CSE344
Software Engineering (SWE)

Week – 2
SW Processes

What is a SW process?

- A set of activities to develop/evolve SW.
- *Generic activities* in all SW processes are:
  - ***Specification*** - ***what*** the system should do and its development constraints
  - ***Development*** – ***how*** to produce the SW system
  - ***Validation & Verification (V&V)*** - checking that
    - the SW product is what the customer really wants, and
    - the SW product does what it is supposed to.
    - two questions symbolizing V&V:
      - ***Am I doing the right product?*** ↔ ***Validation***
      - ***Am I doing the product right?*** ↔ ***Verification***
  - ***Evolution*** - changing the software in response to changing demands.

SW Process Model...

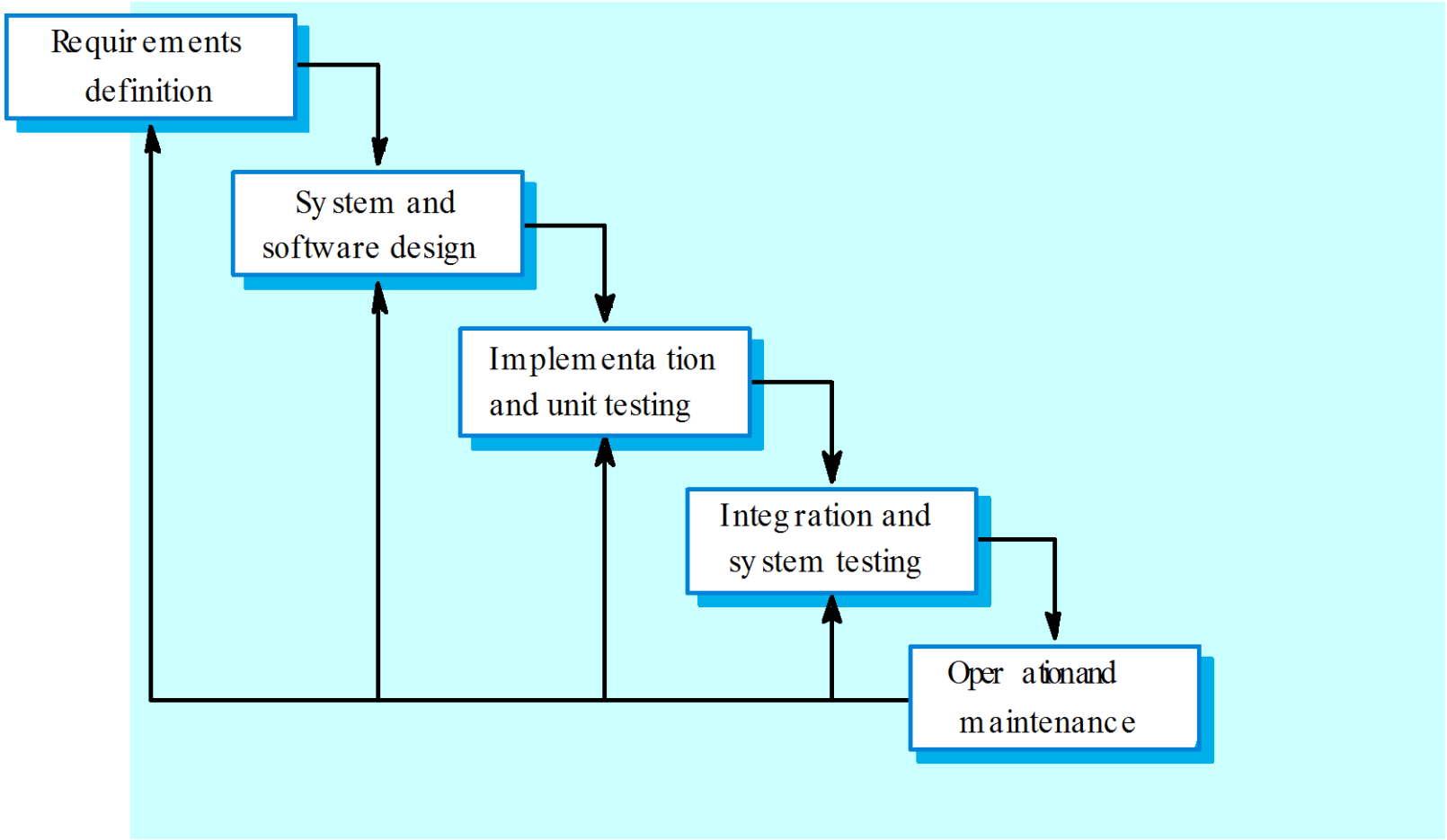
Definition:

A ***SW process model*** is an abstract representation of a process. It presents a description of a process from some particular perspective.

Generic SW Process models

- Waterfall:
  - Separate and distinct phases of specification and development.
- Evolutionary development
  - Specification, development and validation are interleaved.
- Component-based software engineering
  - The system is assembled from existing components.
- Variants of these models e.g., formal development:
  - a waterfall-like process is used
  - a formal specification refined through several stages to an implementable design.

Waterfall model



Waterfall model phases

- Requirements analysis and definition
- System and software design
- Implementation and unit testing
- Integration and system testing
- Operation and maintenance
- Main drawback:
  - Inflexible to change after the process is underway. One phase has to be complete before moving onto the next phase.

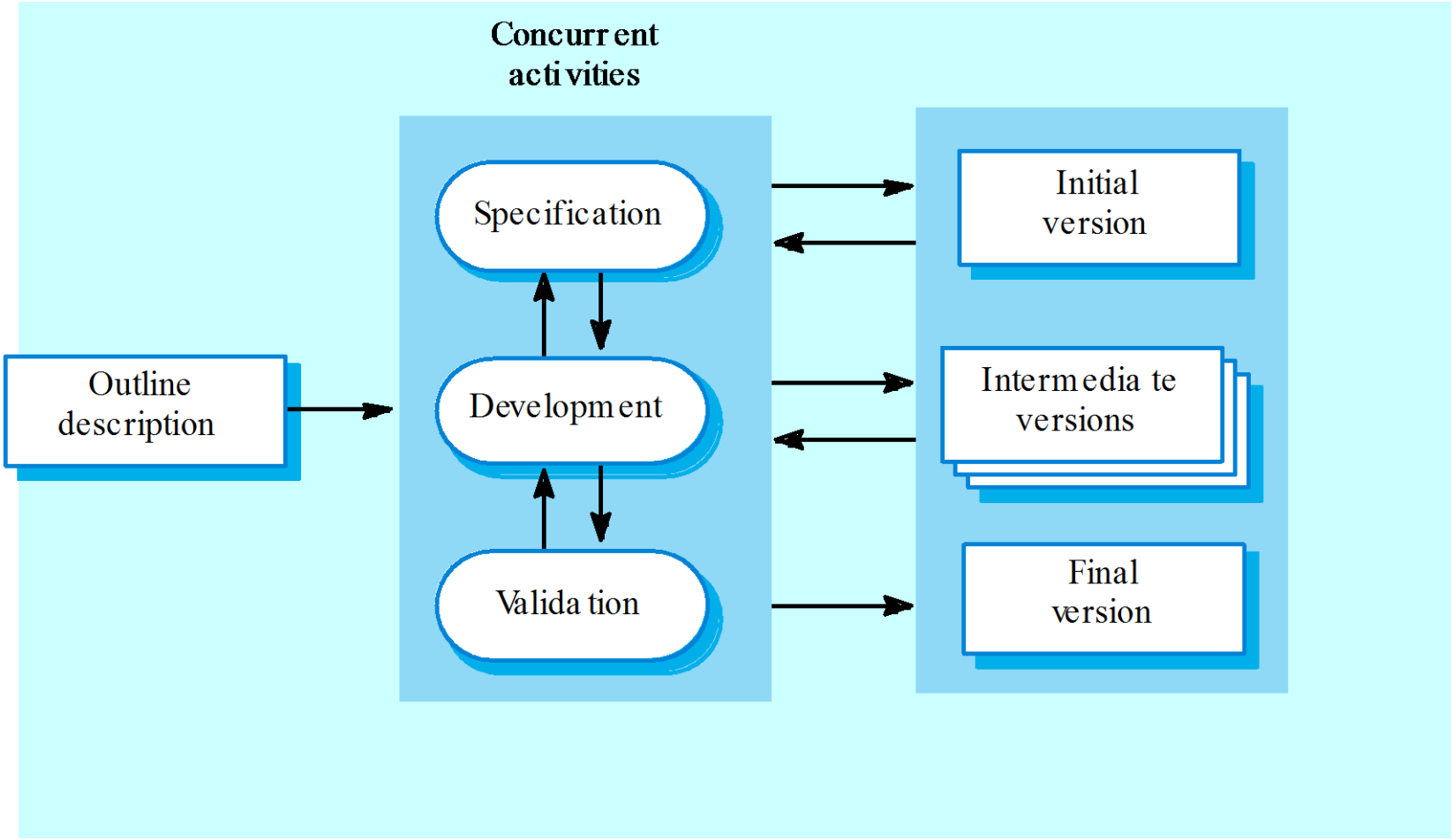
Waterfall model problems

- Inflexible partitioning of process into stages: Hard to handle changing customer requirements.
- Hence, waterfall is appropriate for projects with stable and well-understood requirements and hence, with limited changes 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.

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* and add new features as proposed by the customer.
- Throw-away prototyping
  - Objective is to *understand the system requirements*. *Should start with poorly understood requirements* to clarify what is really needed.

Evolutionary development



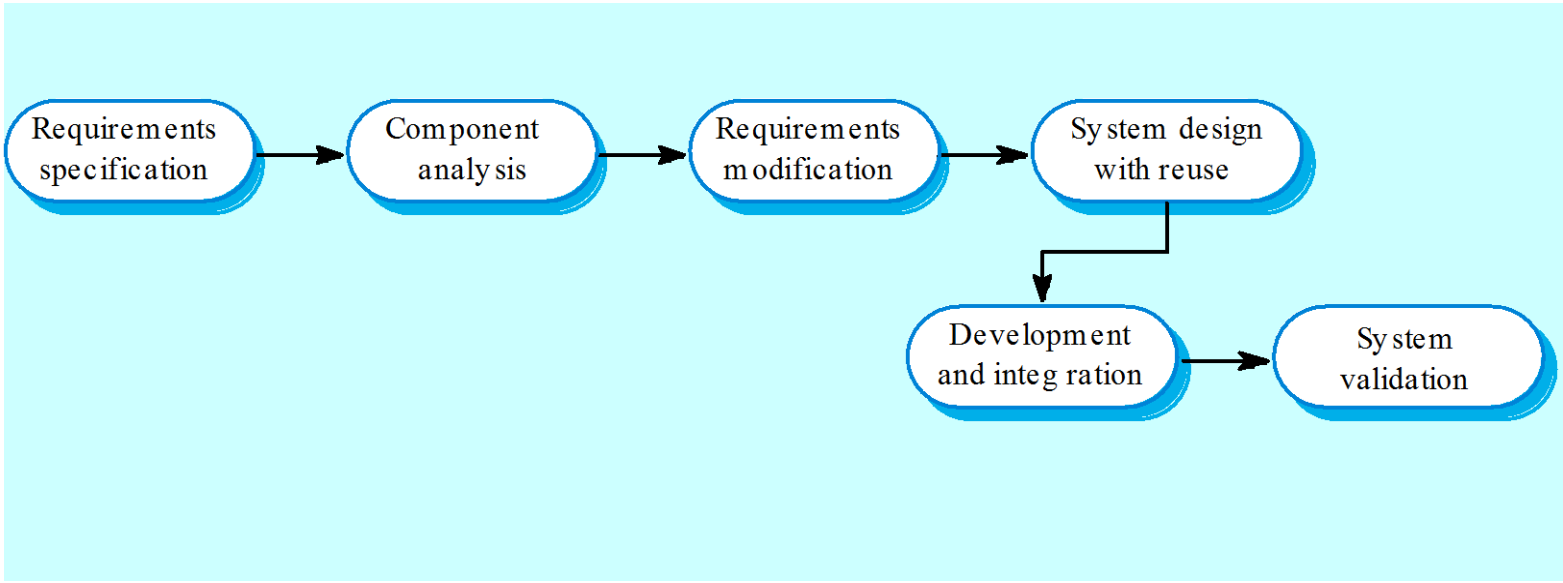
Evolutionary development

- ***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 increasingly used*** as component standards have emerged.

Reuse-oriented development



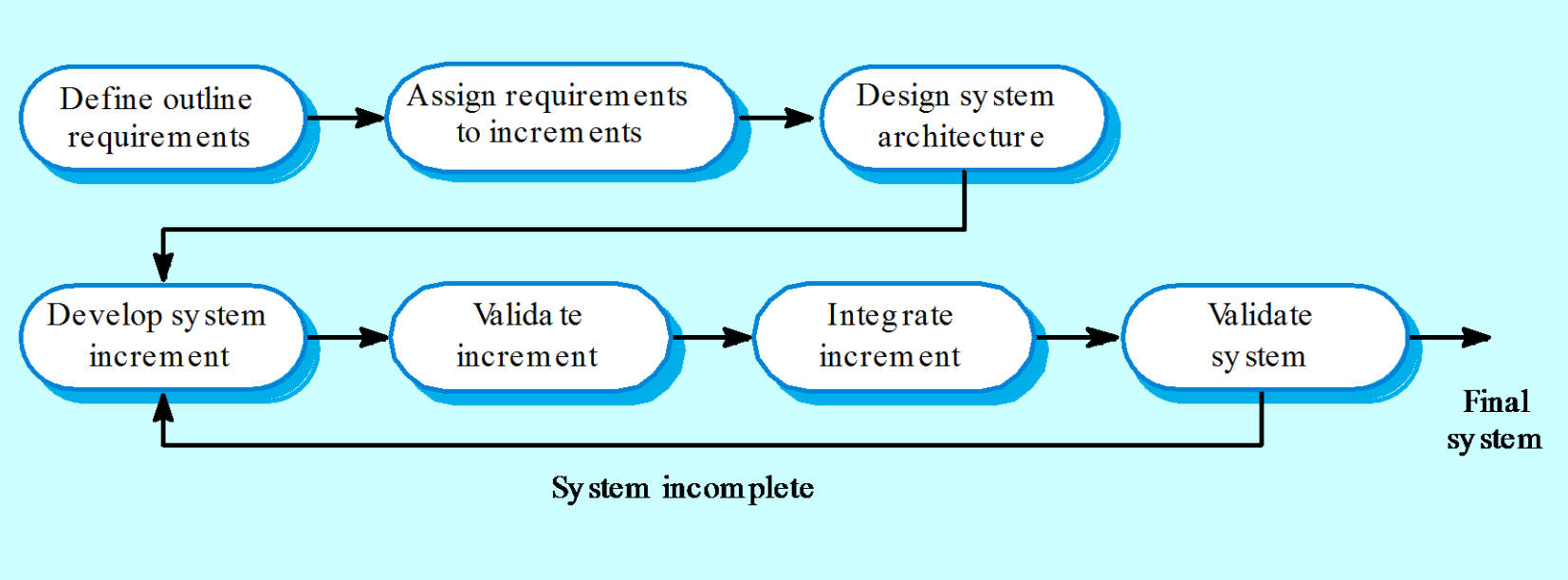
Process iteration

- System *requirements always evolve* during 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 delivery*;
  - *Spiral development*.

Incremental delivery

- Not a single delivery;
- Development and delivery in increments
- Each increment satisfies part of the required functionality.
- User requirements are prioritised and the higher the priority of a requirement the sooner it is delivered.
- Current increment’s requirements are frozen; but requirements for later increments may evolve.

Incremental development



Incremental development advantages

- Delivery of increments provides *earlier availability of the system functionality*.
- *Early increments* act as a prototype to help *elicit requirements for later increments*.
- *Lower risk of overall project failure*.
- The *highest priority system services* tend to receive the *most testing*.

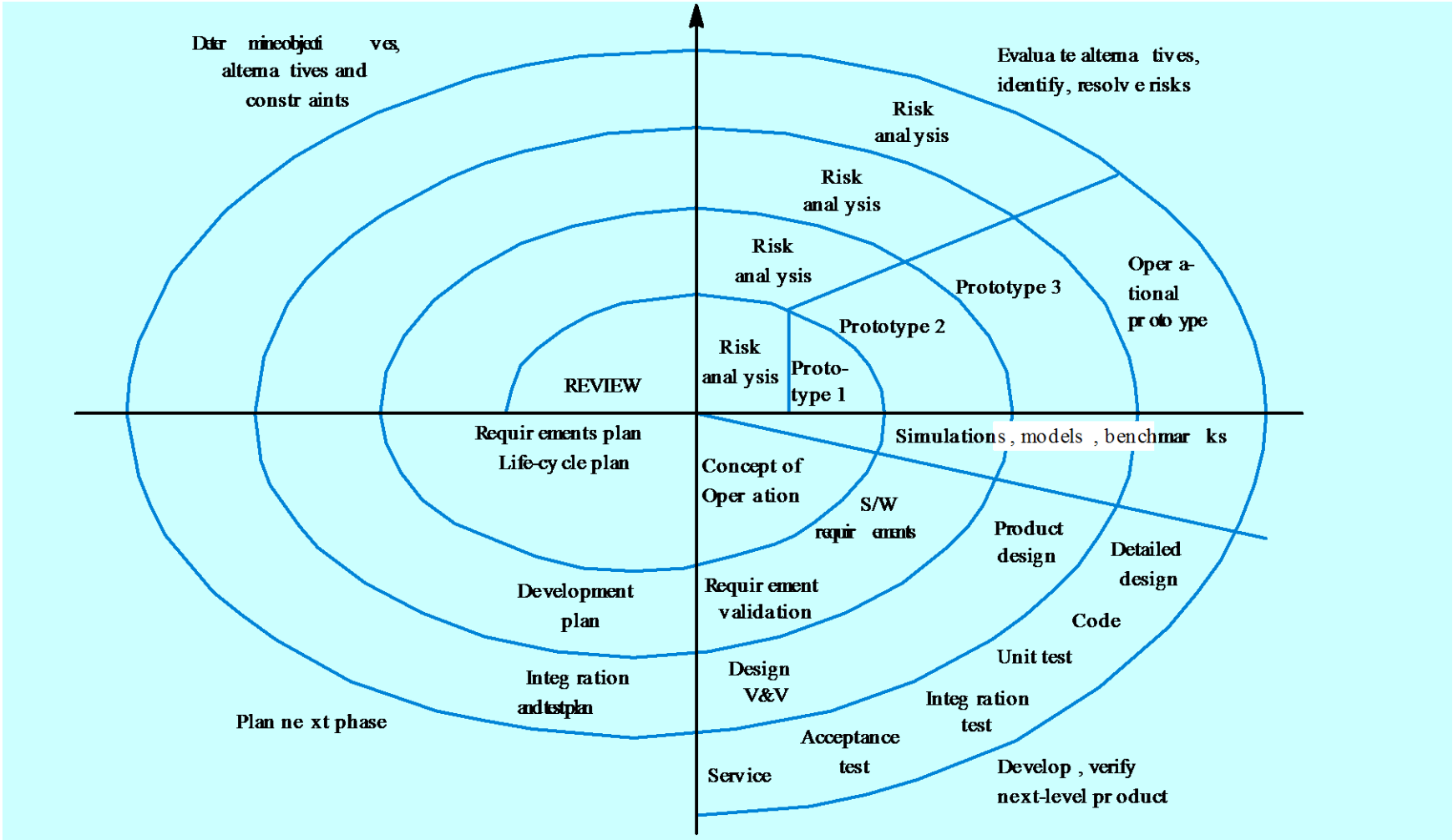
Extreme programming

- A derivative of incremental development.
- An approach to development based on the development and delivery of *very small (even weekly or biweekly) increments of functionality*.
- Relies on
  - *constant code improvement,*
  - *user involvement in the development team* and
  - *pairwise programming*.

Spiral development

- *Spiral representation of the process* 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***.

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.

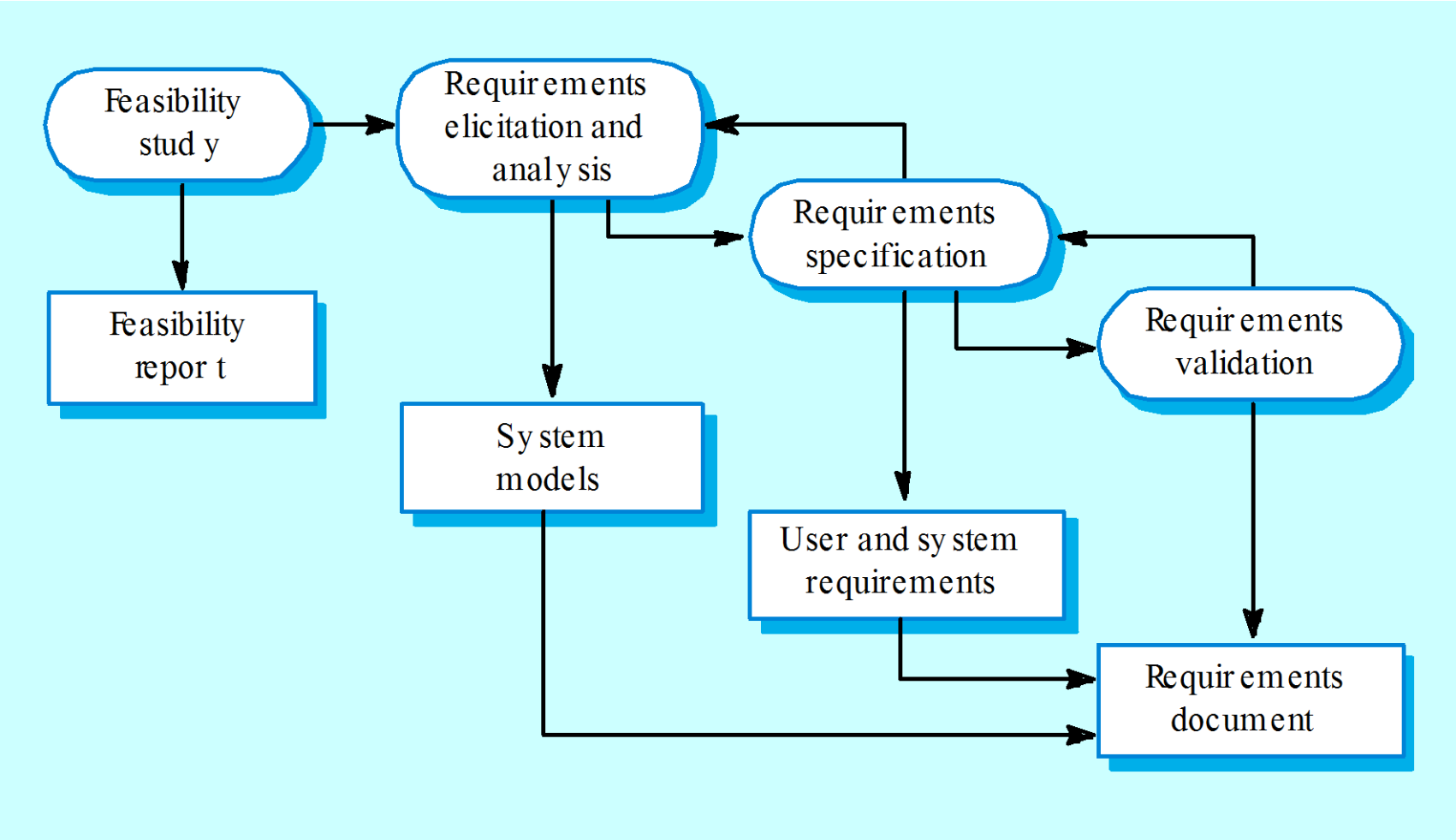
Process activities

- *Software specification*
- *Software design and implementation*
- *Software validation & verification*
- *Software evolution*

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



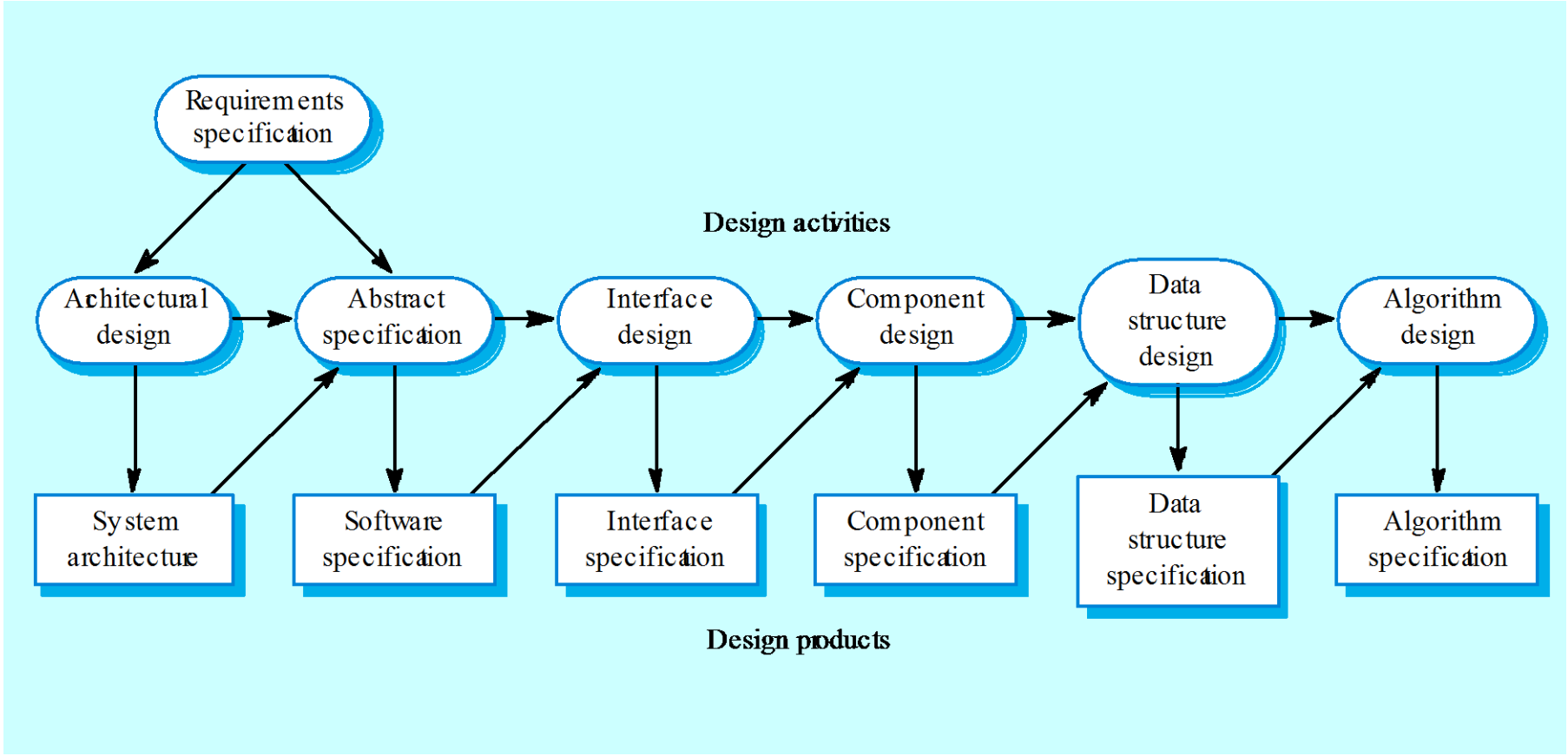
Software design and implementation

- The process of *converting the system specification into an executable system*.
- *Software design*
  - Design a software structure that realises the specification;
- *Implementation*
  - Translation of design to an executable program;
- Design and implementation may be inter-leaved.

Design process activities

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

The software design process



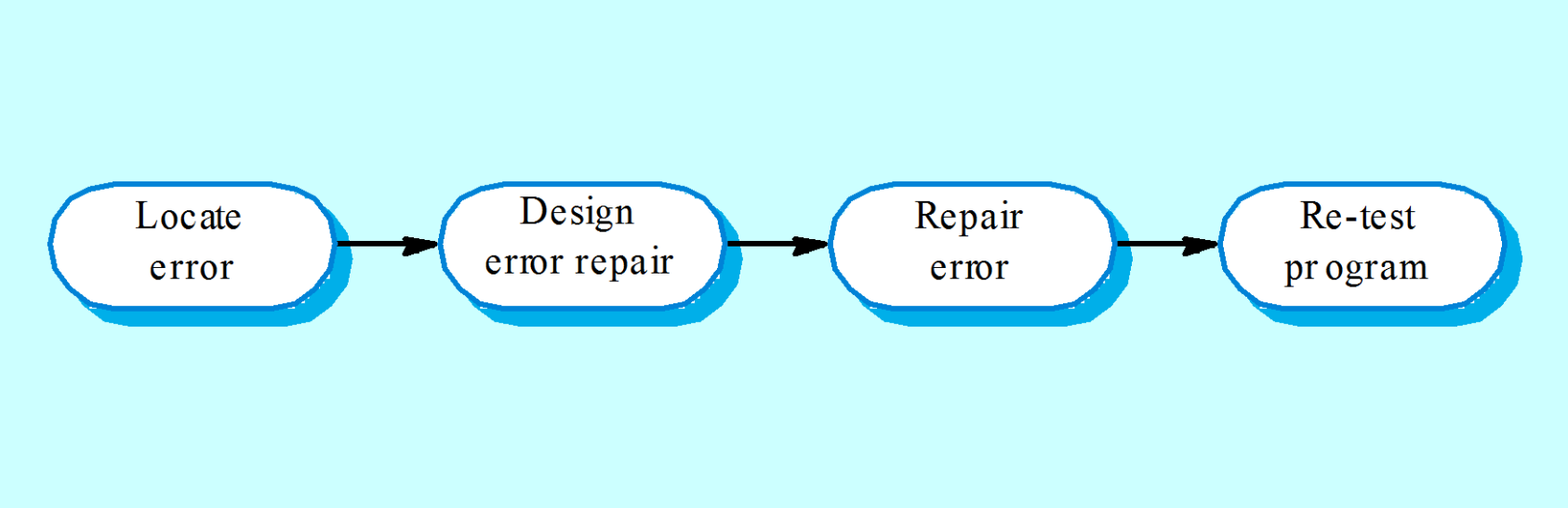
Structured methods

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

Programming and debugging

- Design $\Rightarrow$ program + error removal.
- 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.

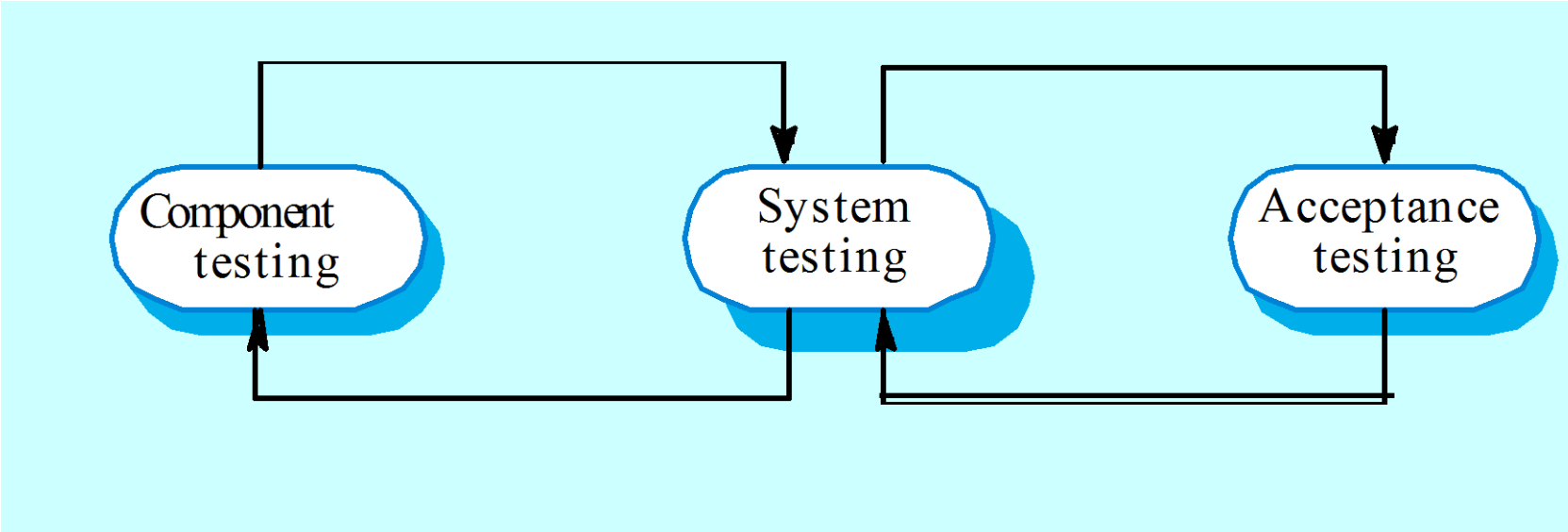
The debugging process



Software validation & verification

- Verification and validation (V & V) 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.

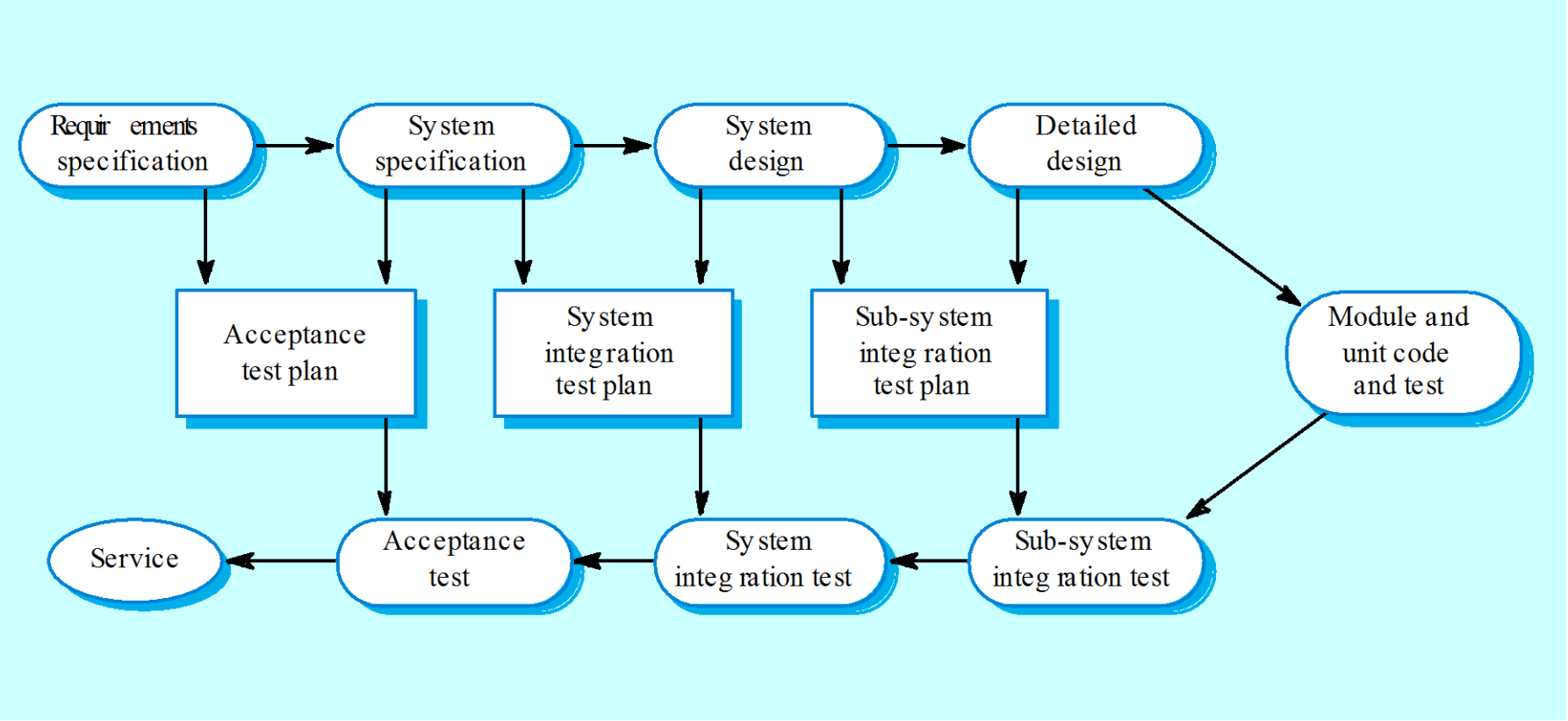
The testing process



Testing stages

- *Component or unit testing*
  - Individual components are tested independently;
  - Components may be functions or objects or coherent groupings of these entities.
- *System testing*
  - Testing of the system as a whole. Testing of emergent properties is particularly important.
- *Acceptance testing*
  - Testing with customer data to check that the system meets the customer's needs.

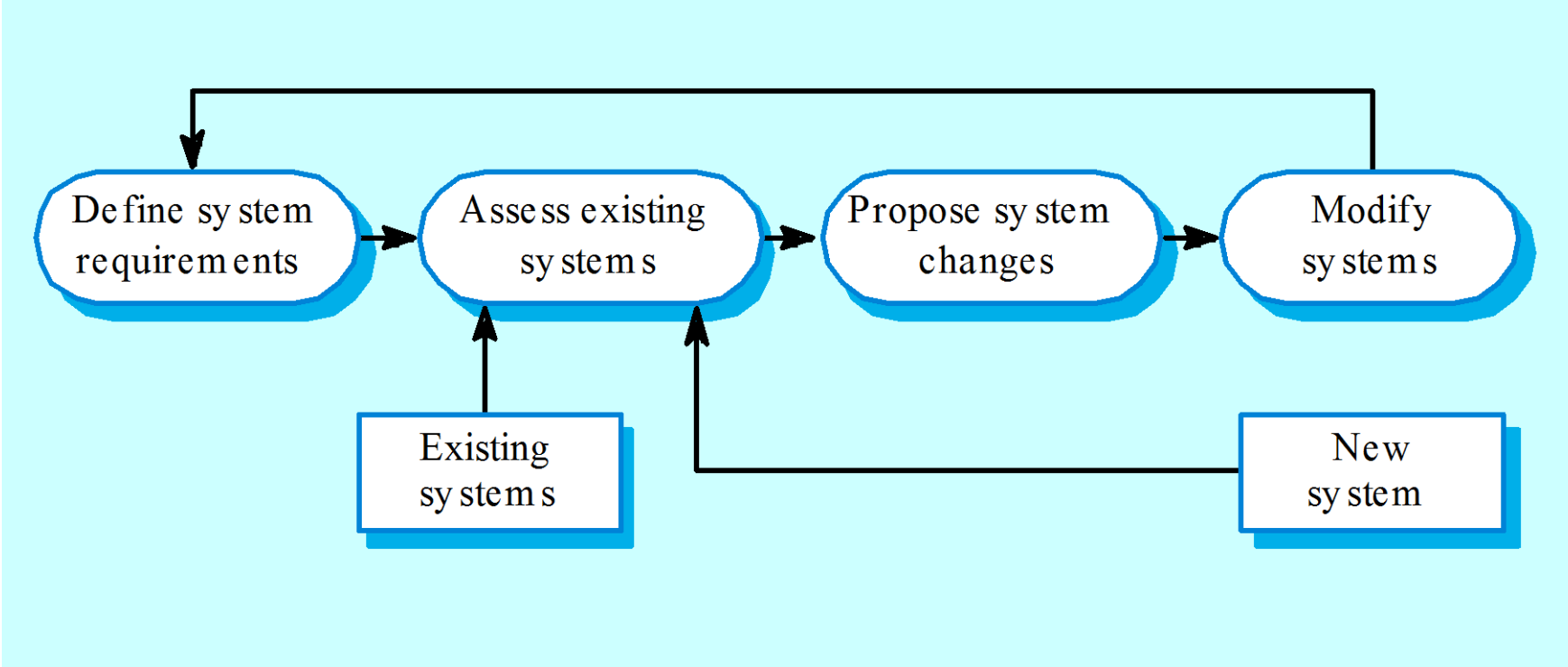
Testing phases



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 isolation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new.

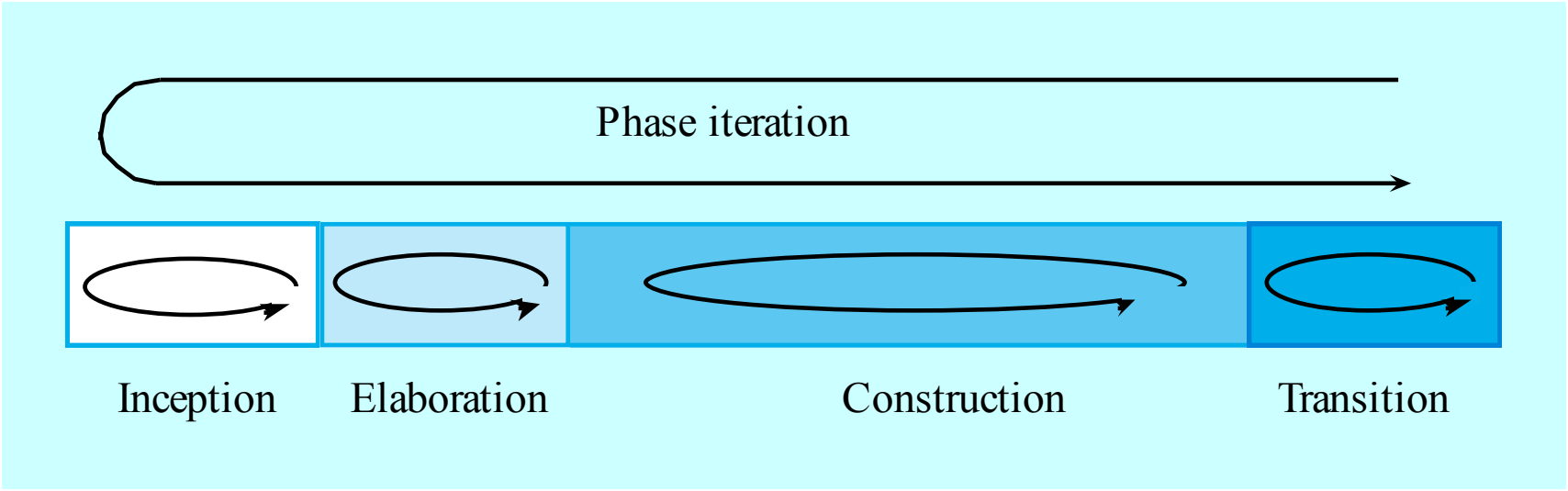
System evolution



The Rational Unified Process

- A modern process model derived from the work on the UML and associated process.
- Normally described from 3 perspectives
  - A *dynamic perspective* that shows phases over time;
  - A *static perspective* that shows process activities;
  - A *practice perspective* that suggests good practice.

RUP phase model



RUP phases

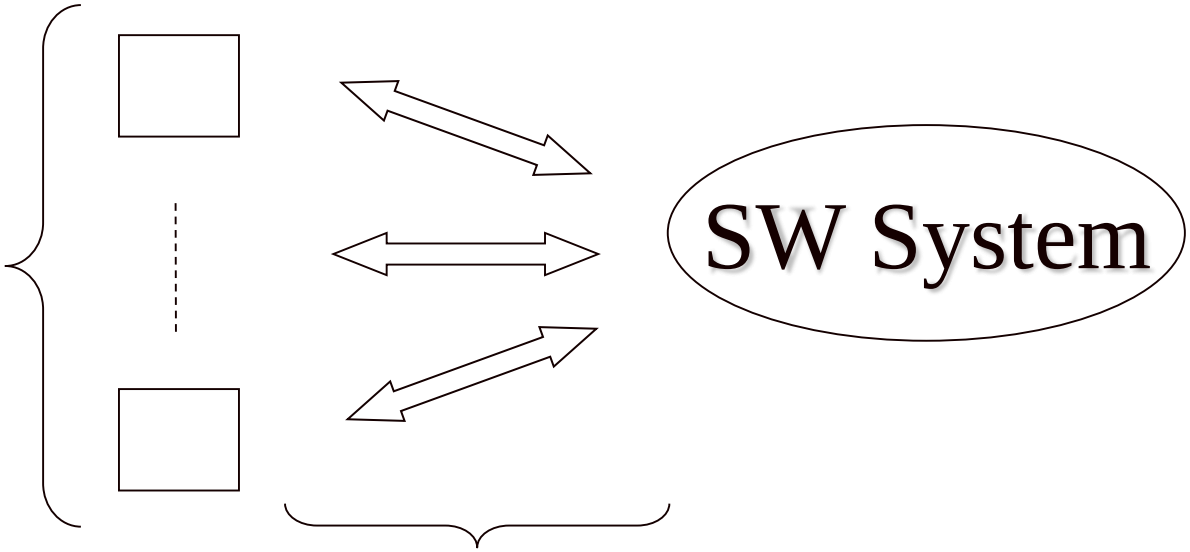
- *Inception*
  - Establish the business case for the system.
- *Elaboration*
  - Develop an understanding of the problem domain and the system architecture.
- *Construction*
  - System design, programming and testing.
- *Transition*
  - Deploy the system in its operating environment.

Inception

• Identify all external entities and their interactions with the SW system

External entities

People & other systems



Interactions of SW System with external entities

RUP good practice

- Develop software iteratively
- Manage requirements
- Use component-based architectures
- Visually model software
- Verify software quality
- Control changes to software

Static workflows

Workflow	Description
Business modelling	The business processes are modelled using business use cases.
Requirements	Actors who interact with the system are identified and use cases are developed to model the system requirements.
Analysis and design	A design model is created and documented using architectural models, component models, object models and sequence models.
Implementation	The components in the system are implemented and structured into implementation sub-systems. Automatic code generation from design models helps accelerate this process.
Test	Testing is an iterative process that is carried out in conjunction with implementation. System testing follows the completion of the implementation.
Deployment	A product release is created, distributed to users and installed in their workplace.
Configuration and change management	This supporting workflow managed changes to the system (see Chapter 29).
Project management	This supporting workflow manages the system development (see Chapter 5).
Environment	This workflow is concerned with making appropriate software tools available to the software development team.

Computer-aided software engineering (CASE)

- ... denotes software to support software development and evolution processes.
- ... automates process activities
  - *Graphical editors* for system model development;
  - *Data dictionary* to manage design entities;
  - *Graphical UI builder* for user interface construction;
  - *Debugger*s to support program fault finding;
  - *Automated translators* to generate new versions of a program.

CASE technology

- ... has led to significant improvements in software process, but not the order-of-magnitude (i.e.,essential) improvements as once predicted
  - Software engineering requires *creative thought* - this is not readily automated;
  - Software engineering is a *team activity* and, for large projects, *much time is spent in team interactio*ns. CASE technology does not really support these.

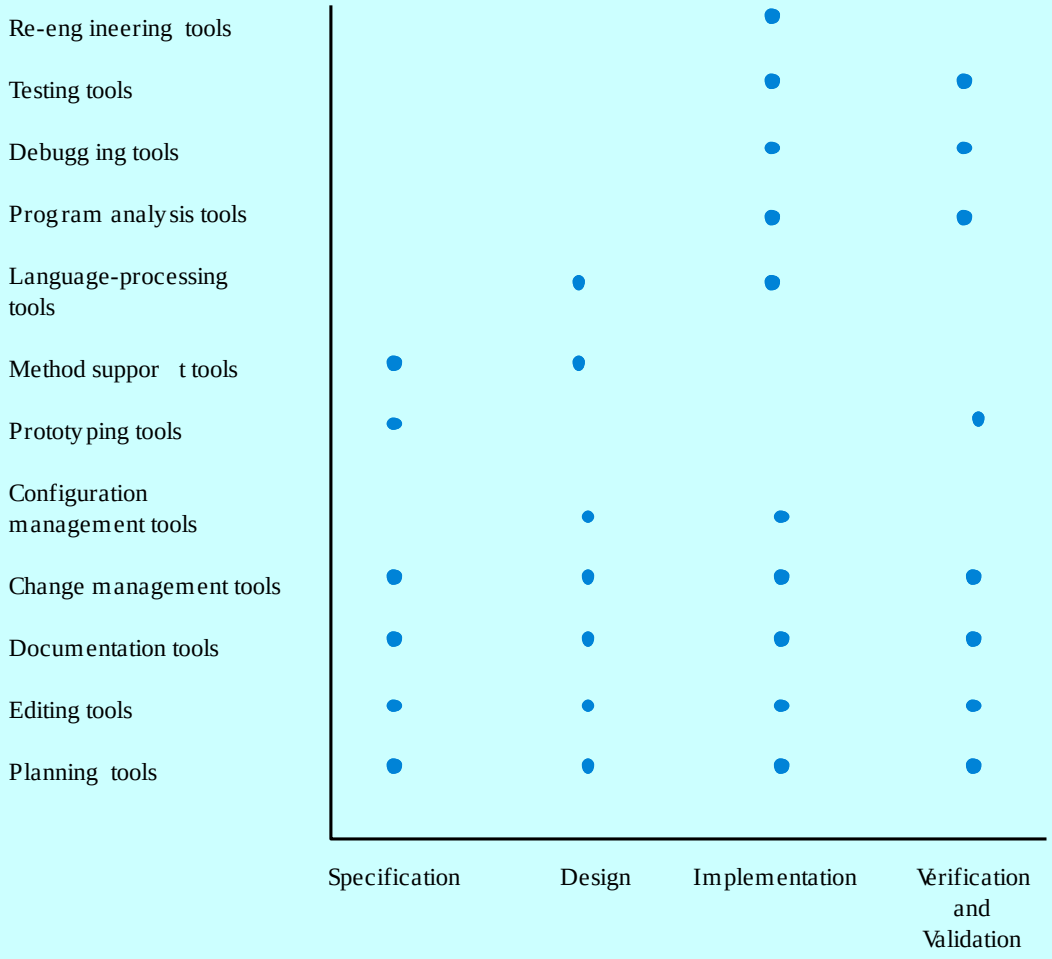
CASE classification

- 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
Change management 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 re-structuring systems

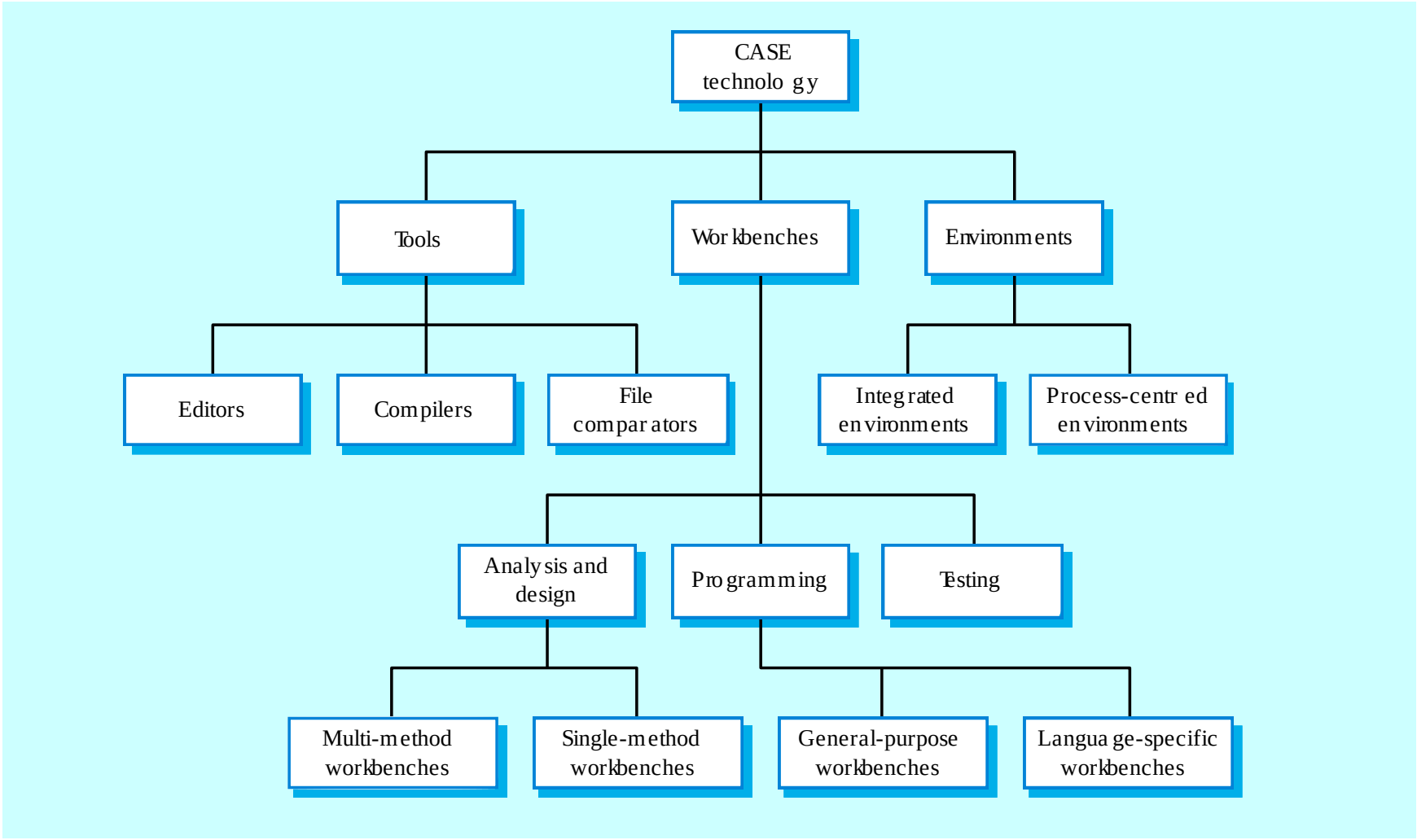
Activity-based tool classification



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



References

[1] Ian Sommerville, Software Engineering, 8th ed. 2007

[2] P. Naur, R.Randell (eds.): Software Engineering: A Report on a Conference Sponsored by the NATO Science Committee, 1968