Week 2:
Software Processes

Nguyễn Thị Minh Tuyền

KHOA CÔNG NGHỆ T
TRƯỜNG ĐẠI HỌC K

Topics covered

1. Software process models
2. Process activities
3. Coping with change
4. Boehm's spriral and RUP

Definitions

- What is a process?
- Four activities that are fundamental to software engineering?
- What is process model?

The software process

- A structured set of activities required to develop a software system.
- 4 fundamental 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.
- A software process model
  - Is an abstract representation of a process.
  - Presents a description of a process from some particular perspective.

Software process descriptions

- When we describe and discuss processes, we usually talk about
  - the **activities** in these processes such as specifying a data model, designing a user interface, etc. and
  - the **ordering** of these activities.
- Process descriptions may also include:
  - **Products**, which are the outcomes of a process activity;
  - **Roles**, which reflect the responsibilities of the people involved in the process;
  - **Pre- and post-conditions**, which are statements that are true before and after a process activity has been enacted or a product produced.

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.

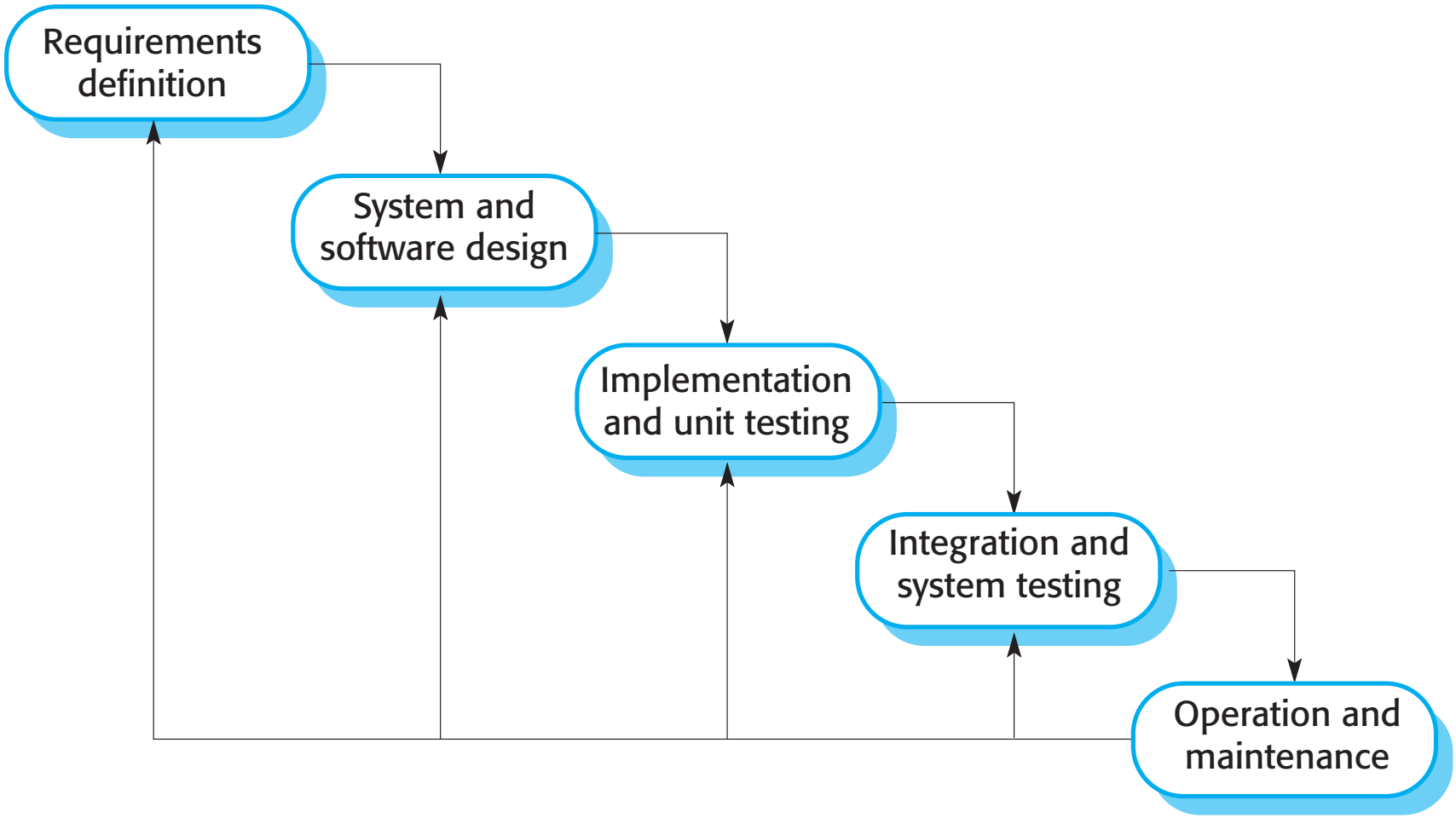
Topics covered

1. Software process models
2. Process activities
3. Coping with change
4. Boehm's spriral and RUP

Software process models

- The waterfall model
  - Plan-driven model. Separate and distinct phases of specification and development.
- Incremental development
  - Specification, development and validation are interleaved. May be plan-driven or agile.
- Reuse-oriented software engineering
  - The system is assembled from existing components. May be plan-driven or agile.
- In practice, most large systems are developed using a process that incorporates elements from all of these models.

The waterfall model



Waterfall model phases

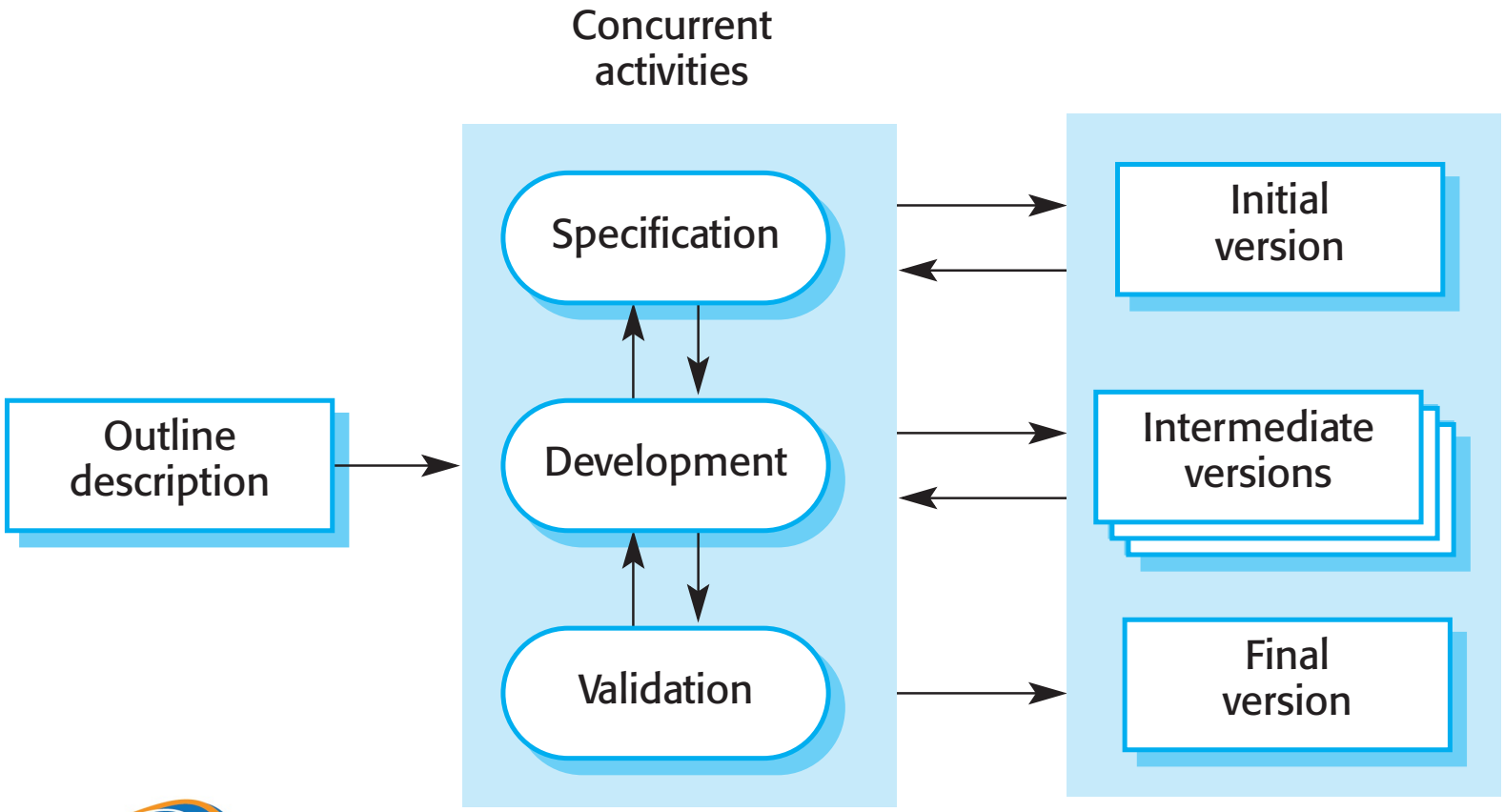
- The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway.
- In principle, a phase has to be complete before moving onto the next phase.

Waterfall model problems

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

- An important variant of the waterfall model is formal system development
  - Use a mathematical model to specify a system.
  - Use mathematical transformations that preserve its consistency, into executable code.
  - Mathematical transformations are correct: a program generated in this way is consistent with its specification.
- Formal development processes (B method for example) are suited to the development of systems that have stringent safety, reliability, or security requirements.

Incremental development



Incremental development benefits

- The cost of accommodating changing customer requirements is reduced.
  - The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.
- It is easier to get customer feedback on the development work that has been done.
  - Customers can comment on demonstrations of the software and see how much has been implemented.
- More rapid delivery and deployment of useful software to the customer is possible.
  - Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

Incremental development problems

- The process is not visible.
  - Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.
- 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. Incorporating further software changes becomes increasingly difficult and costly.

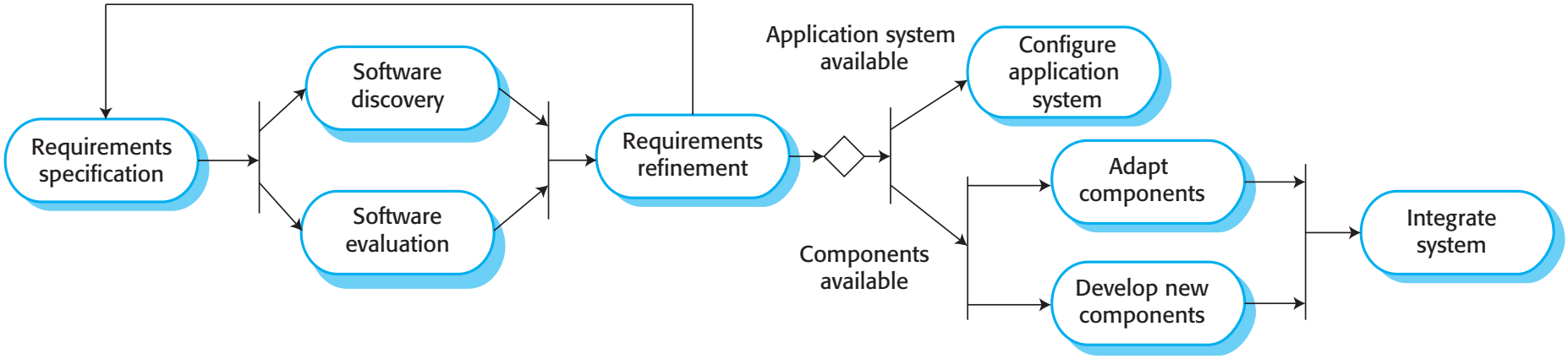
Integration and configuration

- Based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems.
- Reused elements may be configured to adapt their behaviour and functionality to a user's requirements
- Reuse is now the standard approach for building many types of business system

Types of reusable software

- Stand-alone application systems (sometimes called COTS) that are configured for use in a particular environment.
- Collections of objects that are developed as a package to be integrated with a component framework such as .NET or J2EE.
- Web services that are developed according to service standards and which are available for remote invocation.

Reuse-oriented software engineering



Advantages and drawbacks

- Advantages
  - Reduced costs and risks as less software is developed from scratch
  - Faster delivery and deployment of system
- Drawbacks
  - requirements compromises are inevitable
    - this may lead to a system that does not meet the real needs of users.
  - Loss of control over evolution of reused system elements

Topics covered

1. Software process models
2. Process activities
3. Coping with change
4. Boehm's spriral and RUP

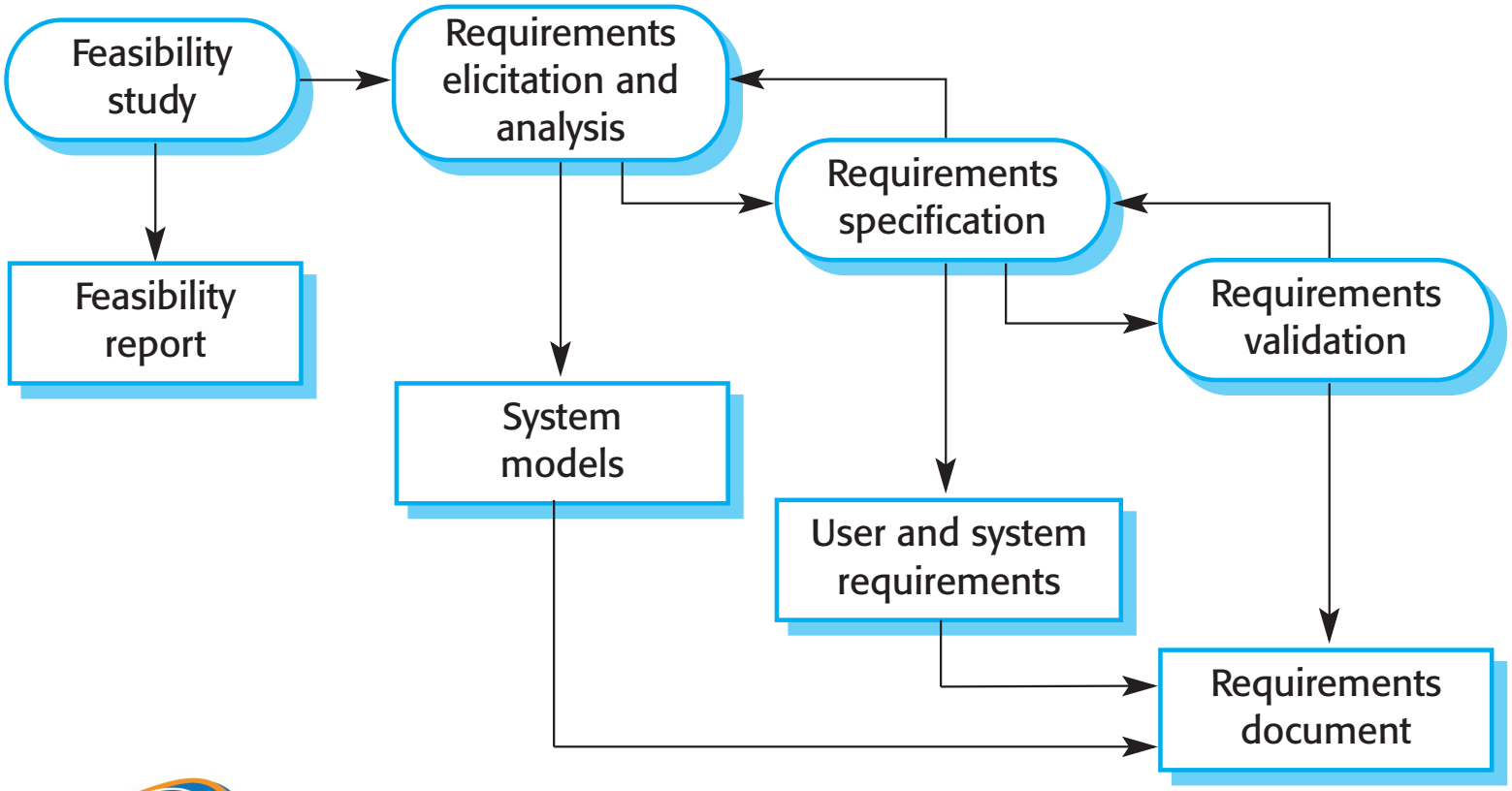
Process activities

- Real software processes are interleaved sequences of technical, collaborative and managerial activities with the overall goal of specifying, designing, implementing and testing a software system.
- The four basic process activities of specification, development, validation and evolution are organized differently in different development processes.
- For example:
  - In the waterfall model, they are organized in sequence,
  - In incremental development, they are interleaved.

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

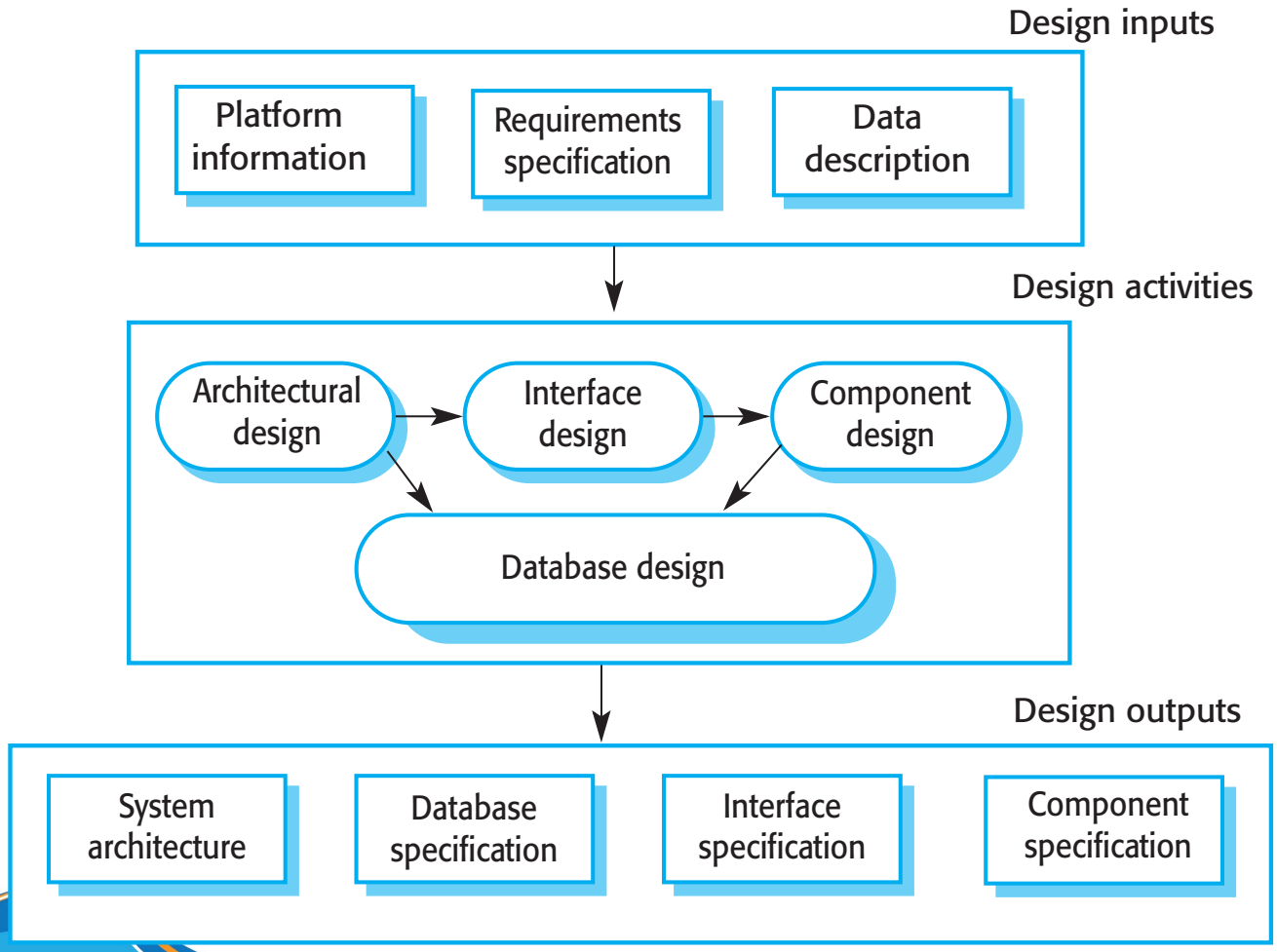
The requirements engineering process



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.

A general model of the design process



Design activities

- Architectural design, where you identify
  - the overall structure of the system,
  - the principal components, their relationships and how they are distributed.
- Interface design
  - define the interfaces between system components.
- Component selection and design
  - where you search for reusable components. If unavailable, you design how it will operate.
- Database design
  - design the system data structures and how these are to be represented in a database.

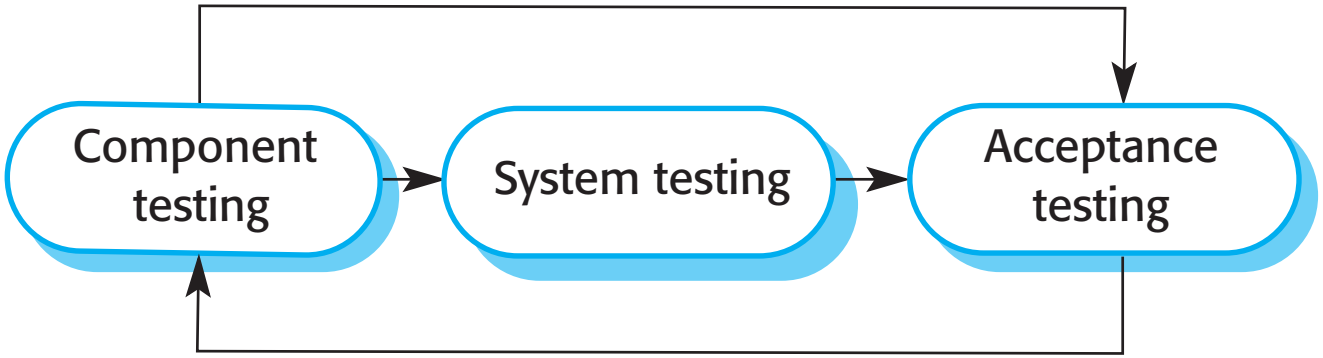
System implementation

- The software is implemented either by developing a program or programs or by configuring an application system.
- Design and implementation are interleaved activities for most types of software system.
- Programming is an individual activity with no standard process.
- Debugging is the activity of finding program faults and correcting these faults.

Software validation

- 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.
- Testing is the most commonly used V & V activity.

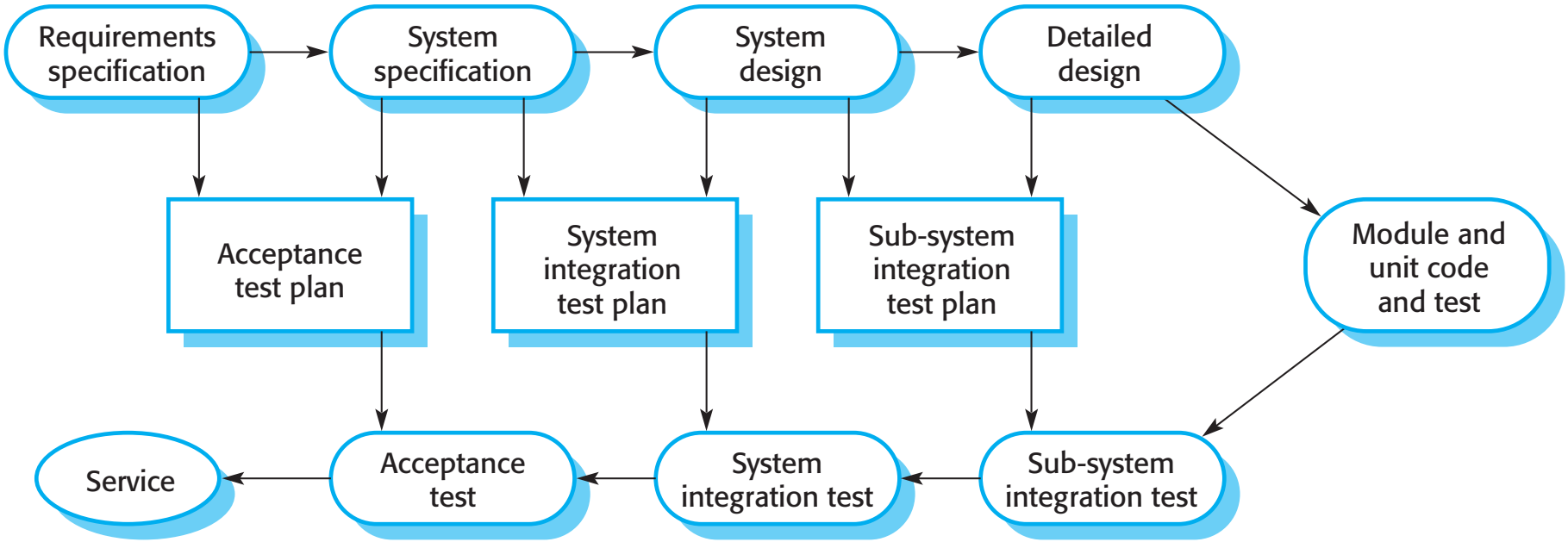
Stages of testing



Testing stages

- Development or component 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.

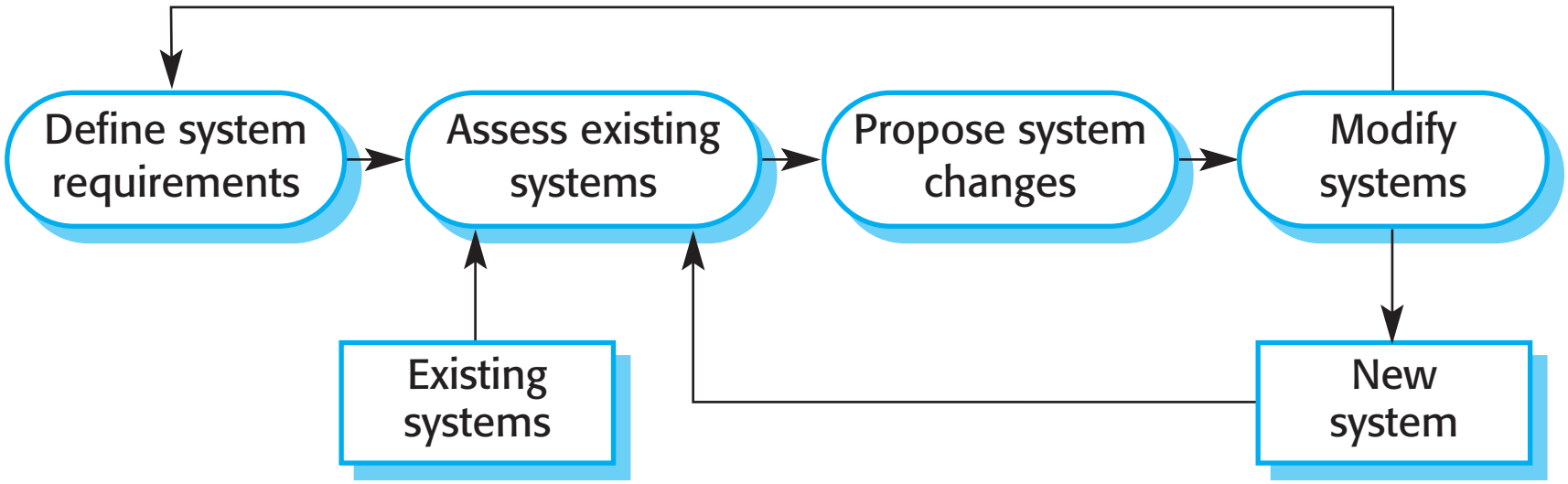
Testing phases in a plan-driven software process



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 this is increasingly irrelevant as fewer and fewer systems are completely new.

System evolution



Topics covered

1. Software process models
2. Process activities
3. Coping with change
4. Boehm's spriral and RUP

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 rework so the costs of change include both rework as well as the costs of implementing new functionality

Reducing the costs of rework

- Change anticipation (Change avoidance)
  - The software process includes activities that can anticipate possible changes before significant rework is required.
  - Example: a prototype system may be developed to show some key features of the system to customers.
- Change tolerance
  - The process is designed so that changes can be accommodated at relatively low cost.
  - This involves some form of incremental development.

Coping with changing requirements

- System prototyping
  - A version of the system or part of the system is developed quickly to check the customer's requirements and the feasibility of design decisions.
  - This approach supports change anticipation.
- Incremental delivery
  - System increments are delivered to the customer for comment and experimentation.
  - This supports both change avoidance and change tolerance.

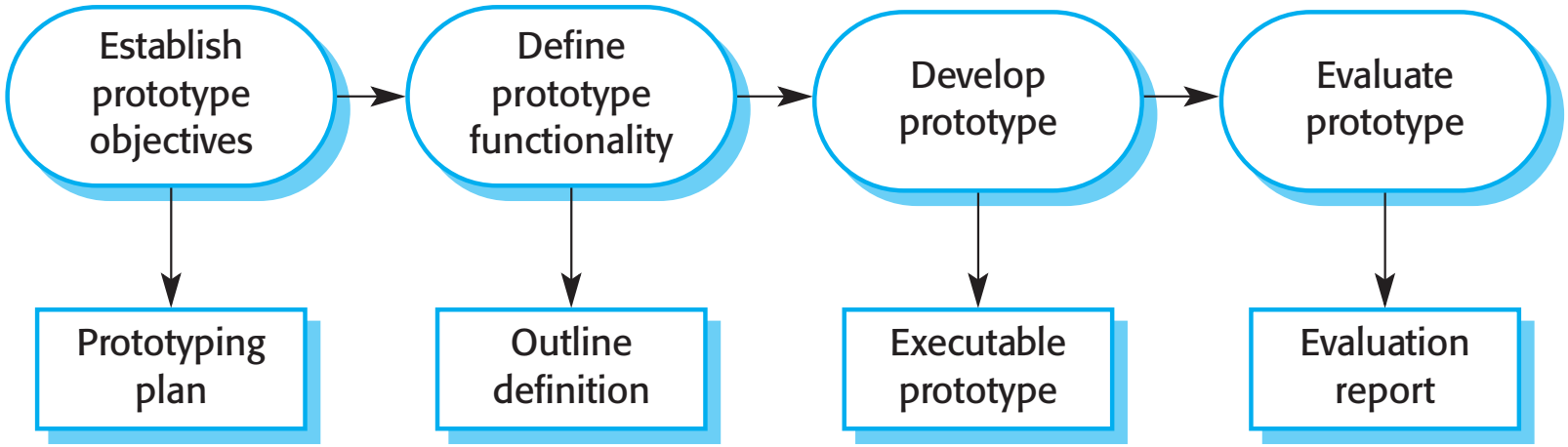
Software prototyping

- 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 design processes to explore options and develop a UI design;
  - In the testing process to run back-to-back tests.

Benefits of prototyping

- Improved system usability.
- A closer match to users' real needs.
- Improved design quality.
- Improved maintainability.
- Reduced development effort.

The process of prototype development



Prototype development

- May be based on rapid prototyping languages or tools
- May involve leaving out functionality
  - Prototype should focus 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

Throwaway prototypes

- Prototypes should be discarded after development as they are not a good basis for a production system:
  - It may be impossible to tune the system 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 organisational quality standards.

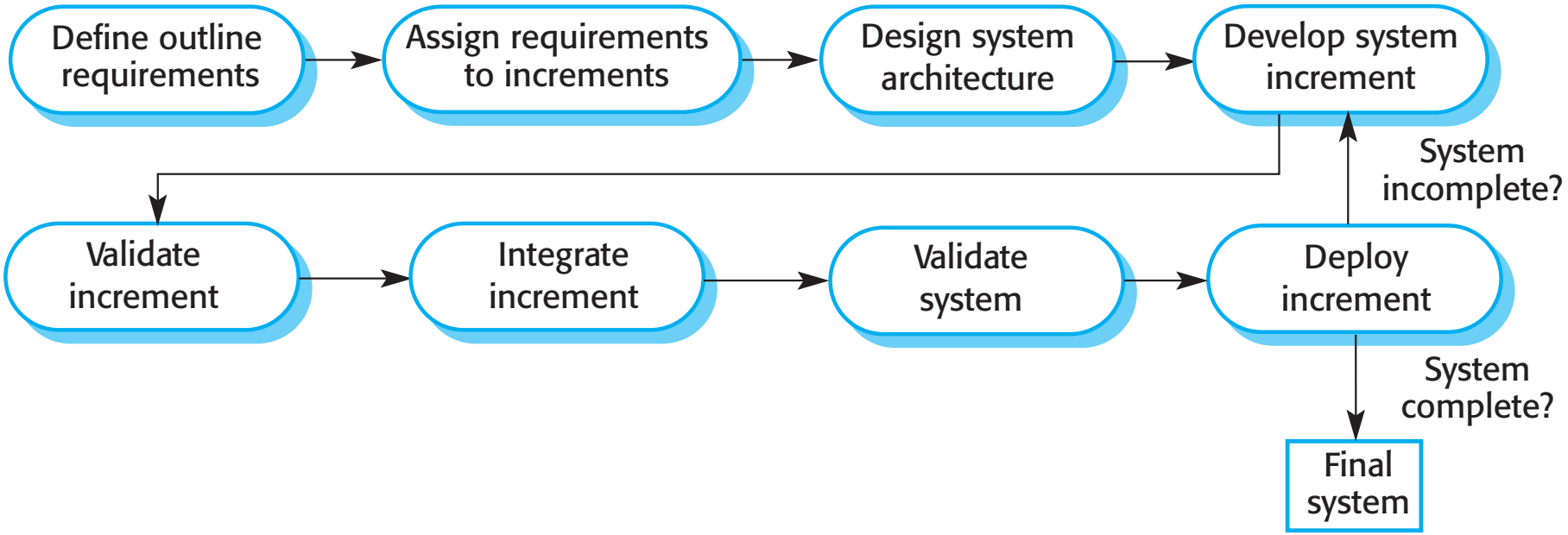
Incremental delivery

- 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 and delivery

- Incremental development
  - Develop the system in increments and evaluate each increment before proceeding to the development of the next increment;
  - Normal approach used in agile methods;
  - Evaluation done by user/customer proxy.
- Incremental delivery
  - Deploy an increment for use by end-users;
  - More realistic evaluation about practical use of software;
  - Difficult to implement for replacement systems as increments have less functionality than the system being replaced.

Incremental delivery



Incremental delivery benefits

- 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.
- Lower risk of overall project failure.
- The highest priority system services tend to receive the most testing.

Incremental delivery problems

- Most systems require a set of basic facilities that are used by different parts of the system.
  - As requirements are not defined in detail until an increment is to be implemented, it can be hard to identify common facilities that are needed by all increments.
- The essence of iterative processes is that the specification is developed in conjunction with the software.
  - However, this conflicts with the procurement model of many organizations, where the complete system specification is part of the system development contract.

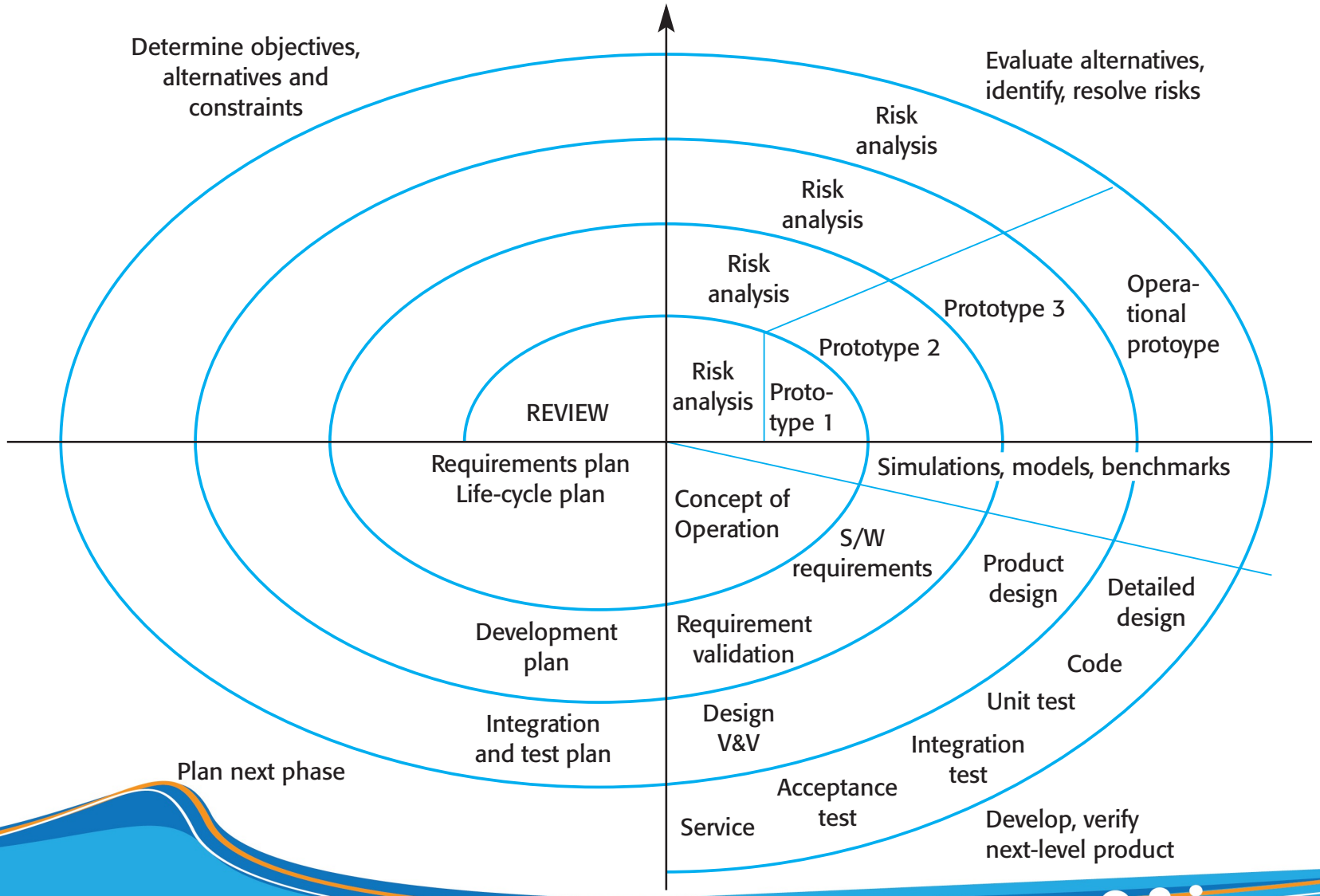
Topics covered

1. Software process models
2. Process activities
3. Coping with change
4. Boehm's spriral and RUP

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.

Boehm's 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.

Spiral model usage

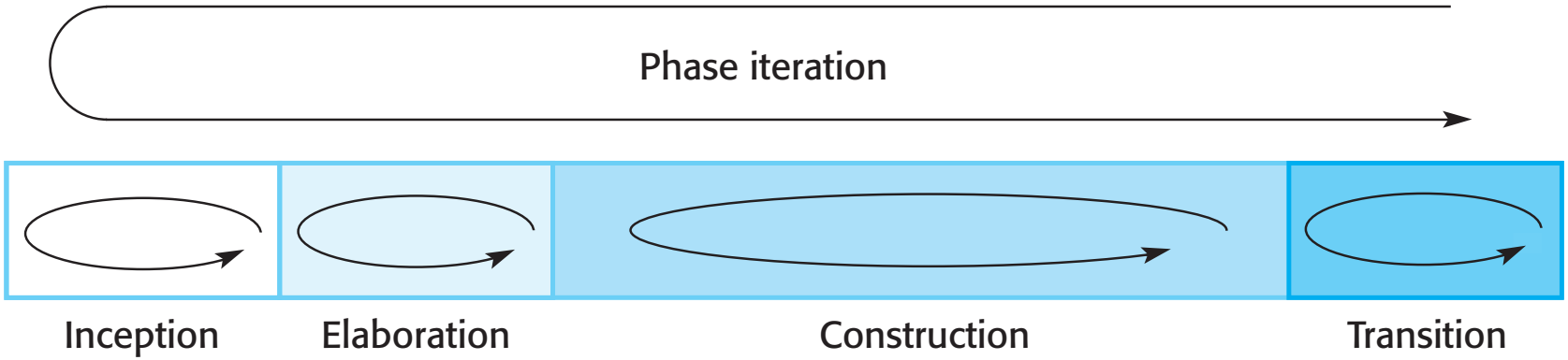
- Spiral model has been very influential in helping people think about iteration in software processes and introducing the risk-driven approach to development.
- In practice, however, the model is rarely used as published for practical software development.

The Rational Unified Process

The Rational Unified Process

- A modern generic process derived from the work on the UML and associated process.
- Brings together aspects of the 3 generic process models discussed previously.
- Normally described from 3 perspectives
  - A dynamic perspective that shows phases over time;
  - A static perspective that shows process activities;
  - A practive perspective that suggests good practice.

Phases in the Rational Unified Process

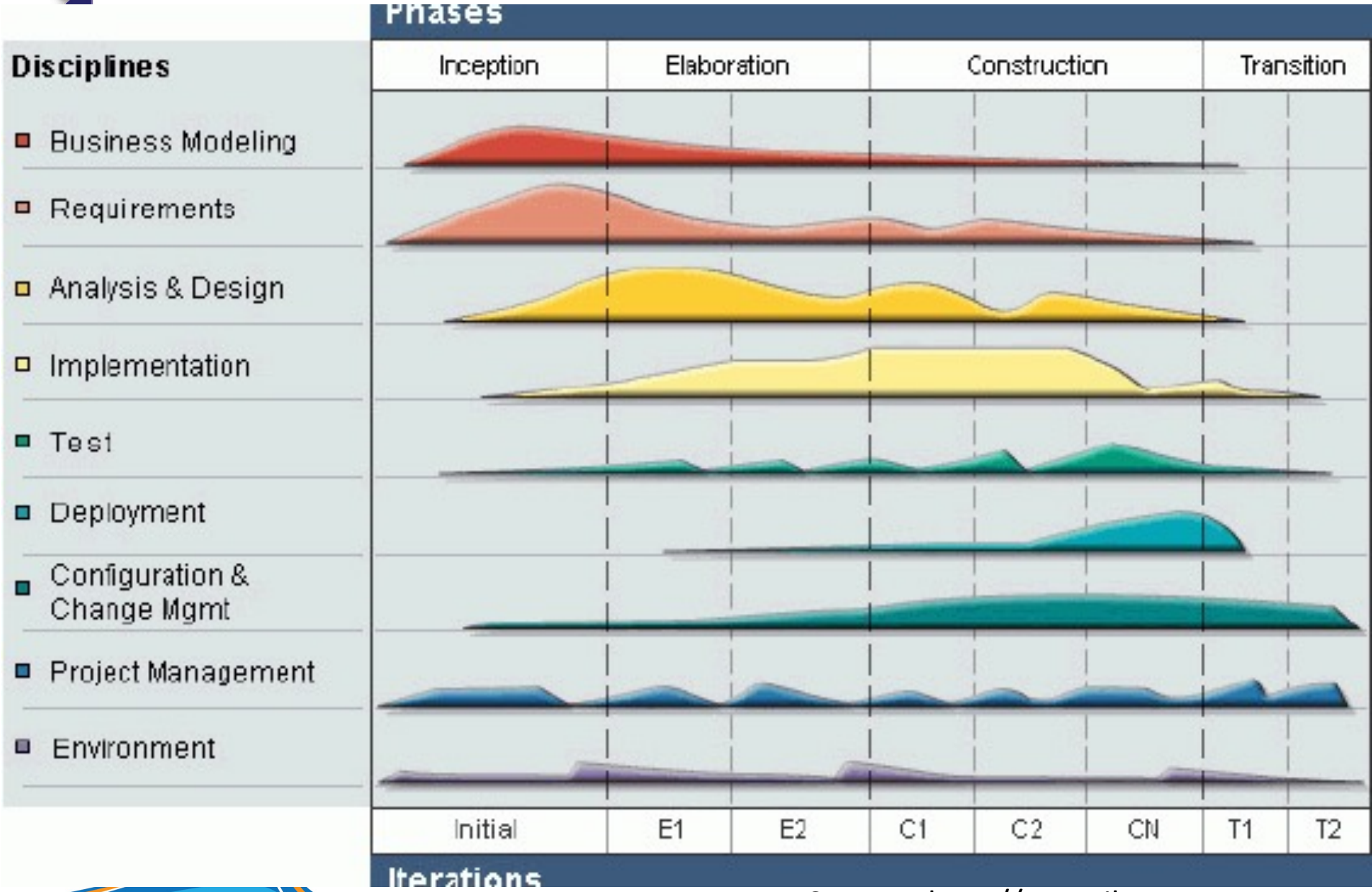


RUP phases

- 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.

RUP iteration

- In-phase iteration
  - Each phase is iterative with results developed incrementally.
- Cross-phase iteration
  - As shown by the loop in the RUP model, the whole set of phases may be enacted incrementally.



Source : http://www.ibm.com

Static workflows in the Rational Unified Process

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.

Static workflows in the Rational Unified Process

Workflow	Description
Testing	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
Project management	This supporting workflow manages the system development
Environment	This workflow is concerned with making appropriate software tools available to the software development team.

RUP good practice

- Develop software iteratively
  - Plan increments based on customer priorities and deliver highest priority increments first.
- Manage requirements
  - Explicitly document customer requirements and keep track of changes to these requirements.
- Use component-based architectures
  - Organize the system architecture as a set of reusable components.

RUP good practice

- Visually model software
  - 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
  - Manage software changes using a change management system and configuration management tools.

Questions?