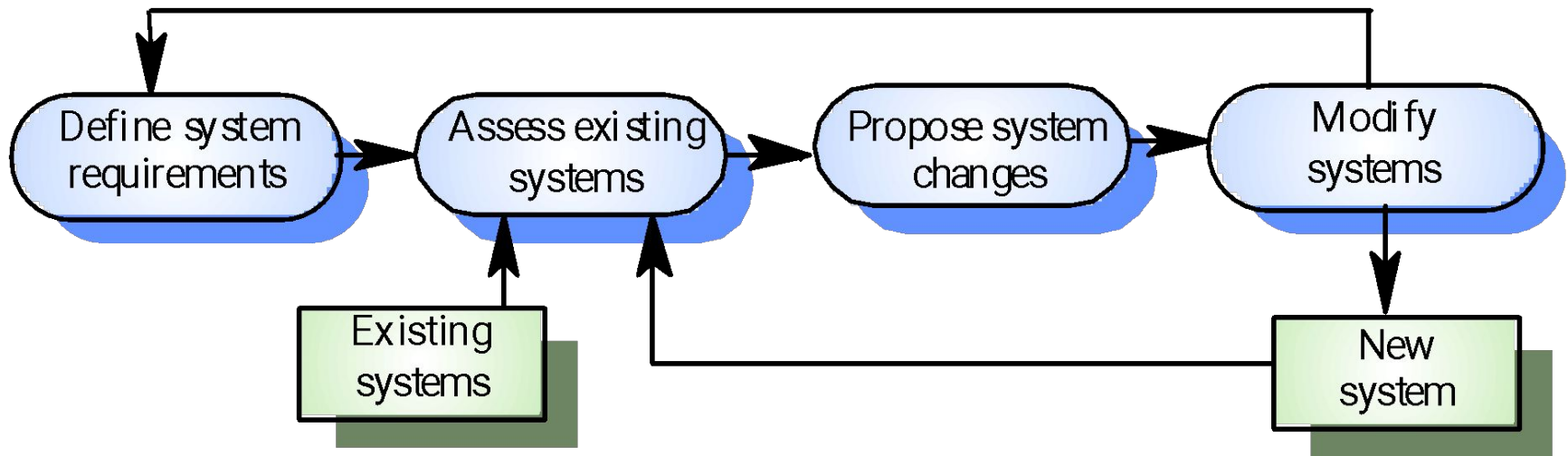
# Software evolution

---

- 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

# System evolution

---



# Automated process support (CASE)

---

- Computer-aided software engineering (CASE) is software to support software development and evolution processes
- Activity automation
  - Graphical editors for system model development
  - Data dictionary to manage design entities
  - Graphical UI builder for user interface construction
  - Debuggers to support program fault finding
  - Automated translators to generate new versions of a program



# Case technology

---

- Case technology has led to significant improvements in the software process though not the order of magnitude improvements that were once predicted
  - Software engineering requires creative thought - this is not readily automatable
  - Software engineering is a team activity and, for large projects, much time is spent in team interactions. CASE technology does not really support these

# CASE classification

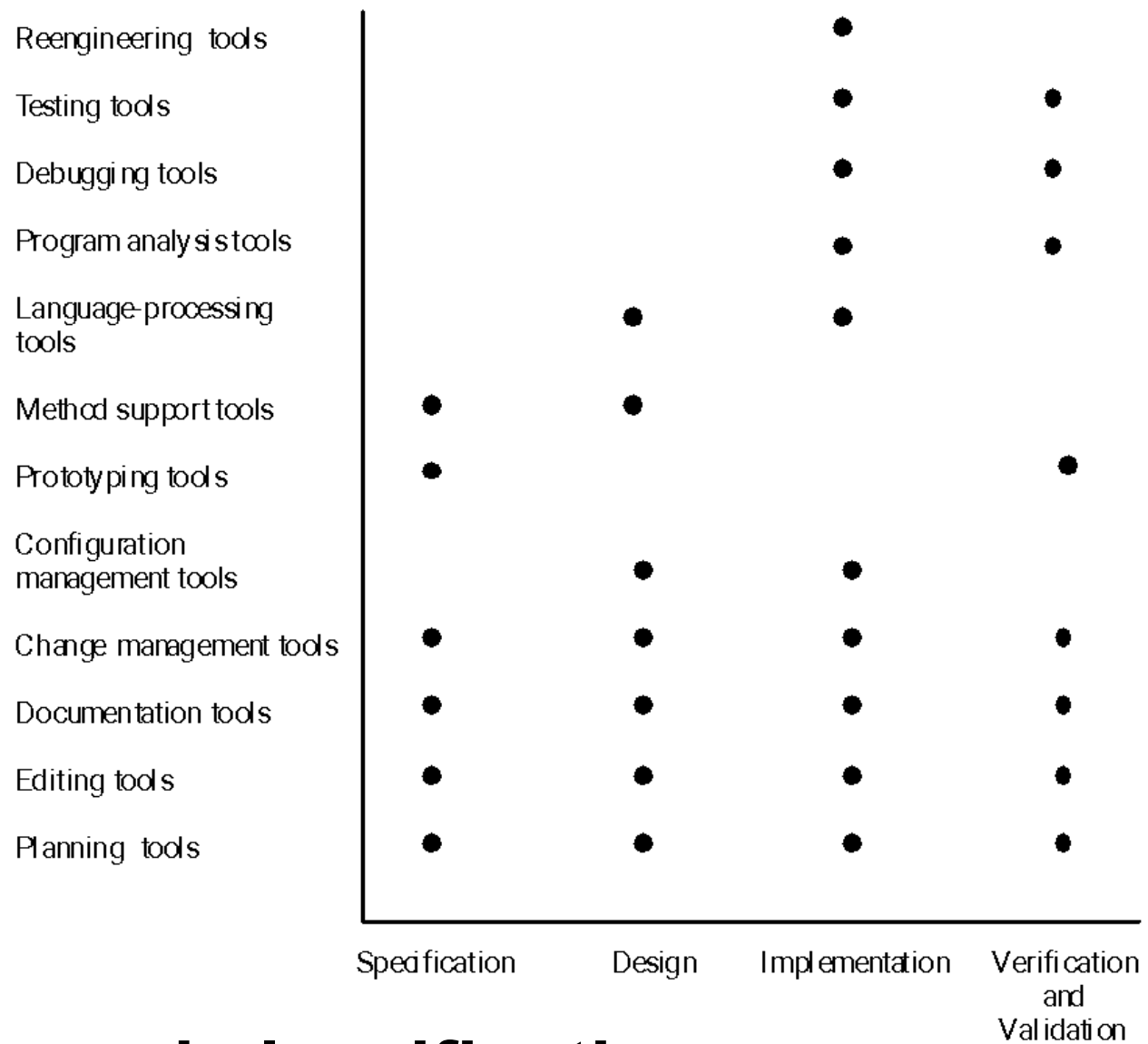
---

- Classification helps us understand the different types of CASE tools and their support for process activities
- **Functional** perspective
  - Tools are classified according to their specific function
- **Process** perspective
  - Tools are classified according to process activities that are supported
- **Integration** perspective
  - Tools are classified according to their **organisation** into **integrated** units

# Functional tool classification

---

Tool type	Examples
Planning tools	PERT tools, estimation tools, spreadsheets
Editing tools	Text editors, diagram editors, word processors
Changemanagement tools	Requirements traceability tools, change control systems
Configuration management tools	Version management systems, system building tools
Prototyping tools	Very high-level languages, user interface generators
Method-support tools	Design editors, data dictionaries, code generators
Language-processing tools	Compilers, interpreters
Program analysis tools	Cross reference generators, static analysers, dynamic analysers
Testing tools	Test data generators, file comparators
Debugging tools	Interactive debugging systems
Documentation tools	Page layout programs, image editors
Re-engineering tools	Cross-reference systems, program restructuring systems



# Activity-based classification

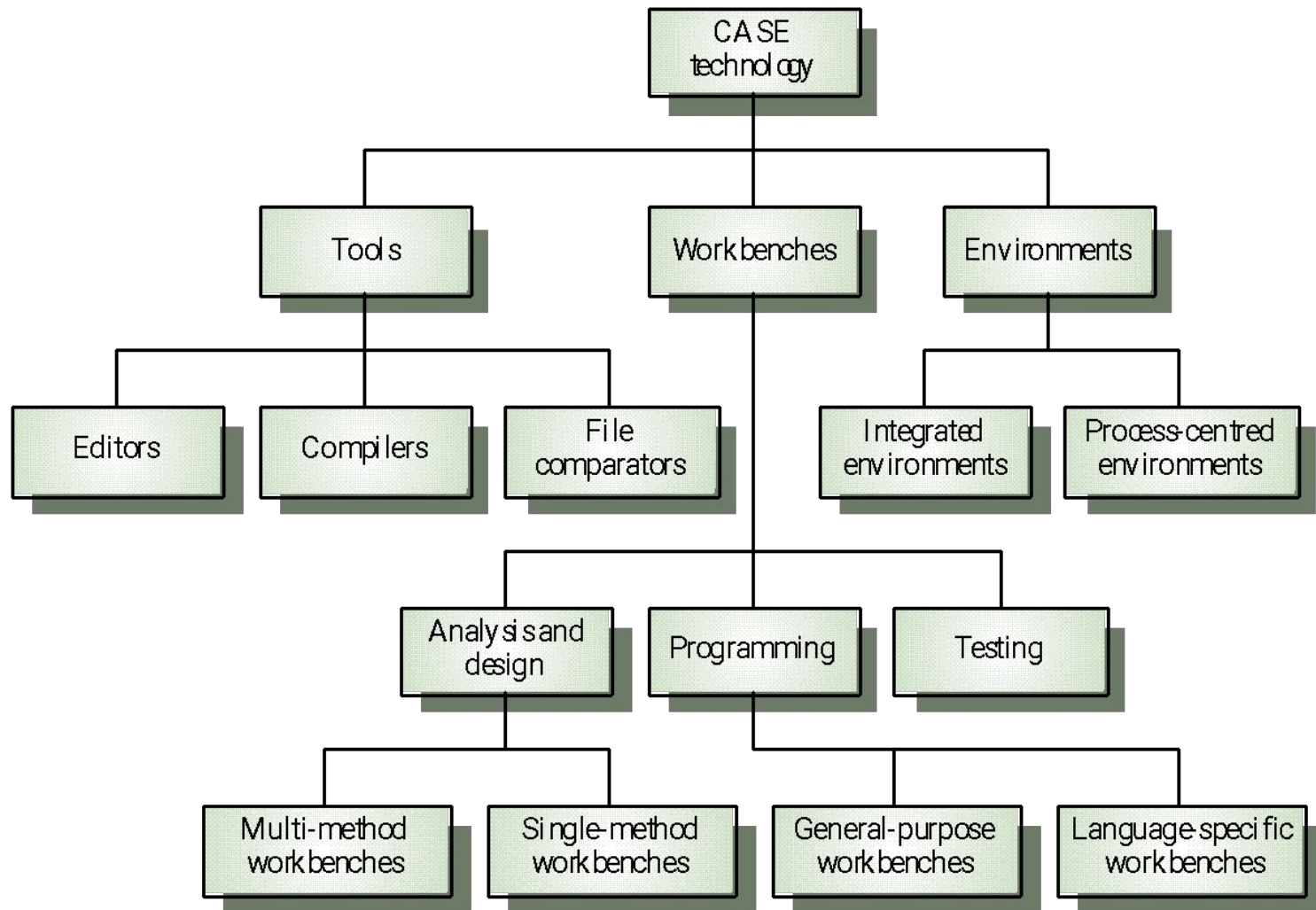
# CASE integration

---

- **Tools**
  - Support individual process tasks such as design consistency checking, text editing, etc.
- **Workbenches**
  - Support a process phase such as specification or design, Normally include a number of integrated tools
- **Environments**
  - Support all or a substantial part of an entire software process. Normally include several integrated workbenches

# Tools, workbenches, environments

---



# Key points

---

- Software processes are the activities involved in producing and evolving a software system. They are represented in a software process model
- General activities are specification, design and implementation, validation and evolution
- Generic process models describe the organisation of software processes
- Iterative process models describe the software process as a cycle of activities

# Key points

---

- 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 to its specification and user needs
- Evolution is concerned with modifying the system after it is in use
- CASE technology supports software process activities