Software Processes

Eunseok Lee, Prof.
College of Software
Sungkyunkwan University

Objectives

- Introduce the idea of a SW process- *a coherent set of activities for SW production*;
- Understand three generic SW process models and when they might be used;
- Know about the fundamental process activities of SW requirements engineering, SW development, testing, and evolution;
- Understand why processes should be organized to cope with changes in the SW requirements and design;
- Understand the basic notion of software process improvement.

Topics covered

- Software process models
- Process activities
- Coping with change
- Process improvement

- Appendix: Other process models(pp.50-70)

The software process

- A structured set of activities required to develop a software system.
- Many different software processes but all involve:
  - 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. It 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.

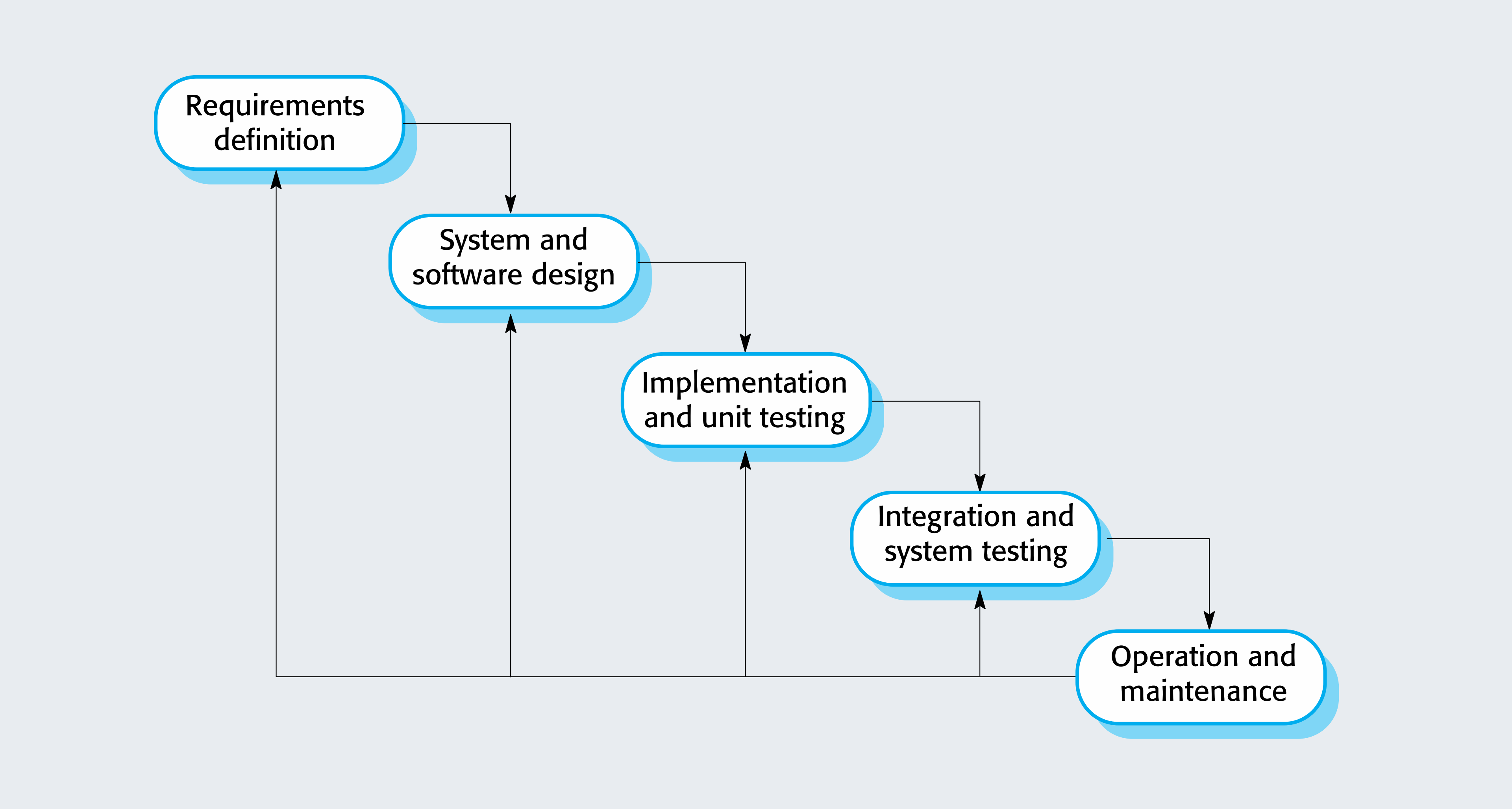
Plan-driven and agile processes

- 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.

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


Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

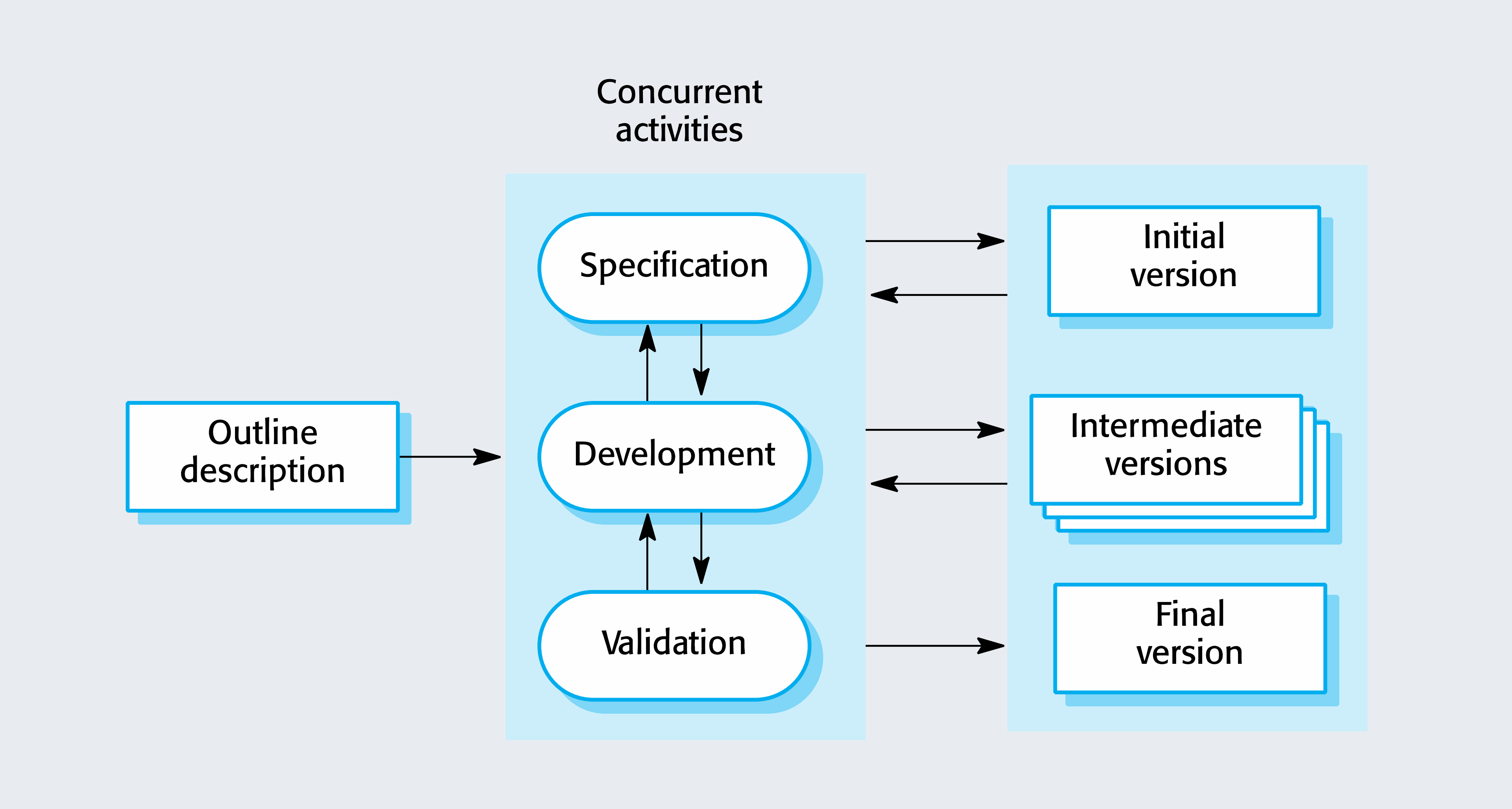
Waterfall model phases

- **There are separate identified phases in the waterfall model:**
  - Requirements analysis and definition
  - System and software design
  - Implementation and unit testing
  - Integration and system testing
  - Operation and maintenance
- 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.
  - 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 nature of the waterfall model helps coordinate the work. (Parallel development)
  - Easy of management

Incremental development model



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

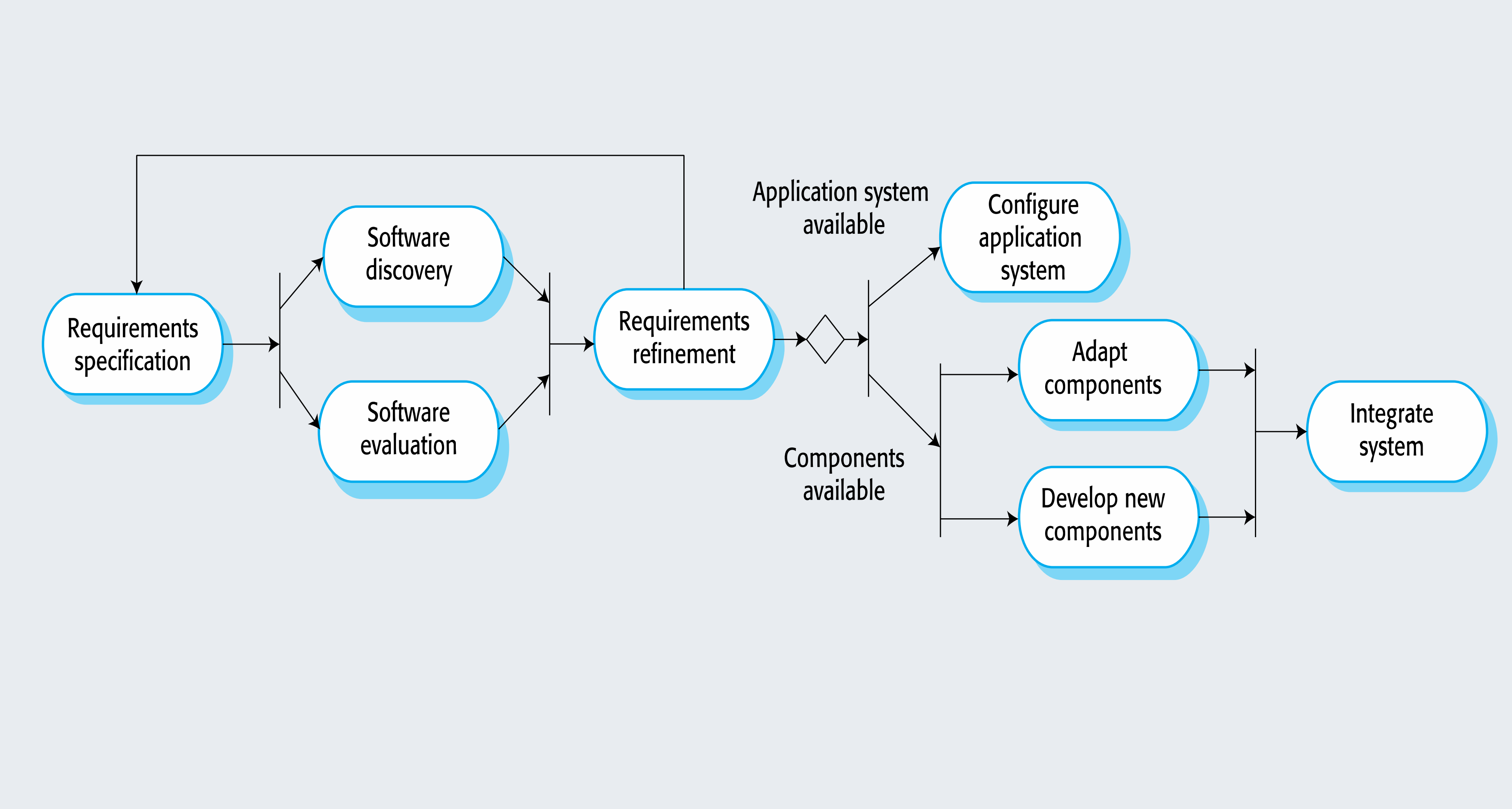
- **The process is not visible.**
  - Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.

- **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.

Reuse-oriented software engineering

- Based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems.
- Process stages
  - Requirements specification
  - Software discovery and evaluation
  - Requirements refinement
  - Application system configuration
  - Component adaptation and integration
- Reuse is now the standard approach for building many types of business system

Reuse-oriented software engineering



Types of software component

- Web services that are developed according to service standards and which are available for remote invocation.
- Collections of objects that are developed as a package to be integrated with a component framework such as .NET or J2EE.
- Stand-alone software systems (COTS) that are configured for use in a particular environment.
- Open source software components License-based software delivered through various open source communities

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

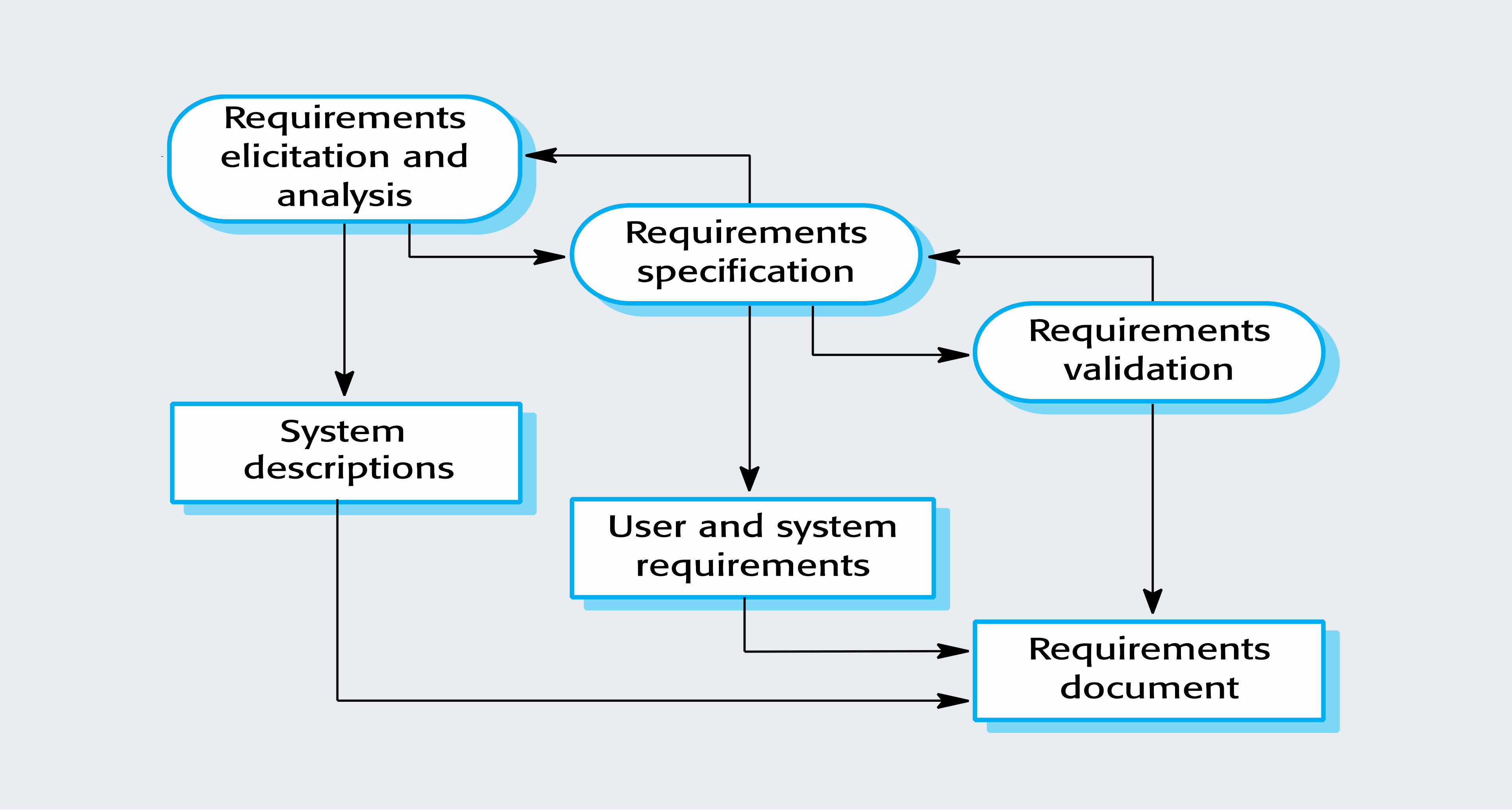
Process activities

- Real software processes are *inter-leaved 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.
- In the waterfall model, they are organized in sequence, whereas in incremental development they are inter-leaved.

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 (if not did)
    - 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



Requirements elicitation and analysis

Requirements specification

Requirements validation

System descriptions

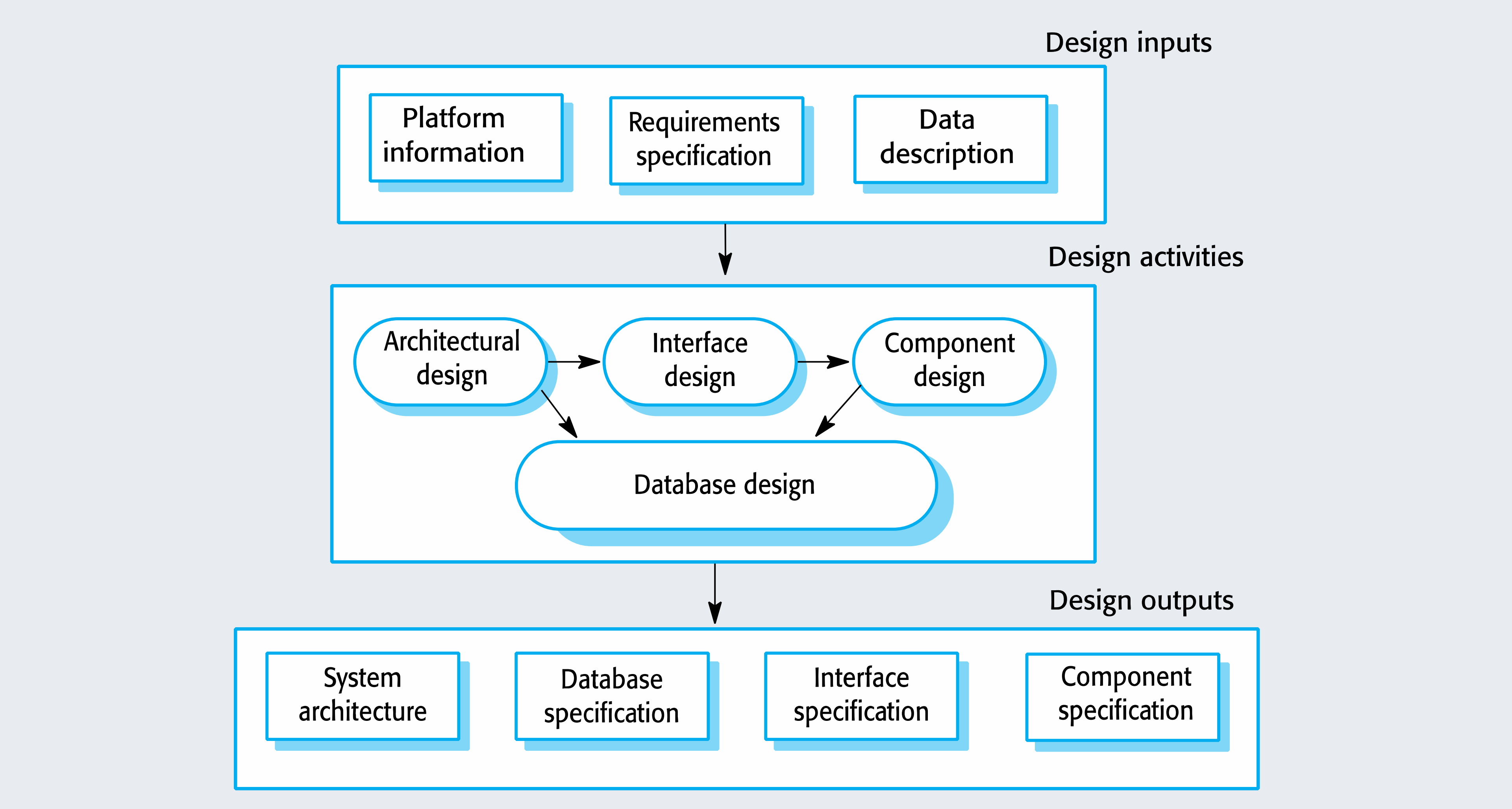
User and system requirements

Requirements document

Software design and implementation

- The process of converting the system specification into an executable system.
- Software design
  - Design a software structure that realizes 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 (sometimes called *sub-systems* or *modules*), their relationships and how they are distributed.
- *Interface design*, where you 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*, where you design the system data structures and how these are to be represented in a database.

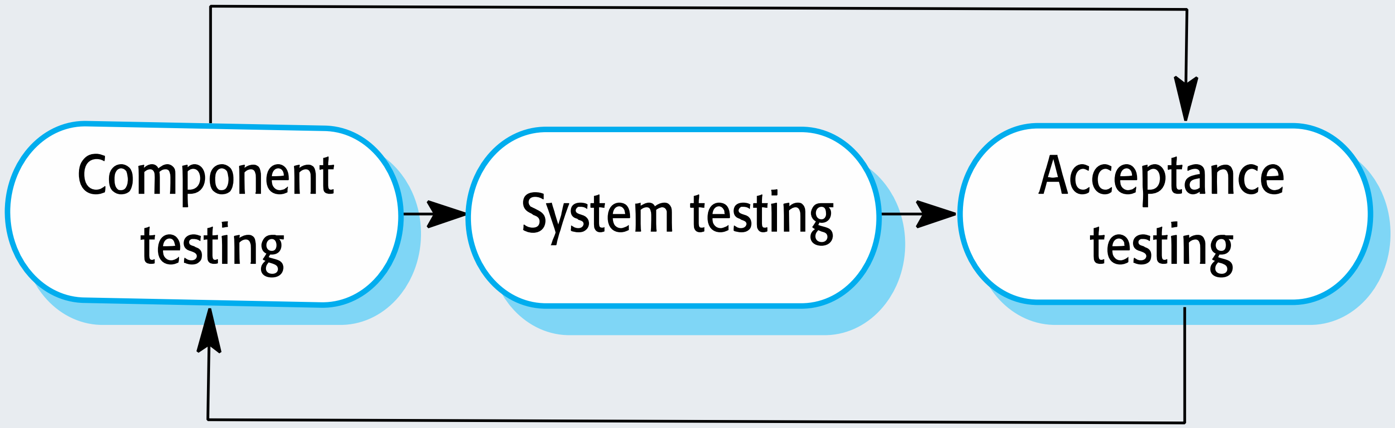
System implementation

- 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.

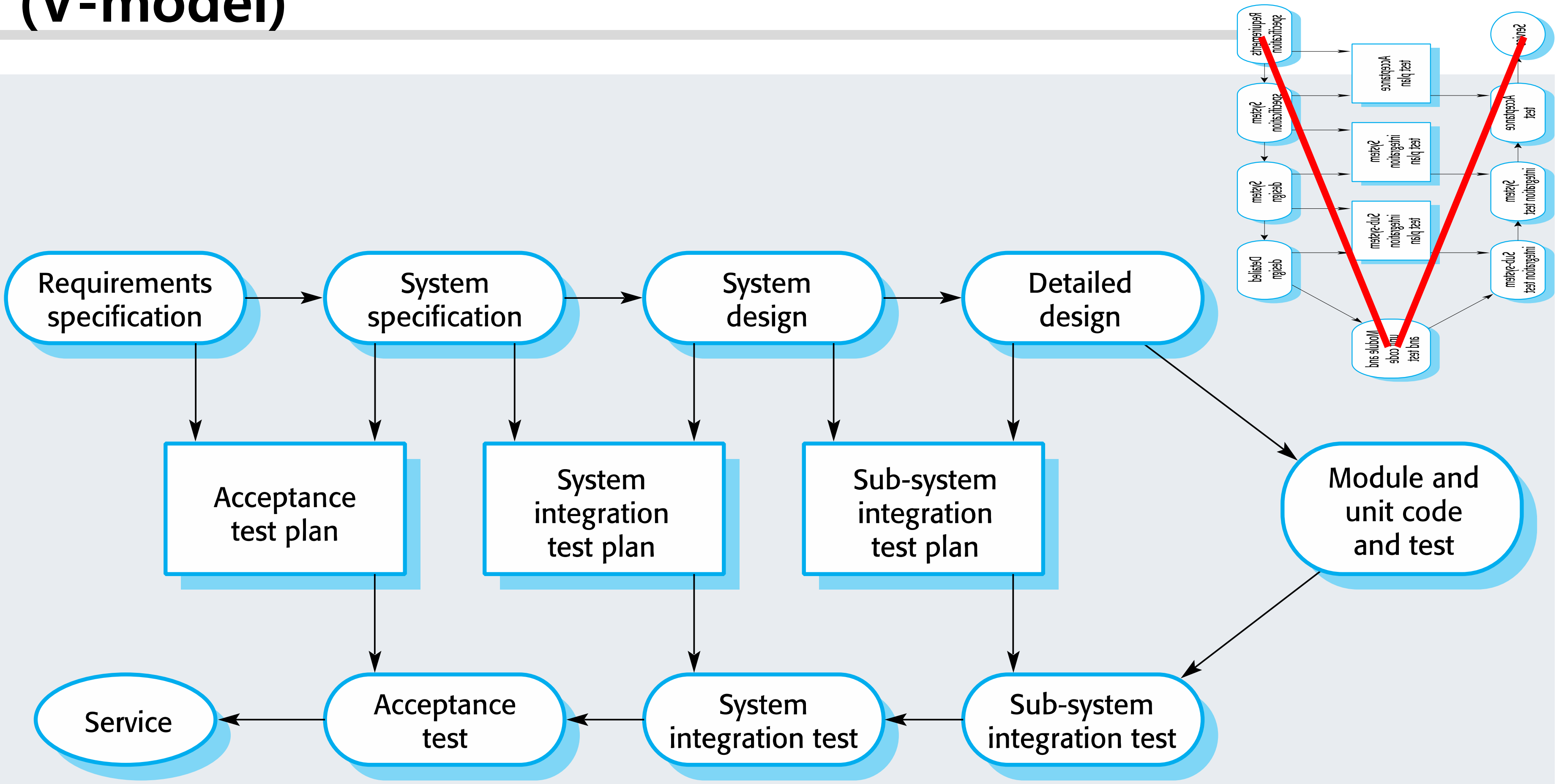
Stages of testing



Testing stages

- **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.

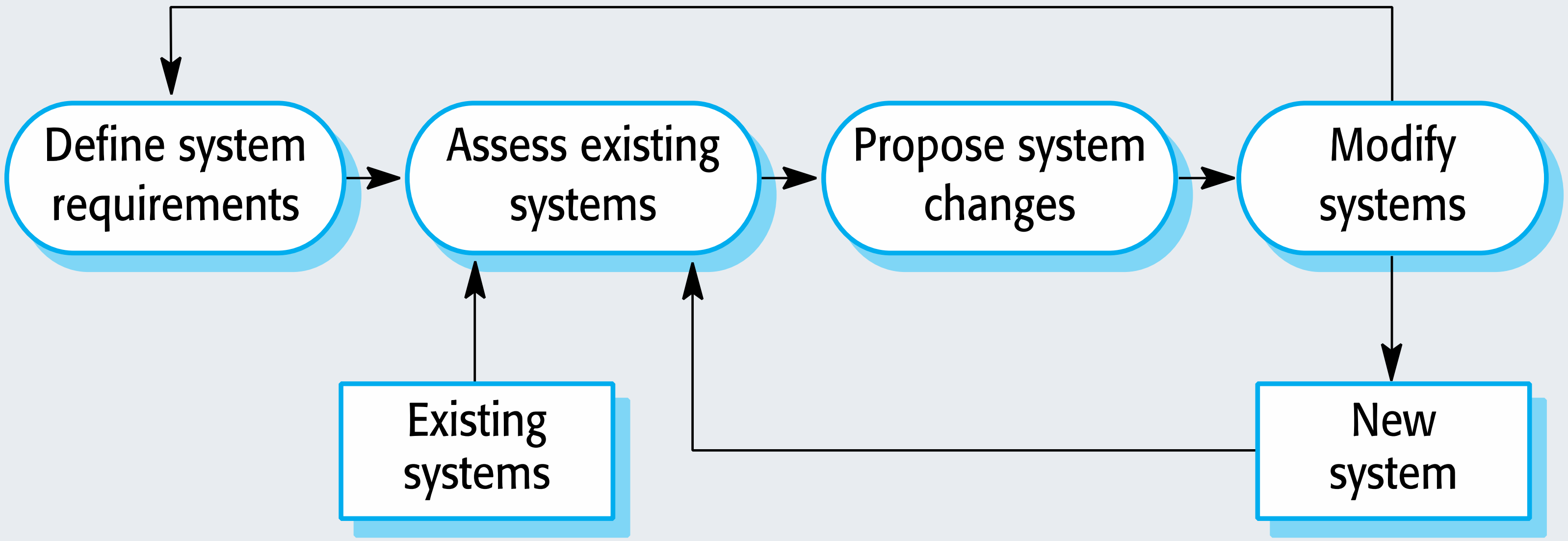
Testing phases in a plan-driven software process (V-model)


Requirements specification
System specification
System design
Detailed design
Acceptance test plan
System integration test plan
Sub-system integration test plan
Module and unit code and test
Service
Acceptance test
System integration test
Sub-system integration test

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



Key points

- Software processes are the set of coherent activities involved in producing a software system. Software process models are abstract representations of these processes.

- General process models describe the organization of software processes. Examples of these general models include the ‘waterfall’ model, incremental development, and reuse-oriented development.

Key points

- *Requirements engineering* is the process of developing a software specification. Services and Constraints.
- *Design* and *implementation* processes are concerned with transforming a requirements specification into an executable software system.
- *Software verification and validation* is the process of checking that the system conforms to its specification and that it meets the real needs of the users of the system.
- *Software evolution* takes place when you change existing software systems to meet new requirements. The software must evolve to remain useful.

Software Processes

Part 2

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 (e.g. re-analyzing requirements) as well as the costs of implementing new functionality**

Reducing the costs of rework

- **Change avoidance, where the software process includes activities that can anticipate possible changes before significant rework is required.**
  - For example, a **prototype system** may be developed to show some key features of the system to customers.
- **Change tolerance, where the process is designed so that changes can be accommodated at relatively low cost.**
  - This normally involves some form of **incremental development**. Proposed changes may be implemented in increments that have not yet been developed. If this is impossible, then only a single increment (a small part of the system) may have be altered to incorporate the change.

Coping with changing requirements

- System prototyping, where 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, where system increments are delivered to the customer for comment and experimentation. This supports both change avoidance and change tolerance.

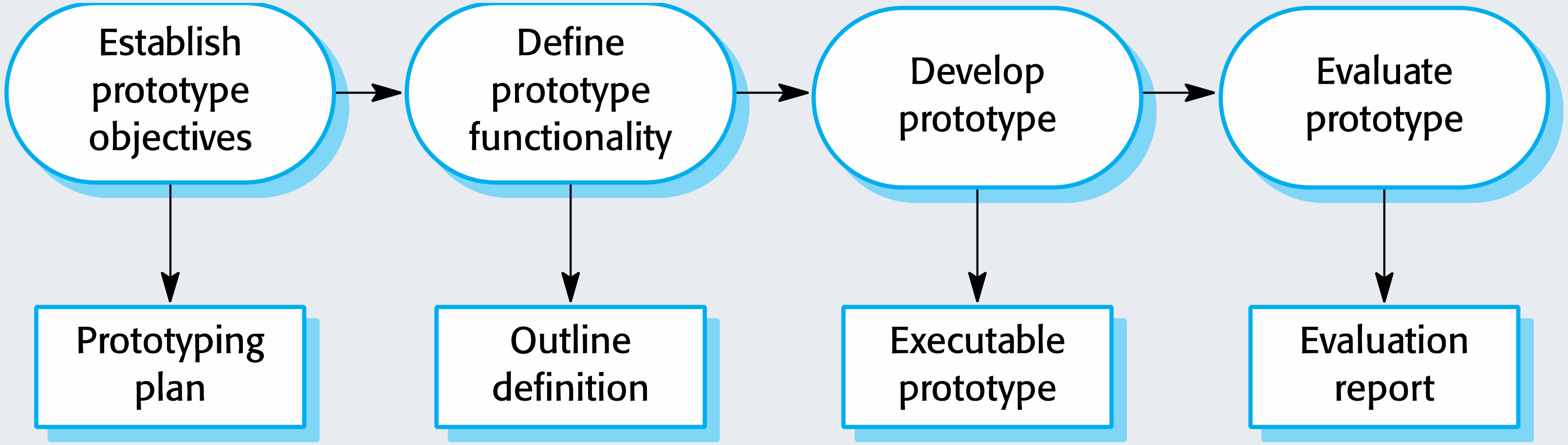
Software prototyping

- **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

Throw-away 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 organizational quality standards.

  - Not **exploratory prototypes**

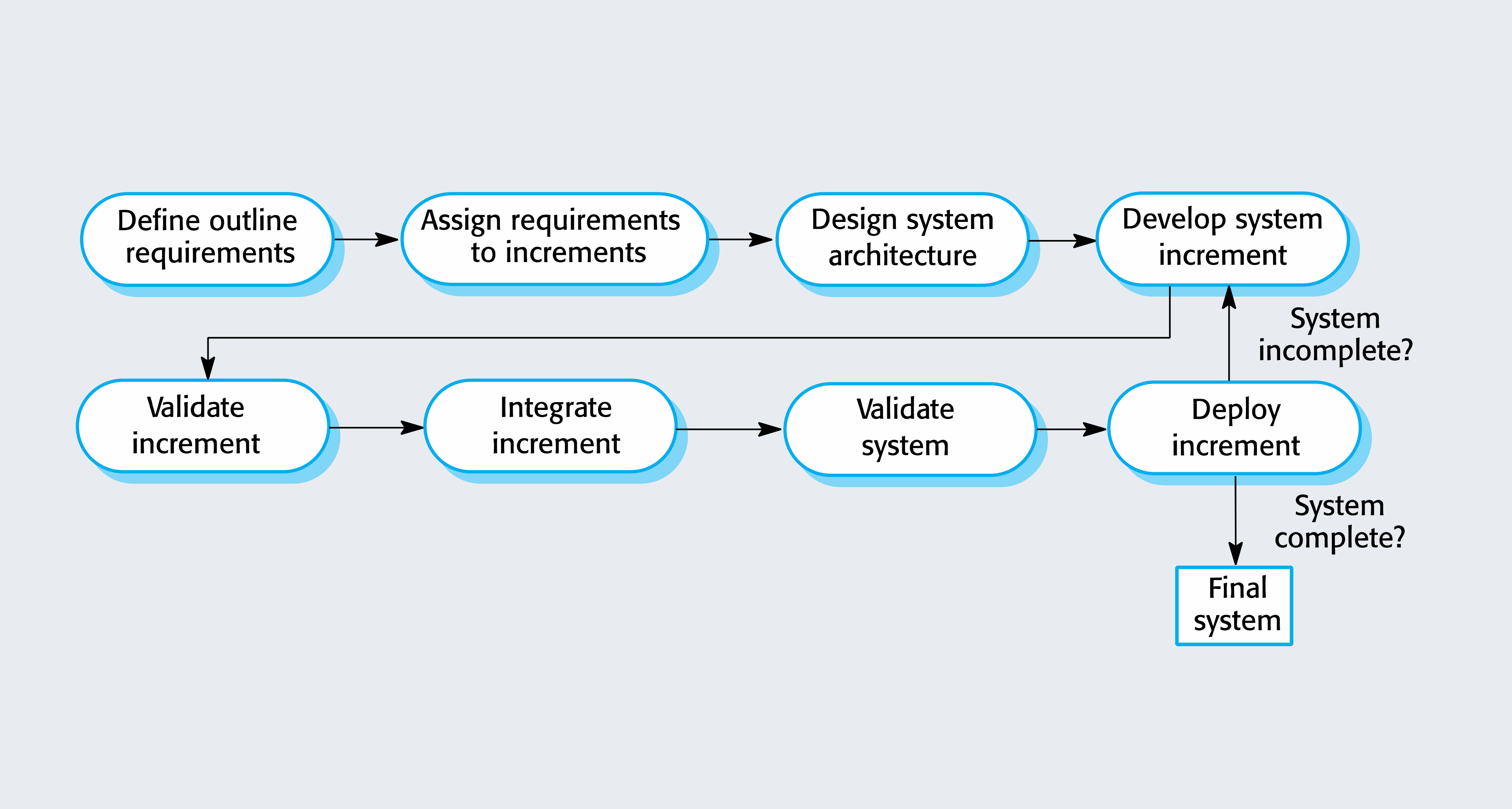
Incremental development and delivery

- **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

- 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 prioritized 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 D & D advantages

- 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 D & D 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.

Process improvement

- Many software companies have turned to software process improvement as a way of enhancing the quality of their software, reducing costs or accelerating their development processes.
- Process improvement means understanding existing processes and changing these processes to increase product quality and/or reduce costs and development time.

Approaches to improvement

- **The process maturity approach, which focuses on improving process and project management and introducing good software engineering practice.**
  - The level of process maturity reflects the extent to which good technical and management practice has been adopted in organizational software development processes.
- **The agile approach, which focuses on iterative development and the reduction of overheads in the software process.**
  - The primary characteristics of agile methods are rapid delivery of functionality and responsiveness to changing customer requirements.
- CMM/CMMI by SEI in CMU
- SPICE by ISO

Key points

- Processes should include activities *to cope with change*. This may involve a prototyping phase that helps avoid poor decisions on requirements and design.
- Processes may be structured for iterative incremental development and delivery so that changes may be made without disrupting the system as a whole.
- Many software companies have turned to software process improvement as a way of enhancing the quality of their software, reducing costs or accelerating their development processes.

Software Processes

Appendix

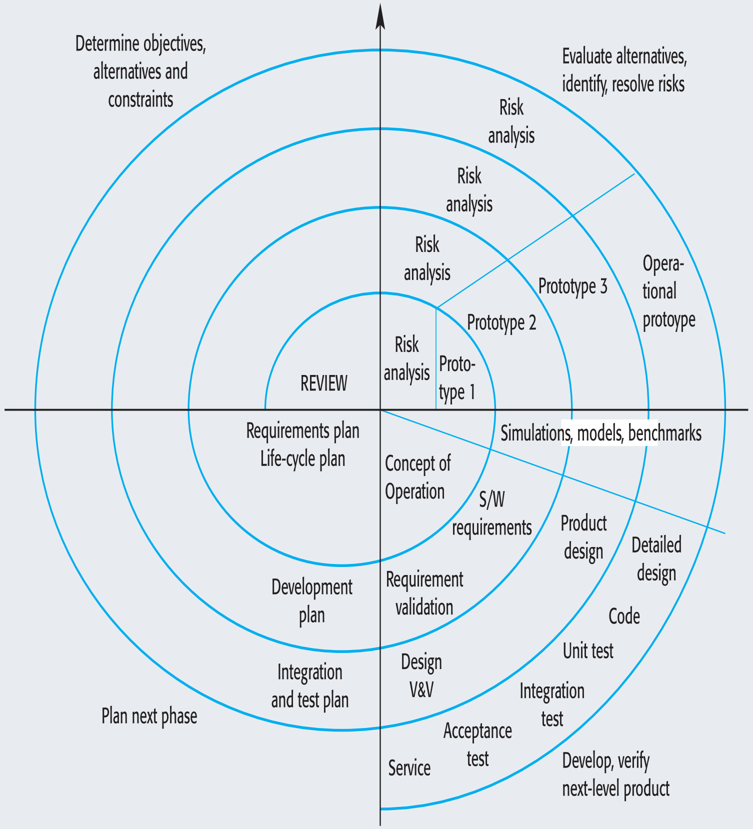
- Other Process Models:

(Boehm’s spiral model, RUP, Cleanroom model)

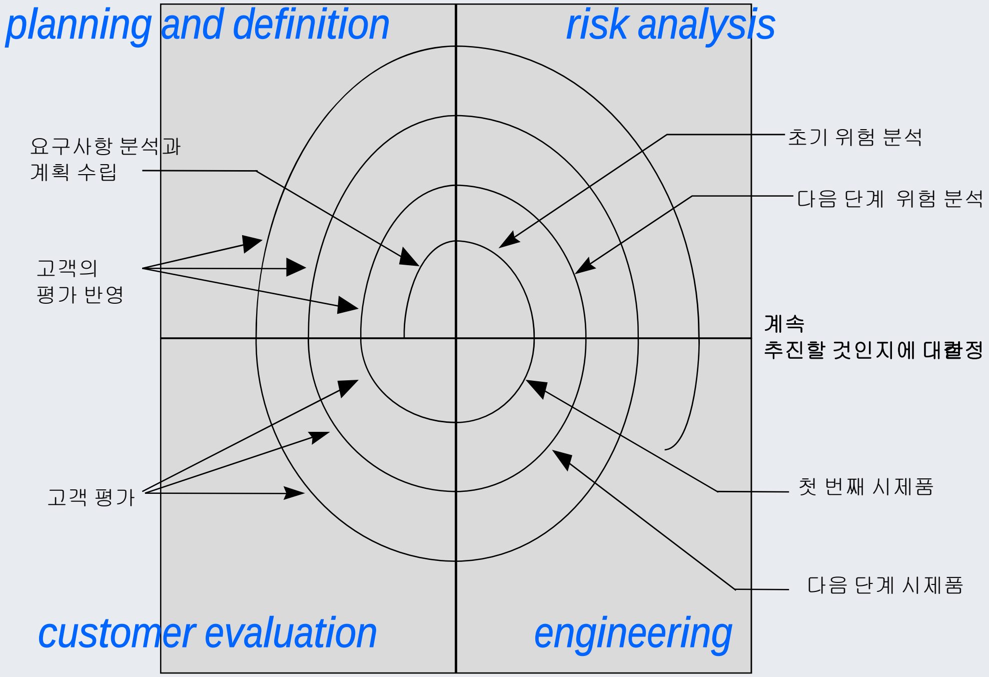
Boehm’s spiral model

- 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



< Applied case 1>



< Applied case 2 in Korea>

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.

Template for Spiral Round

- Objectives
- Constraints
- Alternatives
- Risks
- Risk resolution
- Results
- Plans
- Commitment

Example1: Quality Improvement

- Objectives
  - Significantly improve software quality
- Constraints
  - Within a three-year time-scale
  - Without large-scale capital investment
  - Without radical change to company standards
- Alternatives
  - Reuse existing certified software
  - Introduce formal specification and verification
  - Invest in testing and validation tools

Example1: Quality Improvement

- Risks
  - No cost effective quality improvement possible
  - Quality improvements may increase costs excessively
  - New methods might cause existing staff to leave

- Risk resolution
  - Literature survey
  - Pilot project
  - Survey of potential reusable components
  - Assessment of available tool support
  - Staff training and motivation seminars

Example1: Quality Improvement

- Results

  - Experience of formal methods is limited - very hard to quantify improvements
  - Limited tool support available for company standard development system.
  - Reusable components available but little reuse tool support

- Plans

  - Explore reuse option in more detail
  - Develop prototype reuse support tools
  - Explore component certification scheme

- Commitment

  - Fund further 18-month study phase

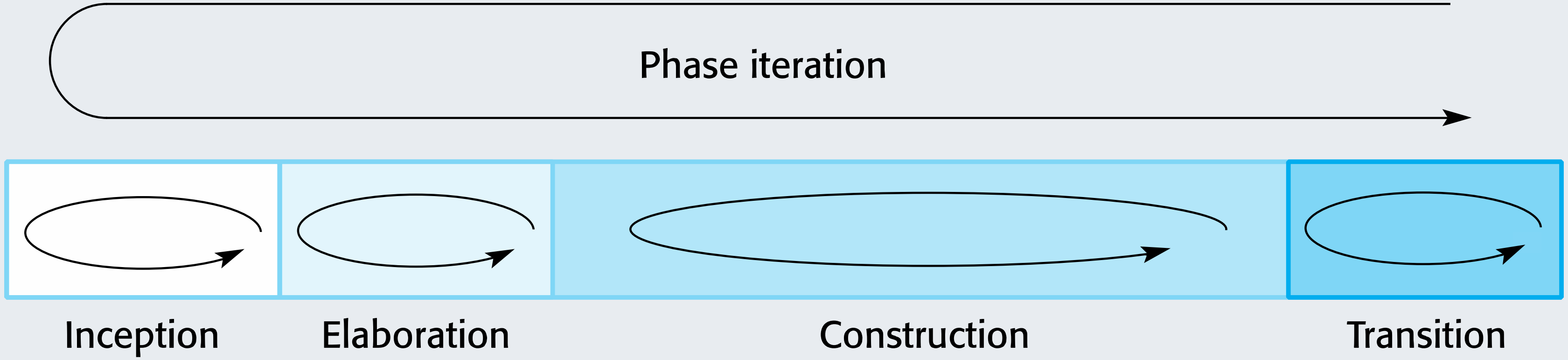
Spiral model usage

- 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

- **A modern generic process derived from the work on the UML and associated process.**
- **Brings together aspects of the three generic process models discussed previously.**
- **Normally described from three perspectives**
  - A dynamic perspective that shows phases over time;
  - A static perspective that shows process activities(workflows in RUP);
  - A practice perspective that suggests good practice to be used during the process.

Phases in the Rational Unified Process



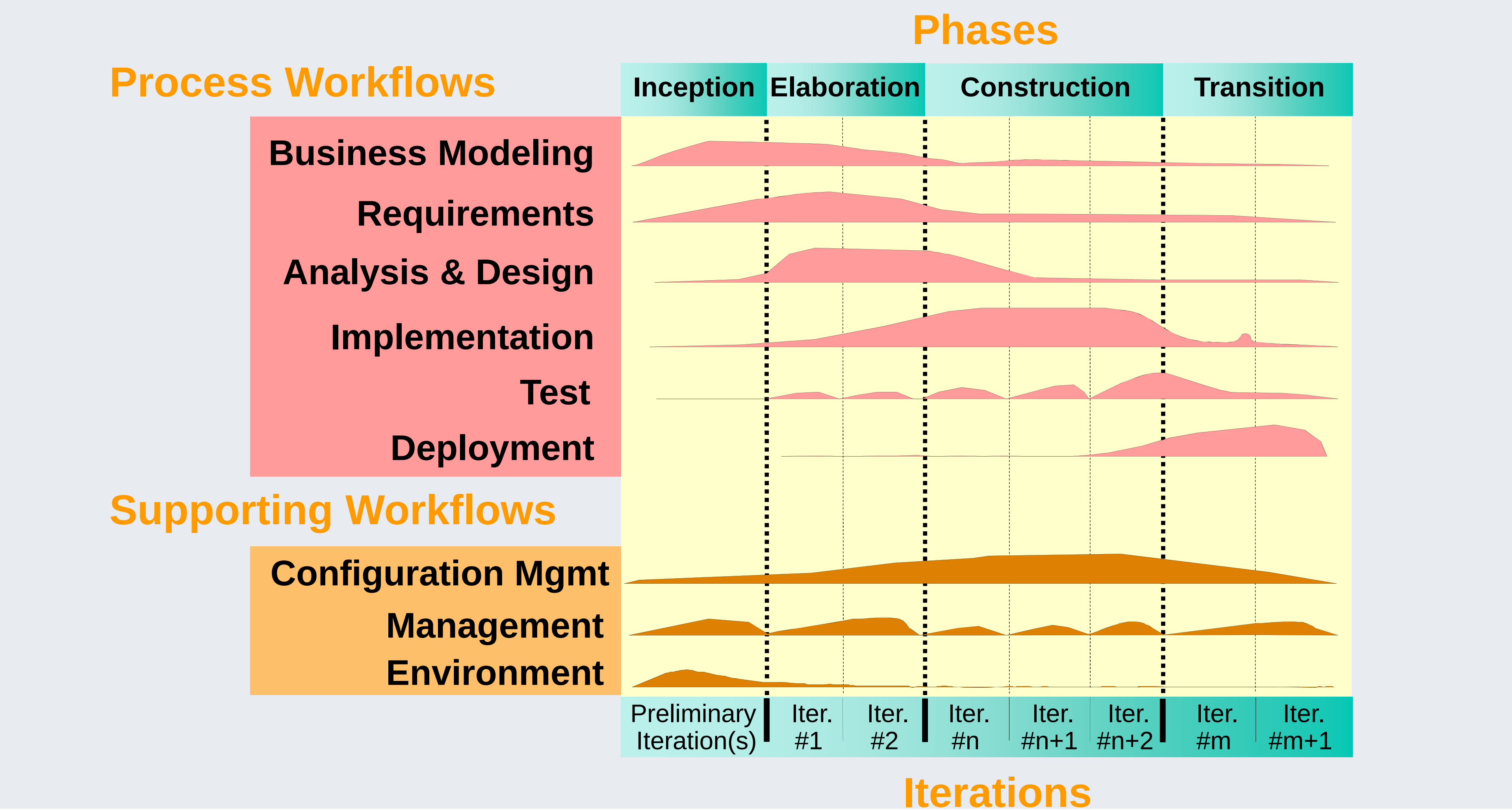
RUP phases

- **Inception(개념화 단계)**
  - Establish the *business case* for the system.
  - Identify all external entities and interactions (context diagram)
  - Assess the contribution that the system makes to the business
- **Elaboration(초기 아키텍처 구축단계)**
  - Develop an understanding of the problem domain and the system architecture.
  - Develop project plan and identify key project risks
  - Req. model, architectural description, development plan
- **Construction(시스템 완성 단계)**
  - System design, programming and testing (parallel dev. & integration)
  - Working software, associated documents
- **Transition(시스템 이전 단계)**
  - Deploy the system in its operating environment.
  - Documented working SW system in its operational environment

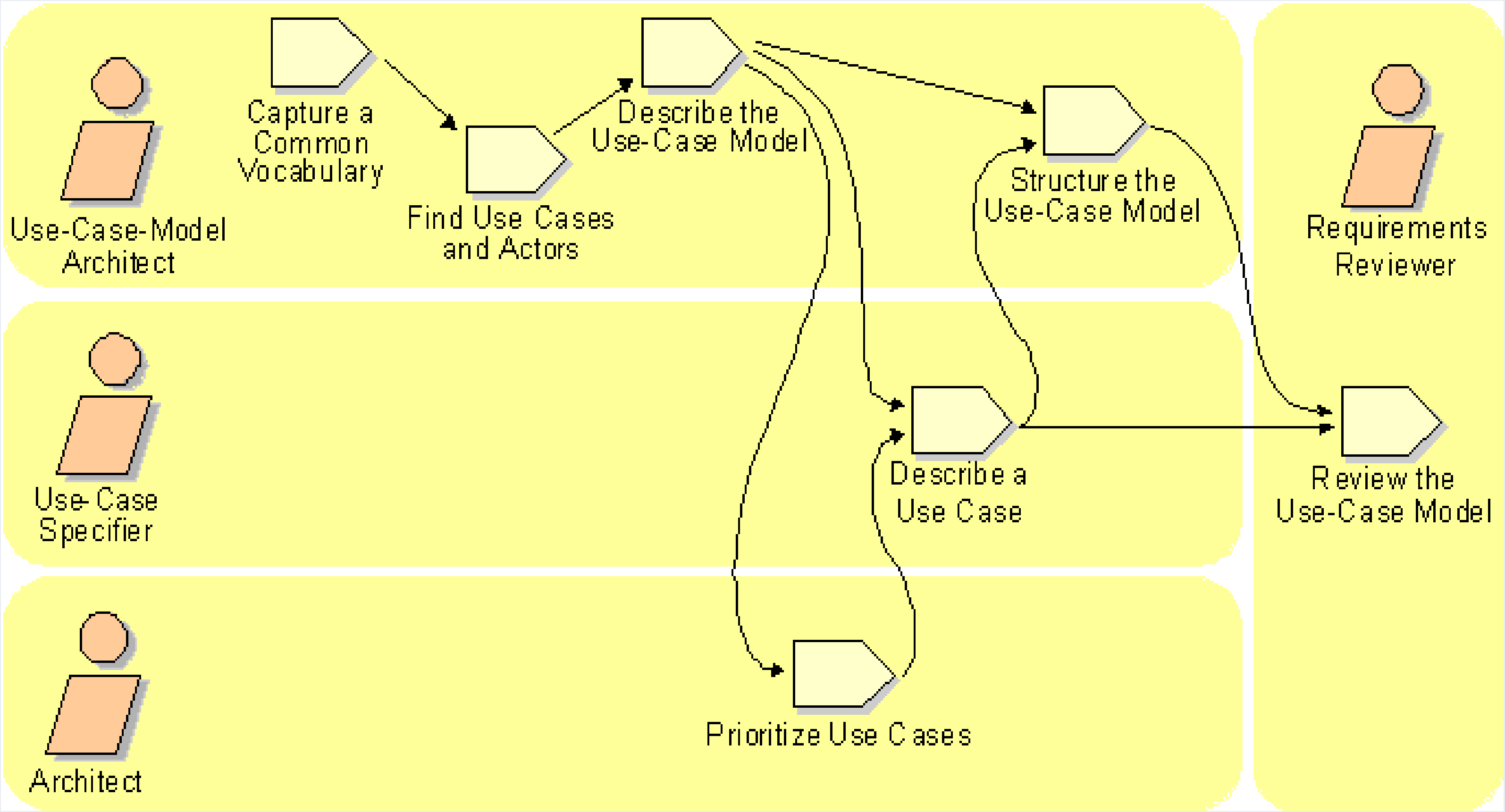
RUP iteration

- **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.

Static workflows in the Rational Unified Process



Workflows in requirement capture


Use-Case-Model Architect
Capture a Common Vocabulary
Find Use Cases and Actors
Describe the Use-Case Model
Structure the Use-Case Model
Requirements Reviewer
Use-Case Specifier
Describe a Use Case
Review the Use-Case Model
Architect
Prioritize Use Cases

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 system (see Chapter 25).
Project management	This supporting workflow manages the system development (see Chapters 22 and 23).
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.

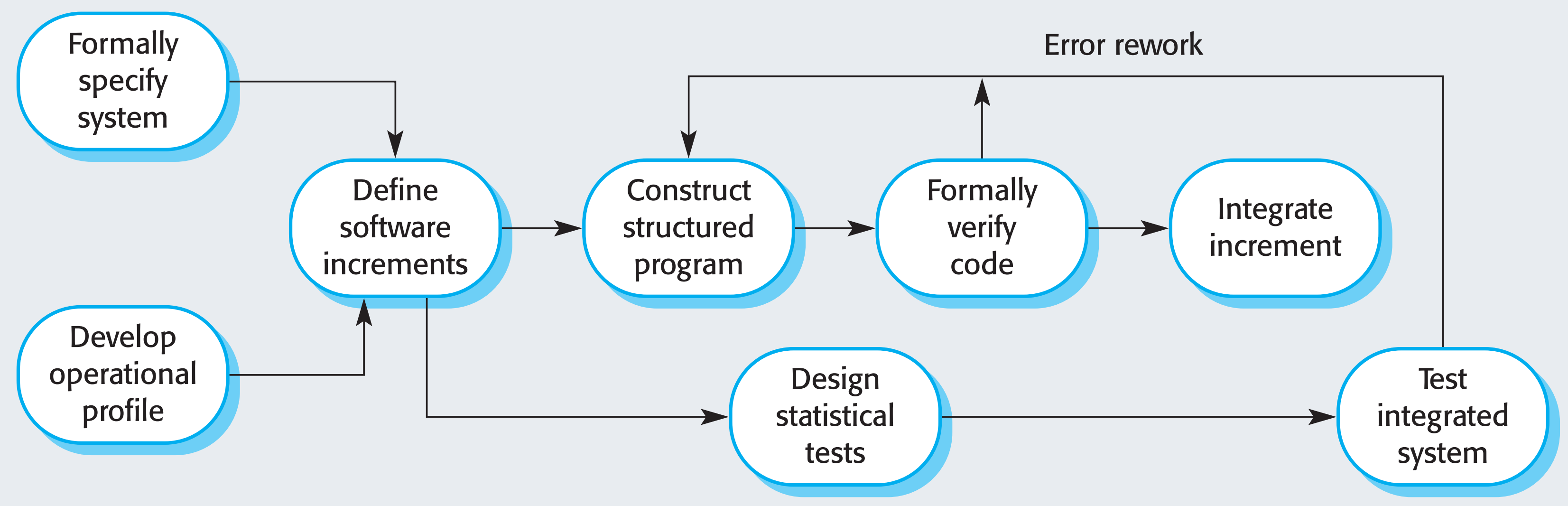
RUP good practice

- **Visually model software**
  - Use graphical UML models to present static and dynamic views of the software.
- **Verify software quality**
  - Ensure that the software meets organizational quality standards.
- **Control changes to software**
  - Manage software changes using a change management system and configuration management tools.

Cleanroom SW development model

- The name is derived from the 'Cleanroom' process in semiconductor fabrication. The philosophy is defect avoidance rather than defect removal. (Mills, et al., 1987)
- This software development process is based on:
  - Incremental development;
  - Formal specification;
  - Structured programming;
  - Static verification using rigorous SW inspections;
  - Statistical testing to determine program reliability.

Cleanroom SW development model


Formally specify system
Develop operational profile
Define software increments
Construct structured program
Formally verify code
Integrate increment
Error rework
Design statistical tests
Test integrated system