Module 2: Software Process Model

Software Engineering

Faculty of Computing / MJIIT

Universiti Teknologi Malaysia

Outline

- Software process
  - Plan-driven and Agile process
- Software process models
  - Waterfall model
  - Incremental model (inc. Boehm’s Spiral model)
  - Integration and configuration
  - *The Rational Unified Process (RUP)*
- Process activities
- Coping with change
- Process improvement

Note: The overall contents of the slide are based on the main reference that is Sommerville (2016) with other references specified directly in respective slides

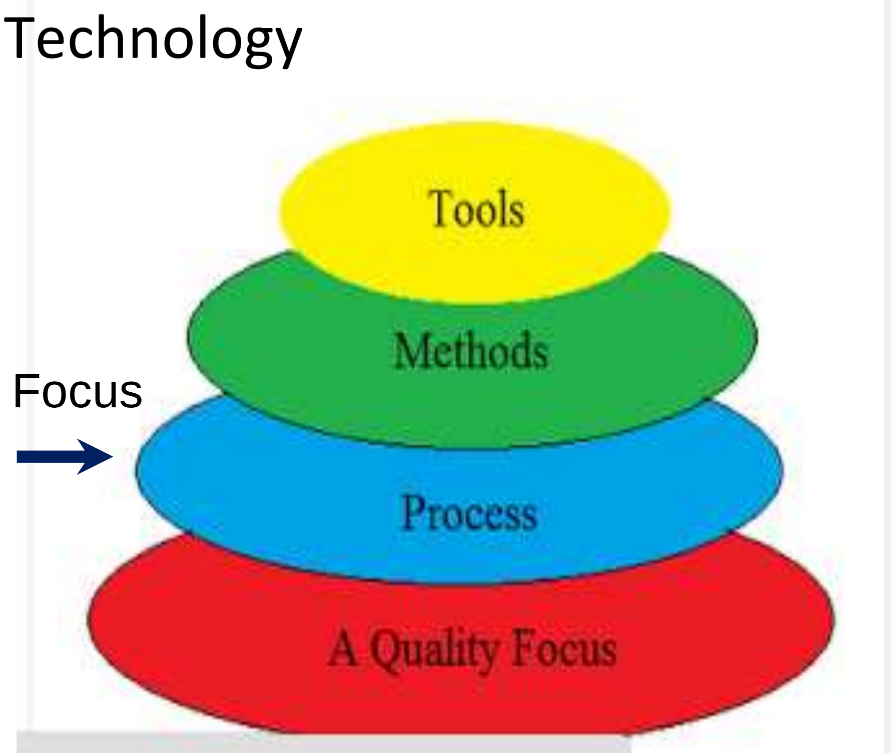
Objectives

The objectives of this module are:

- To understand software process, its activities and diverse software process models
- To recognise the relation between plan-driven and Agile process
- To differentiate the differences among software process models and their uses
- To understand spiral model for coping changes and the Rational Unified Process

Recap: What is Software Engineering?

Software Engineering as Layered Technology



- **Quality Focus:** The bedrock that supports software engineering is quality focus where it ensures continuous process improvement culture.
- **Process :**Foundation for software engineering which enables rational and timely development of computer software
- **Methods:** provide technical how to's for building software. Involve different tasks including requirements analysis, design, program construction, testing and support. Methods also include modeling activities
- **Tools:** provide automated or semi-automated support for the process and methods

** The layered technology will be the focus in this lecture and throughout the semester

Source:

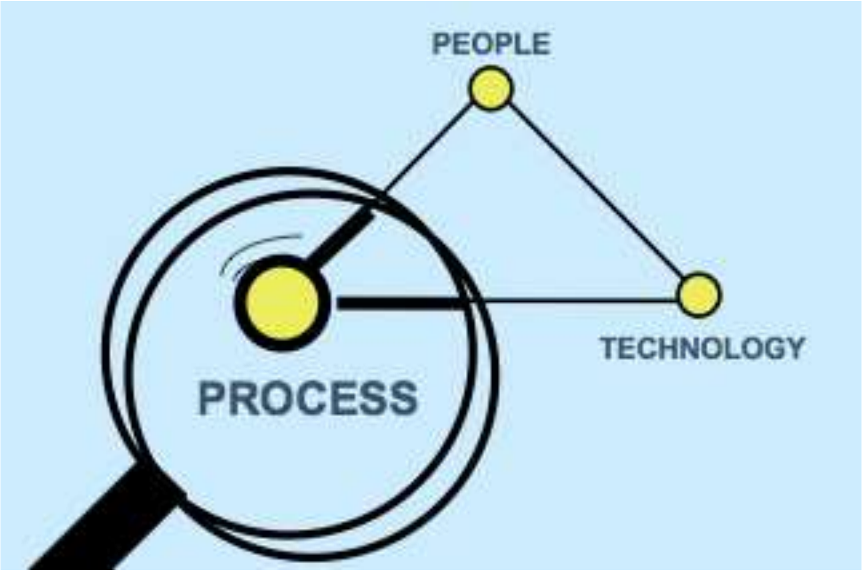
Agarwal, U. (2019). Software Engineering, Kataria and Sons.

https://www.geeksforgeeks.org/layered-technology-in-software-engineering/

Software Process

The Role of Process

Even the finest people cannot perform at their best when the process is not understood or operating at its best



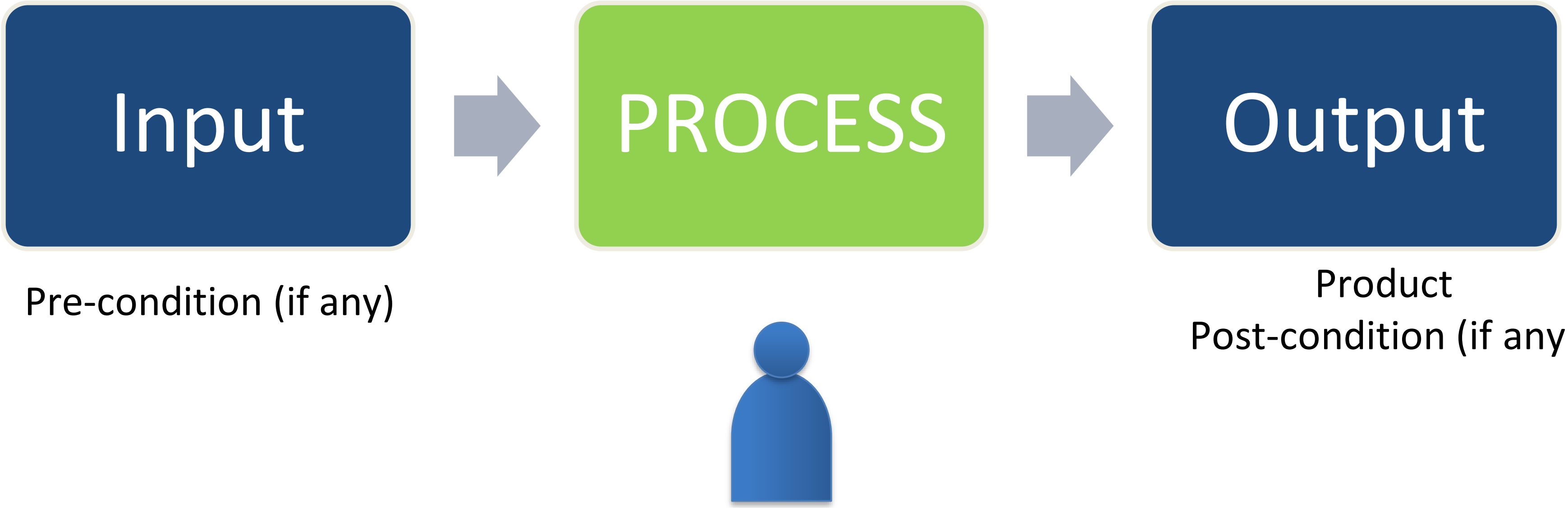
The Software Process

- A structured set of activities required to develop a software system
- Many different software processes but all involve (SDIVE):
  - **S**pecification – defining what the system should do
  - **D**esign and **I**mplementation – defining the organization of the system and implementing the system
  - **V**alidation – checking that it does what the customer wants
  - **E**volution – 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 perspectives.

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 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

Process Descriptions



Input

PROCESS

Output

Pre-condition (if any)

Product
Post-condition (if any)

Roles

Plan-Driven and Agile Processes

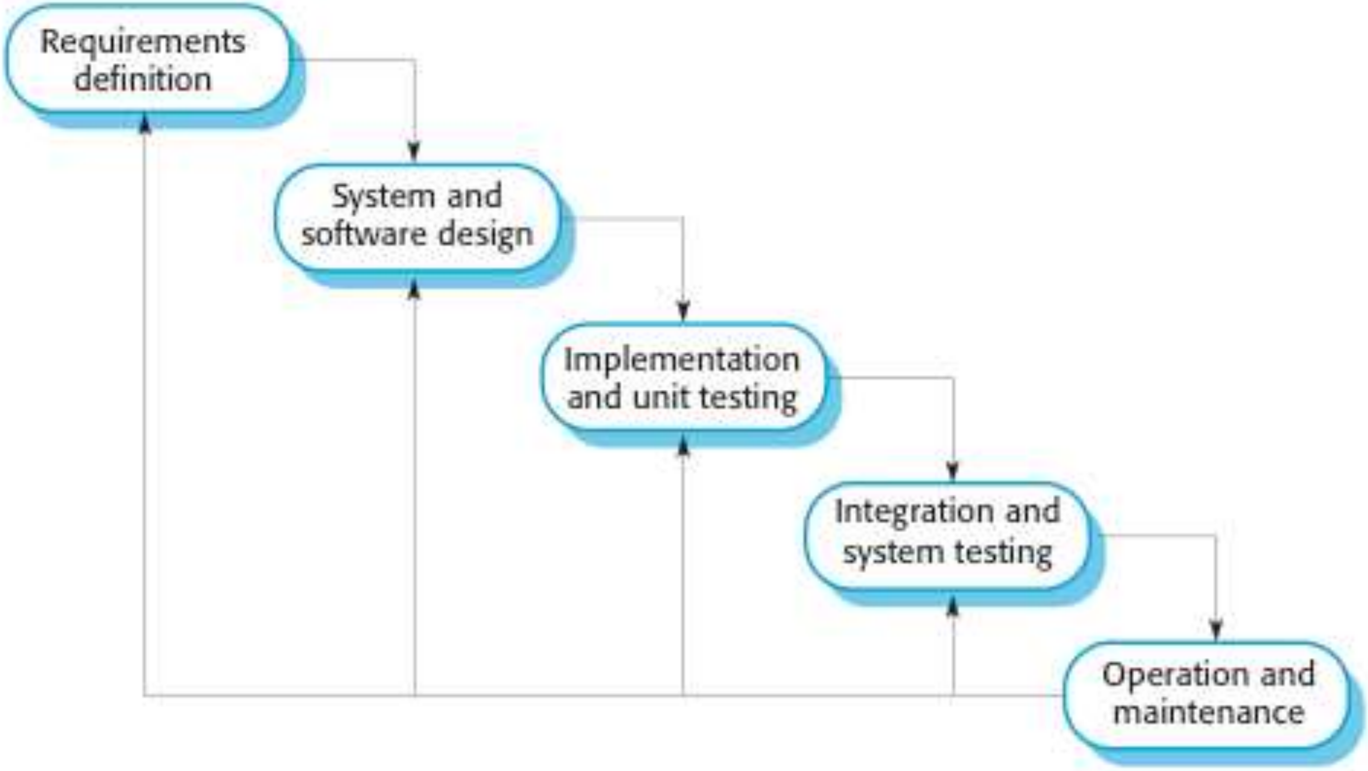
- Plan-driven processes are processes where all of the process activities are **planned in advance** and progress is measured against this plan
- In agile processes, **planning is incremental,** and it is easier to change the process to reflect changing customer requirements
- In practice, most practical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

Software Process Models

Software Process Models

- (1) The waterfall model
  - Plan-driven model, separate and distinct phases of specification and development
- (2) Incremental development
  - Specification, development and validation are interleaved; may be plan-driven or agile
- (3) Integration and configuration
  - 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

(1) The Waterfall Model


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

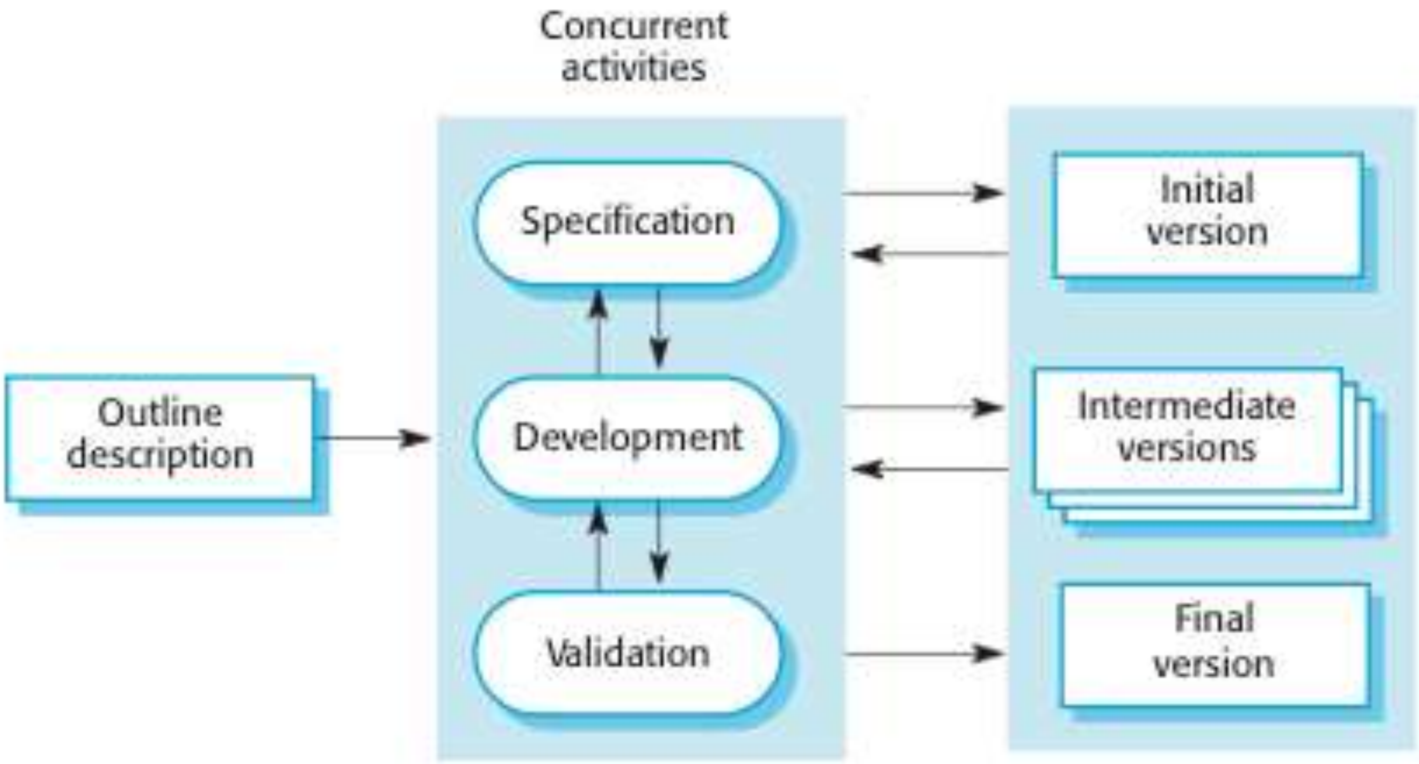
Waterfall Model Phases

- 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 embedded systems** (software must interface with hardware), **critical systems, and large systems** (system is developed at several sites)
  - In those circumstances, the **plan-driven** nature of the waterfall model helps coordinate the work

(2) Incremental Development



Incremental Development

- In a plan-driven approach, the system increments are identified in advance
- If an agile approach is adopted, the early increments are identified, but the development of later increments depends on progress and customer priorities
- Each increment or version of the system incorporates some of the functionality that is needed by the customer
- Generally, the early increments of the system include the most important or most urgently required functionality.

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

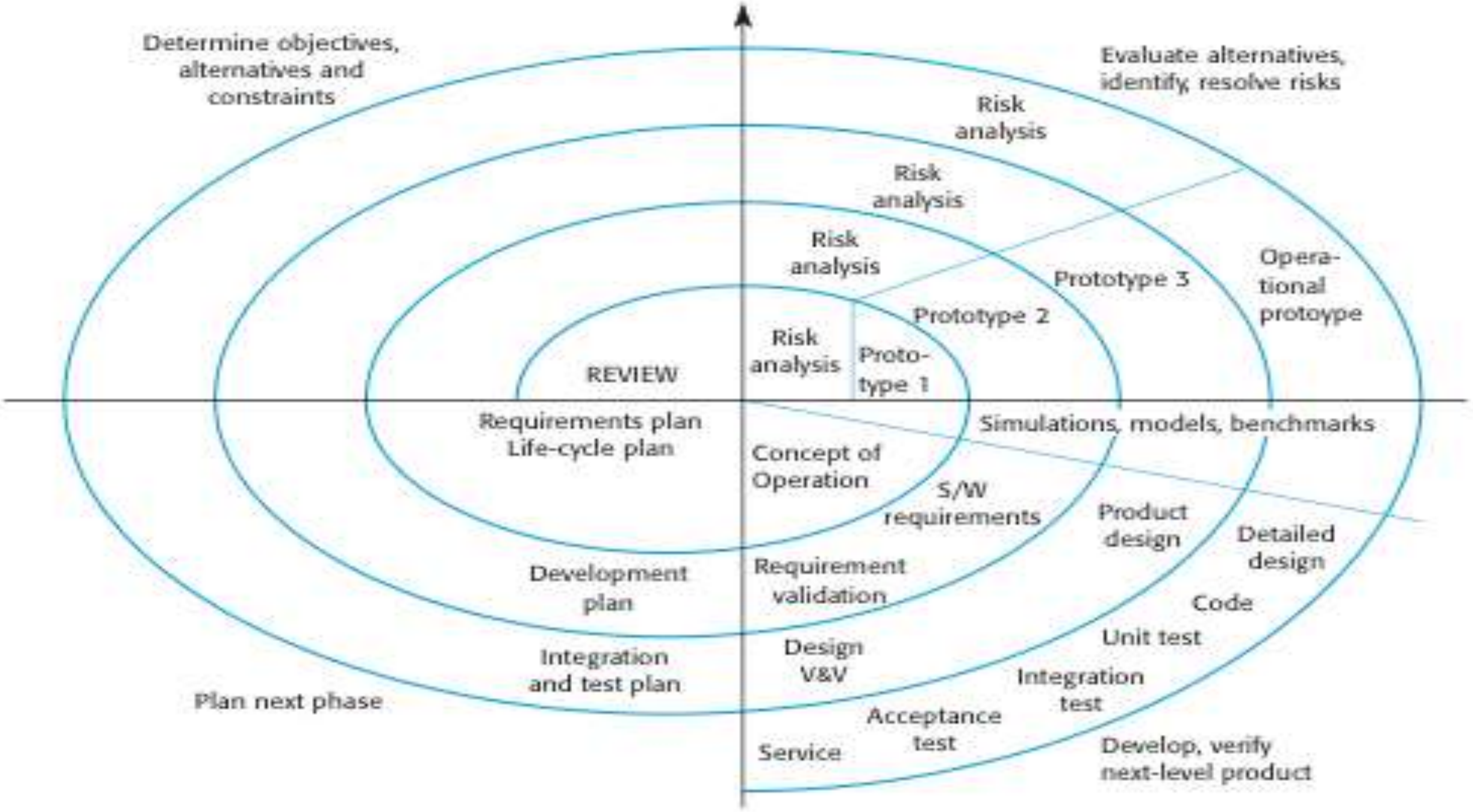
Incremental Development Problems

- 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

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



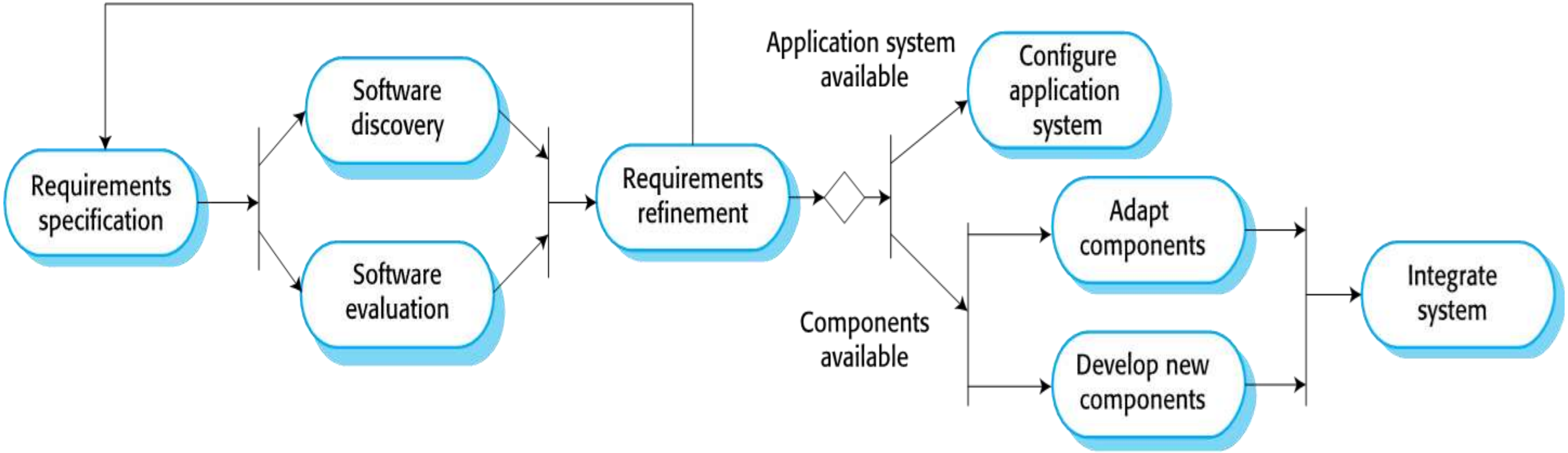
Spiral Model Sectors

- 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

(3) Integration and configuration



Integration and configuration

- 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

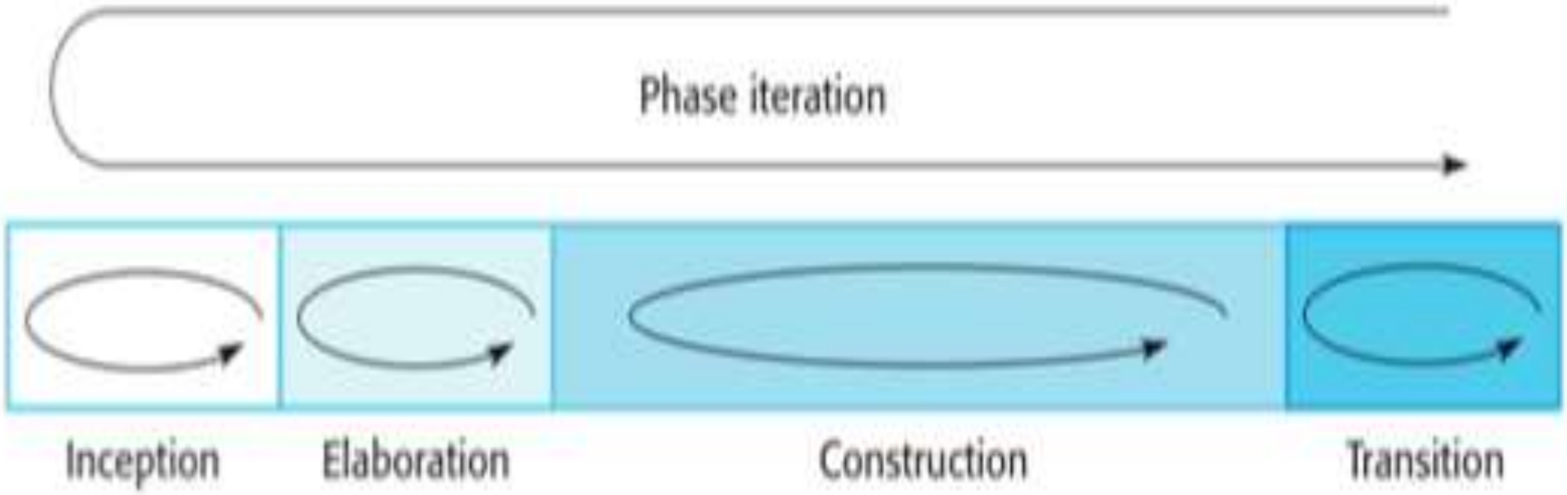
Types of Reusable Software Components

- **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

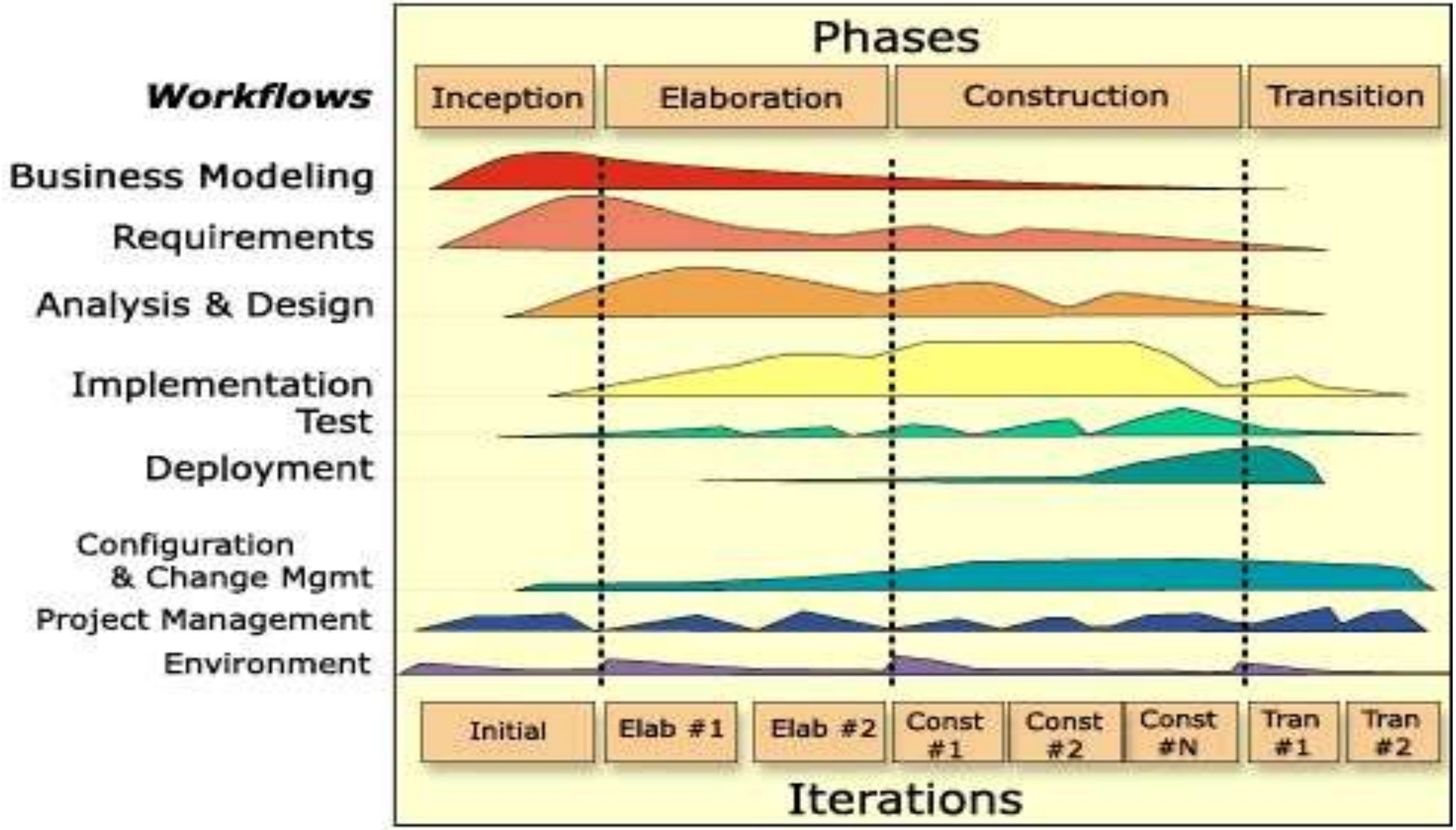
Extra: Rational Unified Process (RUP)

- 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 **practice** perspective that suggests good practice

Phases in the RUP



Iterative and Incremental Process in RUP



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

Static Workflows in the RUP…

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 RUP

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 manages changes to the system.
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

Process Activities

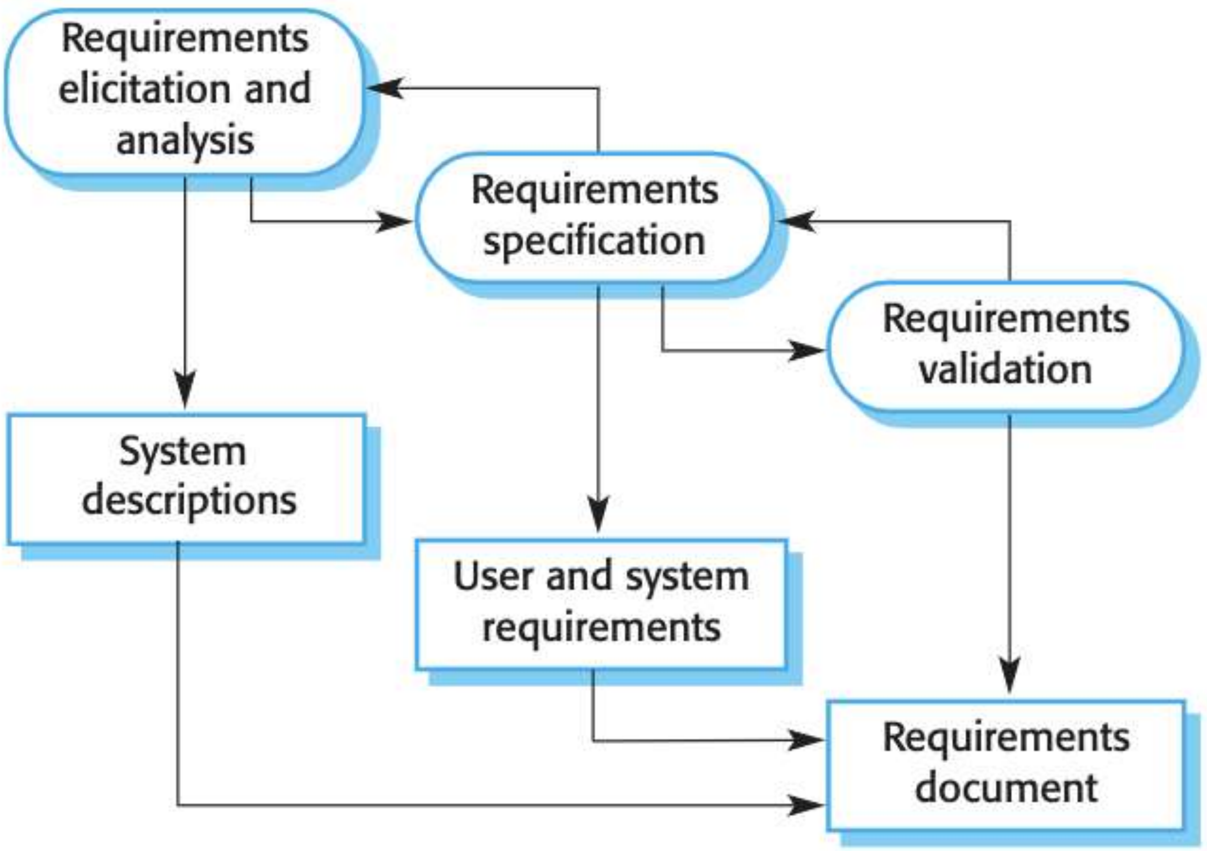
Process Activities

- 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 **(i) specification, (ii) development, (iii) validation and (iv) 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

(i) 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 (short term)
    - Is it technically and financially feasible to build the system?
  1. Requirements elicitation and analysis
     - What do the system stakeholders require or expect from the system?
  2. Requirements specification
     - Defining the requirements in detail
  3. Requirements validation
     - Checking the validity of the requirements

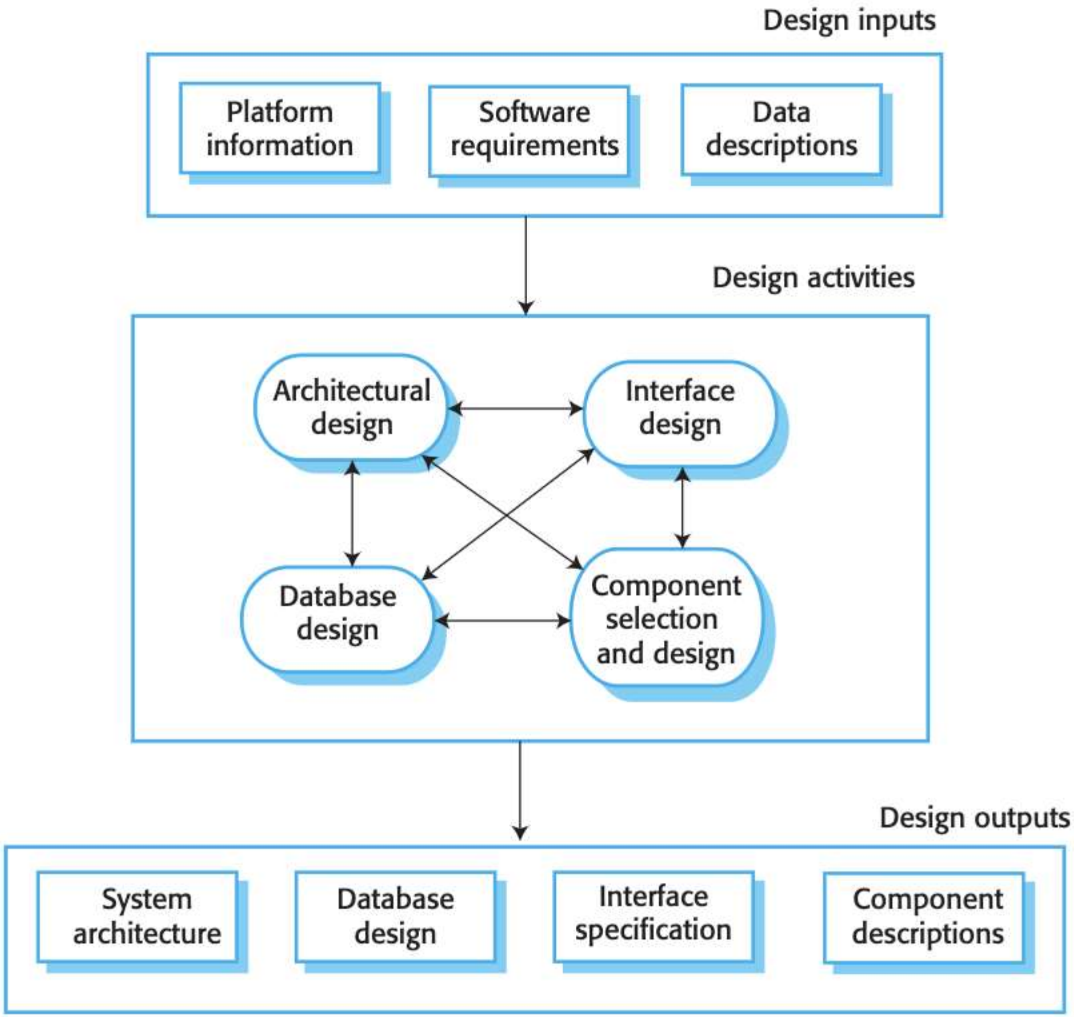
The Requirements Engineering Process


Requirements elicitation and analysis
Requirements specification
Requirements validation
System descriptions
User and system requirements
Requirements document

(ii) 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



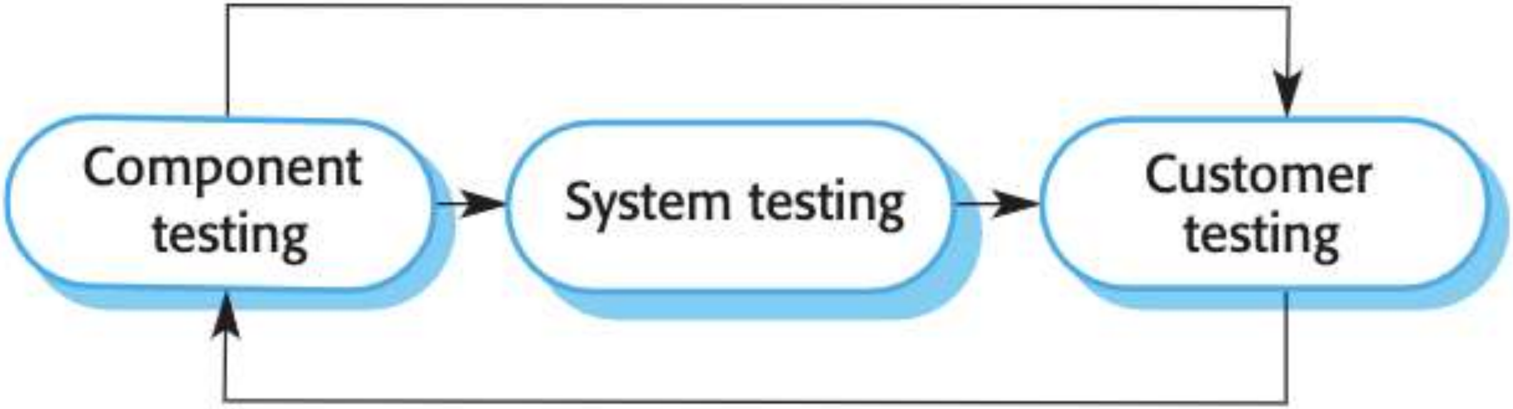
Design Activities

- **Architectural design:** 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:** define the interfaces between system components
- **Component design:** take each system component and design how it will operate
- **Database design:** design the system data structures and how these are to be represented in a database

(iii) 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

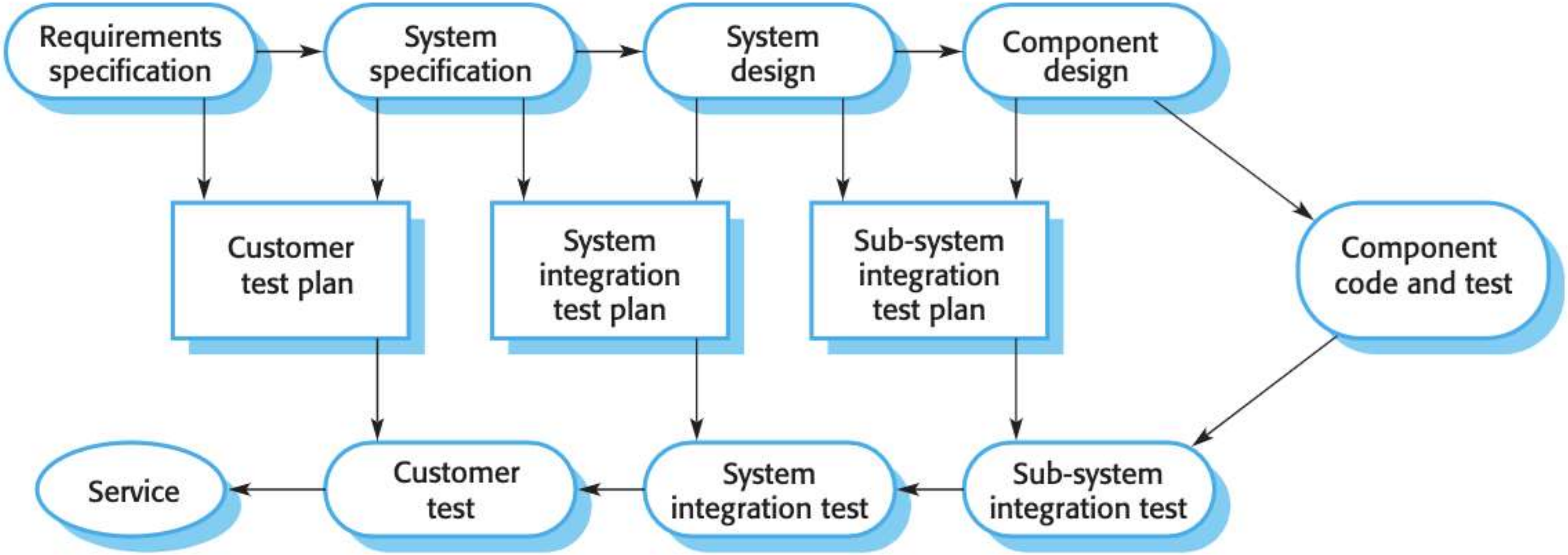
Stages of Testing



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
- Customer testing
  - Testing with customer data to check that the system meets the customer’s needs before the system is accepted for operational use

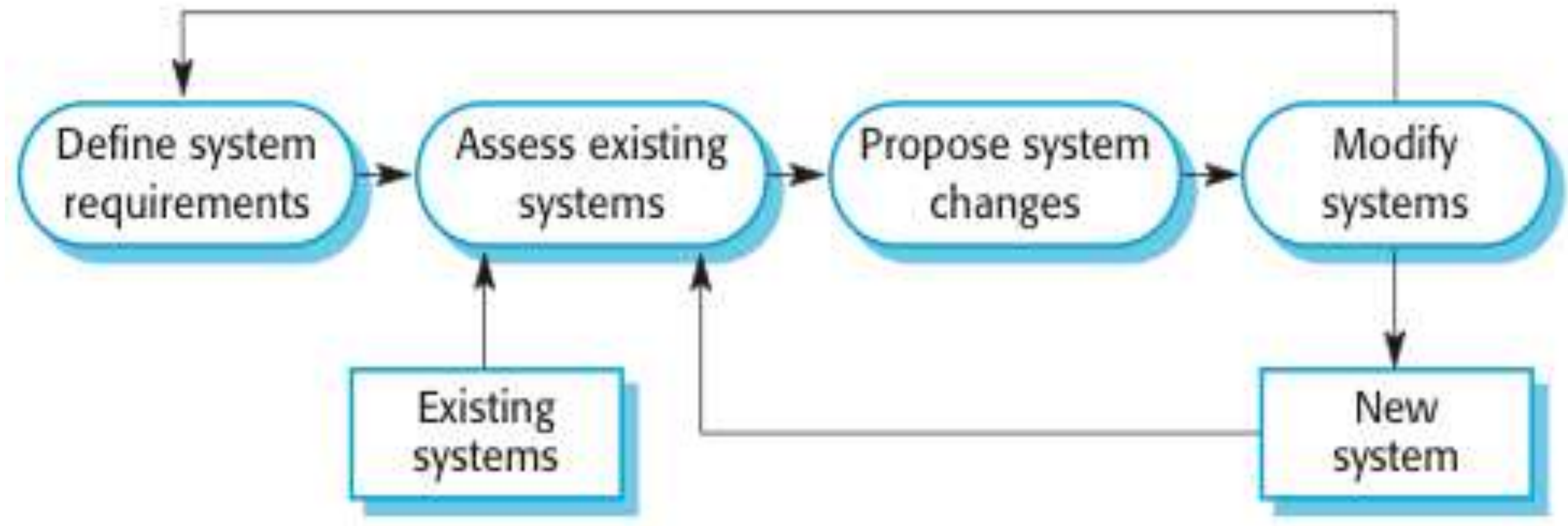
Testing Phases in a **Plan-Driven** Software Process



(iv) 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 (separation) between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new, and it makes much more sense to see development and maintenance as continuum

System Evolution



Coping with Change

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

Approach to Reduce Costs of Rework

1. Change **anticipation**, where the software process includes activities that can anticipate possible changes before significant rework is required
   - E.g. a **prototype** system may be developed to show some key features of the system to customers
2. 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,** which 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

Ways to cope: (1) Software Prototyping

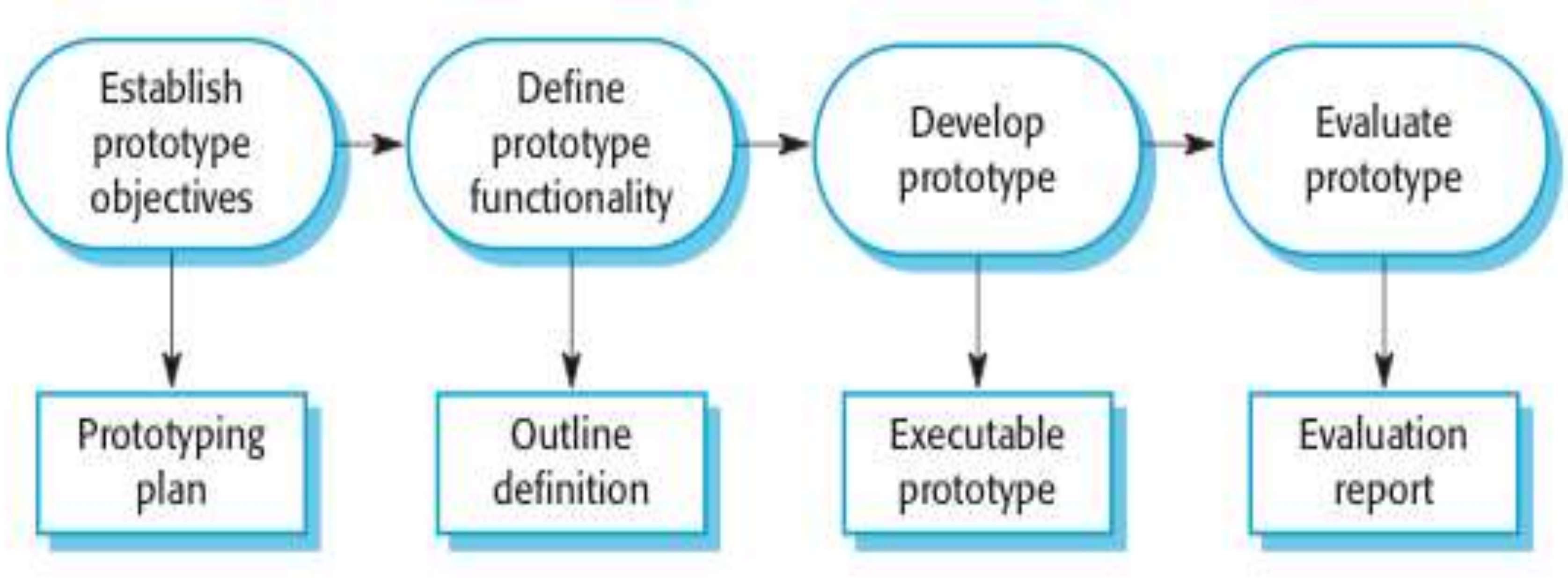
- A prototype is an initial version of a system used to demonstrate concepts and try out design options.

- 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

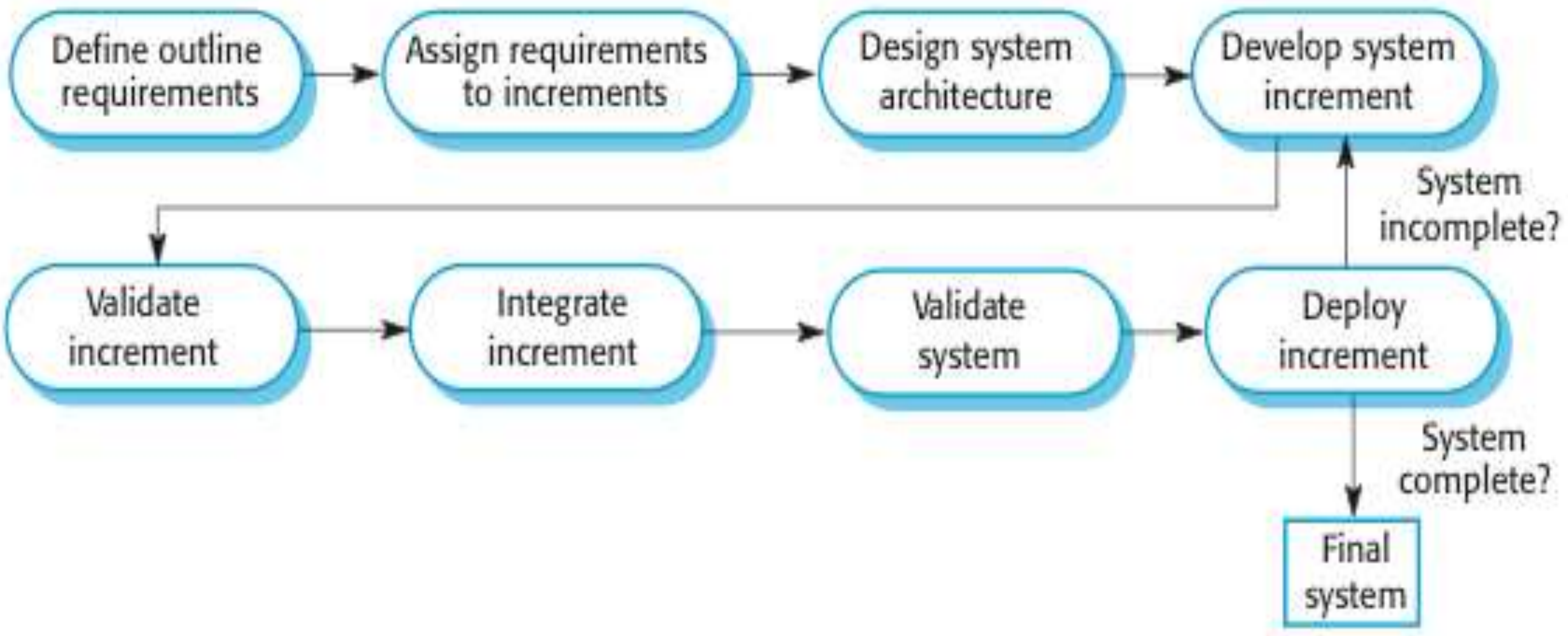
Ways to cope:
(2) Incremental Delivery

- 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 **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 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 Delivery Problems

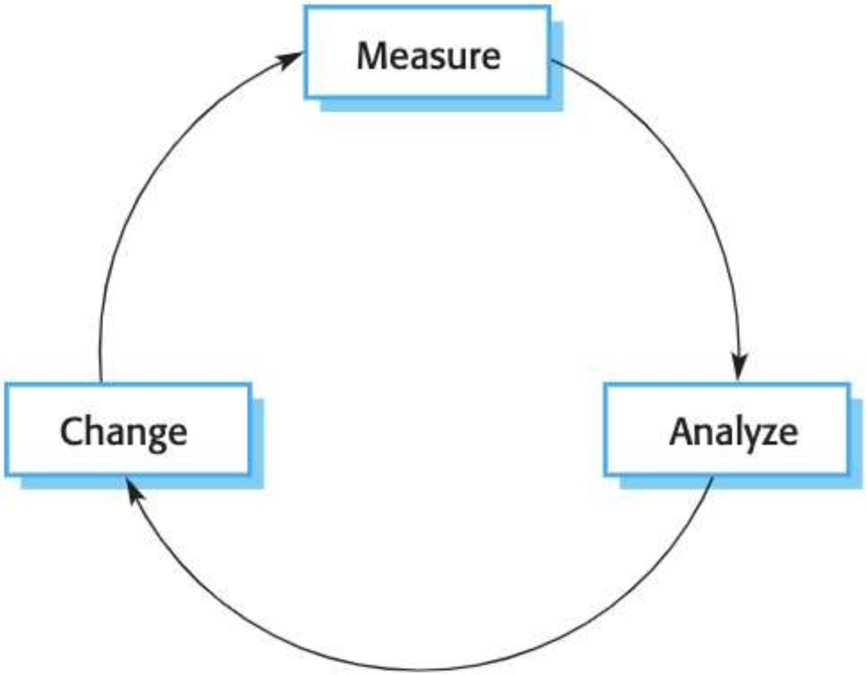
- Most systems require a set of basic facilities that are used by different parts of the system
  – As requirements are not defined in detail until an increment is to be implemented, it can be **hard to identify common facilities** that are needed by all increments
- The essence of iterative processes 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

Process Improvement

Process Improvement

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

Process Improvement Cycle



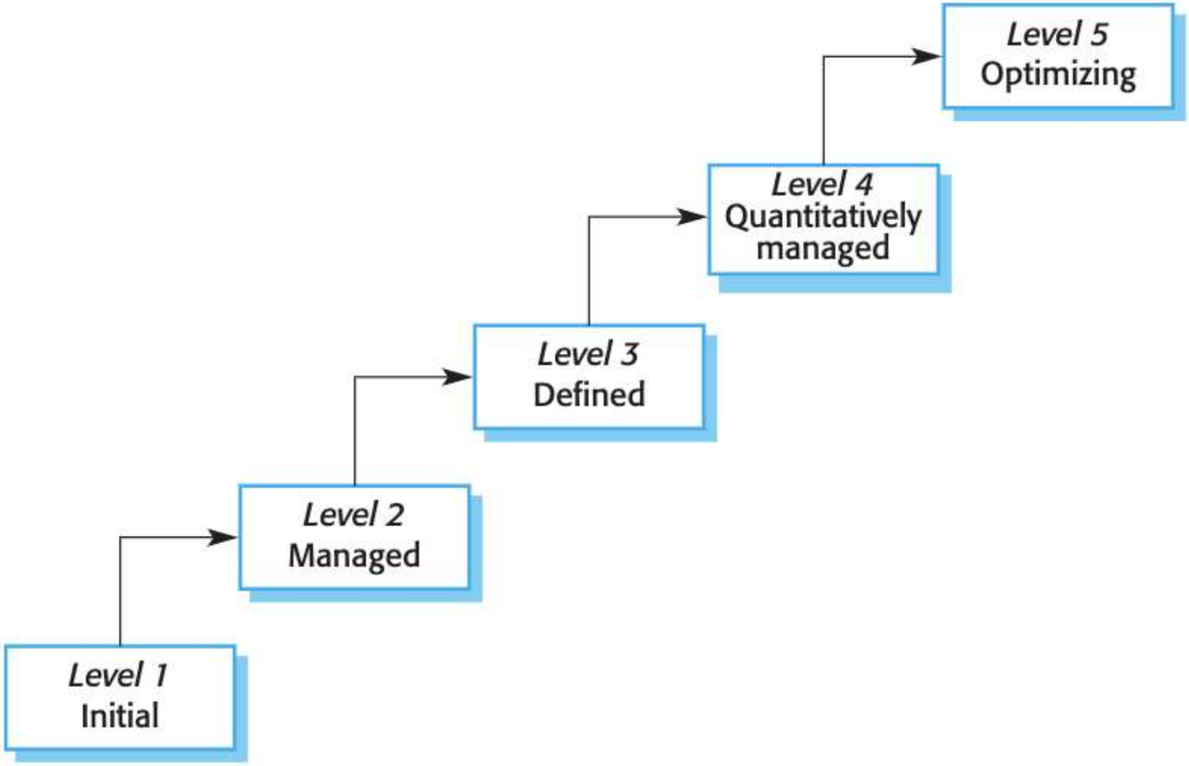
Process Improvement Cycle

- **Process measurement:** measure one or more attributes of the software process or product. These measurements form a baseline that helps to decide if process improvements have been effective. As improvements are introduced, re-measure the same attributes, which will hopefully have improved in some way
- **Process analysis:** The current process is assessed, and process weaknesses and bottlenecks are identified. Process models (sometimes called process maps) that describe the process may be developed during this stage. The analysis may be focused by considering process characteristics such as rapidity and robustness.
- **Process change:** Change are proposed to address some of the identified process weaknesses. These are introduced, and the cycle resumes to collect data about the effectiveness of the changes.

2 Approaches

1. Process maturity approach
   - focus on improving process and project management and introducing good software engineering practice into an organization
   - The level of process maturity reflects the extent to which good technical and management practice has been adopted in organizational software development processes
   - The primary goals of this approach are improved product quality and process predictability
2. Agile approach
   - focus on iterative development and the reduction of overheads in the software process
   - The primary characteristics are rapid delivery of functionality and responsiveness to changing customer requirements
   - The improvement philosophy: the best processes are those with the lowest overheads and agile approaches can achieve this

Capability Maturity Levels



Capability Maturity Levels

1. **Initial:** The goals associated with the process area are satisfied, and for all processes the scope of the work to be performed is explicitly set out and communicated to the team members.
2. **Managed:** At this level, the goals associated with the process area are met, and organizational policies are in place that define when each process should be used. There must be documented project plans that define the project goals. Resource management and process monitoring procedures must be in place across the institution.
3. **Defined:**This level focuses on organizational standardization and deployment of processes. Each project has a managed process that is adapted to the project requirements from a defined set of organizational processes. Process assets and process measurements must be collected and used for future process improvements.

Capability Maturity Levels

4. **Quantitatively managed:** At this level, there is an organizational responsibility to use statistical and other quantitative methods to control subprocesses. That is, collected process and product measurements must be used in process management.

5. Optimizing: At this highest level, the organization must use the process and product measurements to drive process improvement. Trends must be analyzed and the processes adapted to changing business needs.

Key Points…

- Software processes are the 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.
- The Rational Unified Process is a modern generic process model that is organized into phases (inception, elaboration, construction and transition) but separates activities (requirements, analysis and design, etc.) from these phases.

Key Points…

- Requirements engineering is the process of developing a software specification.
- Design and implementation processes are concerned with transforming a requirements specification into an executable software system.
- Software 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.

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 development and delivery so that changes may be made without disrupting the system as a whole.
- Process improvement to enhance the quality of the software, reducing costs, or accelerating the development processes