Introduction To Software Engineering

Presented by:

Basma Hussien Mohamed

- Software process models
- Coping with change
- Rapid software development
- Reusability – Big Bang Model

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.

- Plan Driven Development : Water Fall Model
- Incremental Developement or RAD:
  - Incremental Model
  - ProtoType Model

  - Reusabilty Development: Reuse oriented Model
  - Agile Development : Extreme programming model

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 RAD and 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

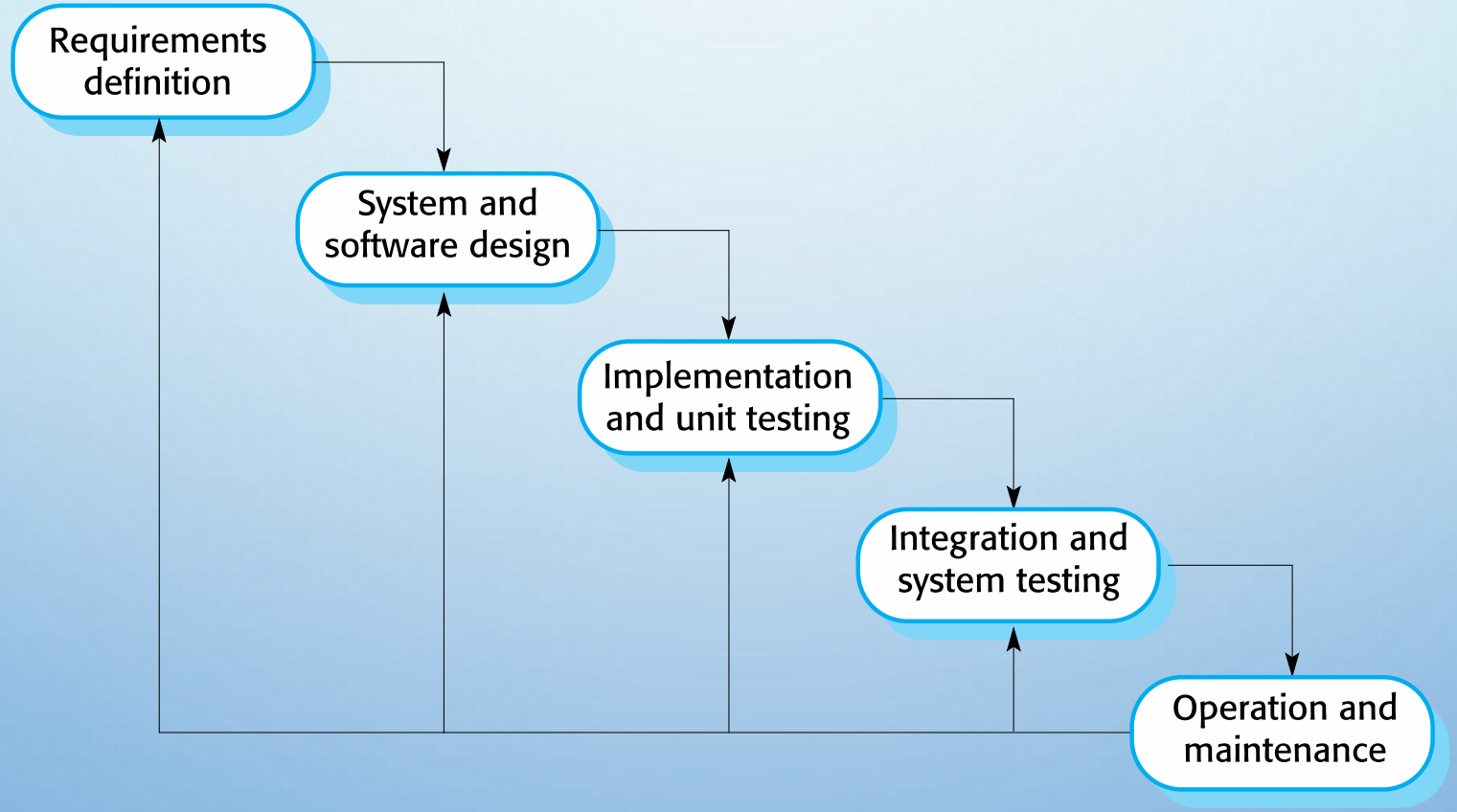
Plan-Driven Development

The Waterfall Model

technology terms

WATERFALL METHODOLOGY

The waterfall model



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, that needs well planning .
  - In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.

The big problem:
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

Reducing the costs of rework

- Change anticipation (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.

Rapid Application Development
(**RAD as answer to cope with change**)

Rapid software development

- Rapid development and delivery is now often the most important requirement for software systems
  - Businesses operate in a fast –changing requirement and it is practically impossible to produce a set of stable software requirements
  - Software has to evolve quickly to reflect changing business needs.
- Rapid software development
  - Specification, design and implementation are inter-leaved
  - System is developed as a series of versions with stakeholders involved in version evaluation
  - User interfaces are often developed using an IDE and graphical toolset

Coping with changing requirements

A. System prototyping, where a version of the system is developed quickly to check the customer's requirements and the feasibility of design decisions. This approach supports change anticipation.

B. Incremental delivery, where system increments are delivered to the customer for comment and experimentation. This supports both change avoidance and change tolerance.

A. Evolutionary Prototyping Model

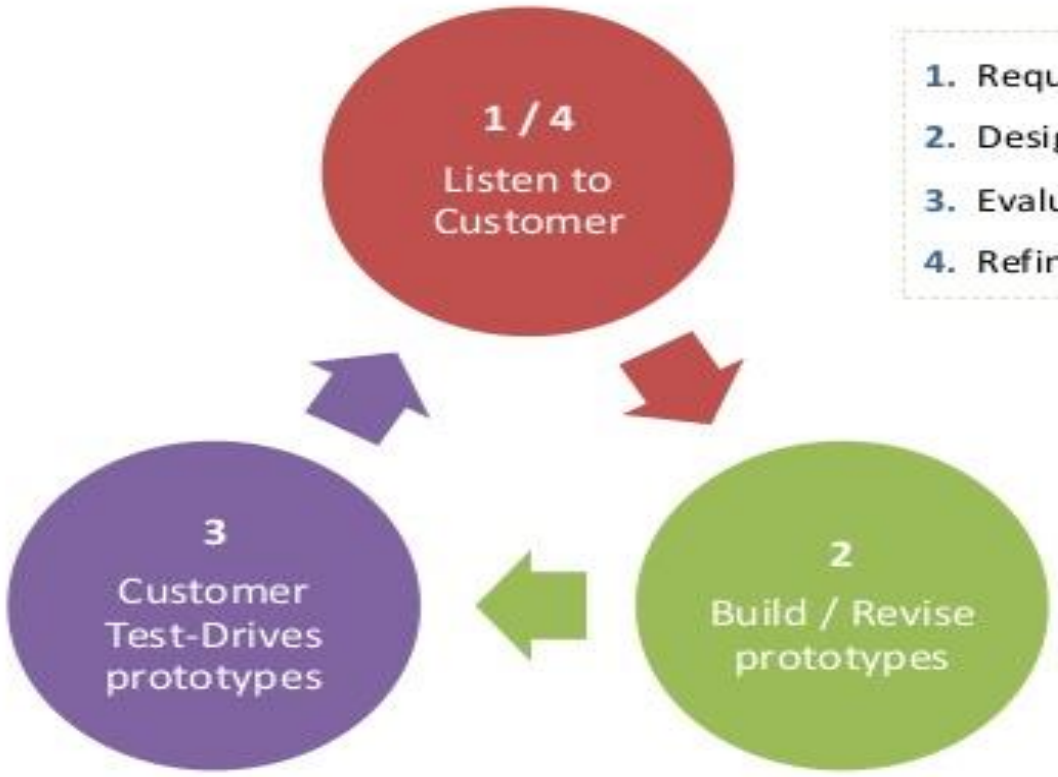
Software prototyping

- 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;

The process of prototype development

Prototyping Model



1. Requirements gathering.
2. Design and build SW prototype.
3. Evaluate prototype with customer.
4. Refine requirements.

Benefits of prototyping

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

Prototype development

- May be based on rapid prototyping 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.

B. Incremental delivery

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.

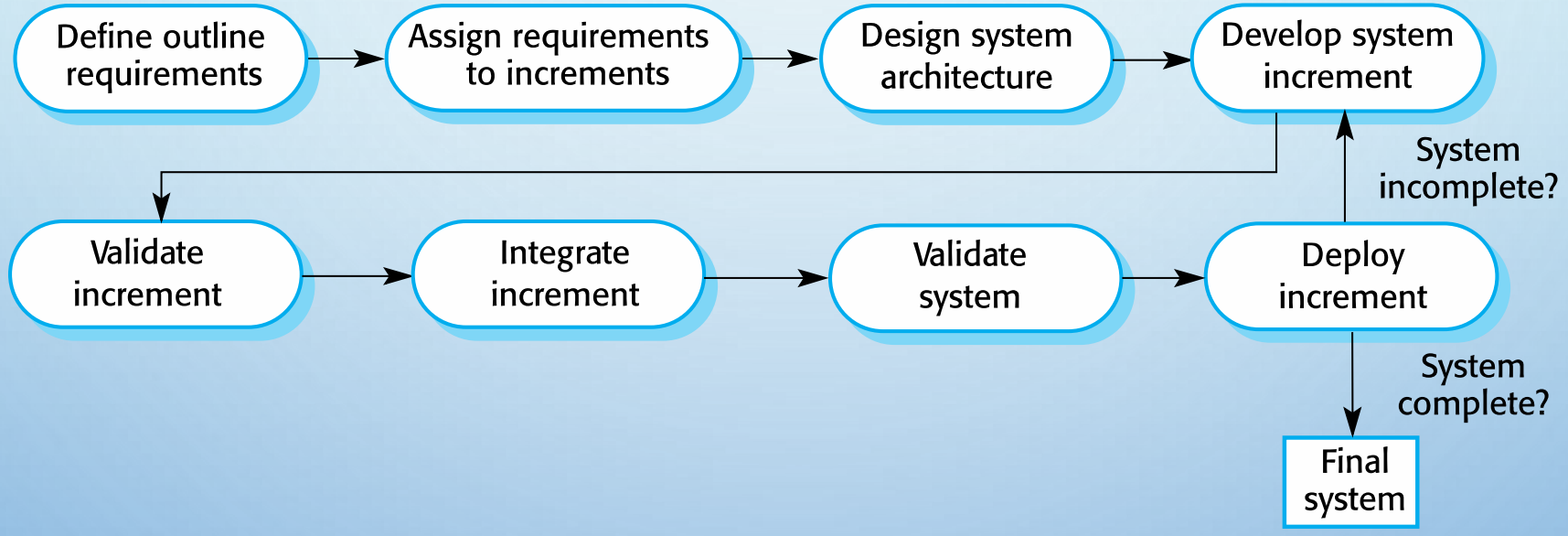
- User requirements are prioritised and the highest priority requirements are included in early increments.

- 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.
- 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.

Iteration vs Increment

Iteration vs Increment

- When discussing iterative and incremental development, the terms *iteration* and *increment* are often used freely and interchangeably. However, **They are NOT synonyms.**

- ***Iteration*** refers to the cyclic nature of a process in which activities are repeated in a structured manner.

- ***Increment*** refers to the quantifiable outcome of each iteration.

Iteration vs Increment

- Iterations can offer a development process two key things:
  - iterative refinement, where the process improves what already exists and is being done.
  - and incremental development, where the process results in progress against project objectives.

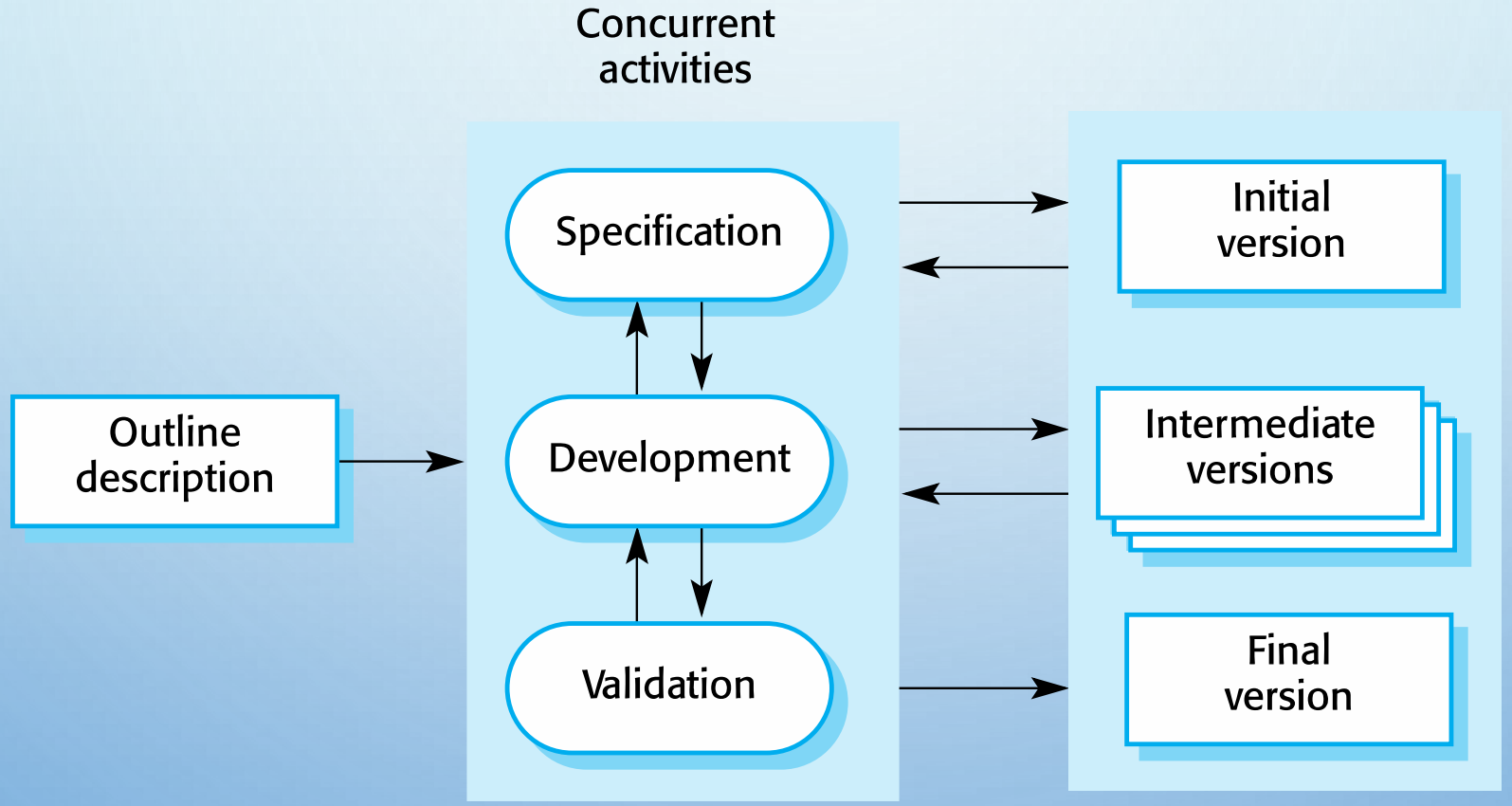
- Additionally, the term *increment* has the obvious implication that there should be more of something at the end of an iteration than there was at the start “new release”.

- Without a clear notion of an increment, iterations are likely to just go in circles!!

- So, we **develop software *iteratively*** and **release *incrementally*** in various sizes over time.

Incremental Development Summary

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.

Reuse-Oriented Software Engineering

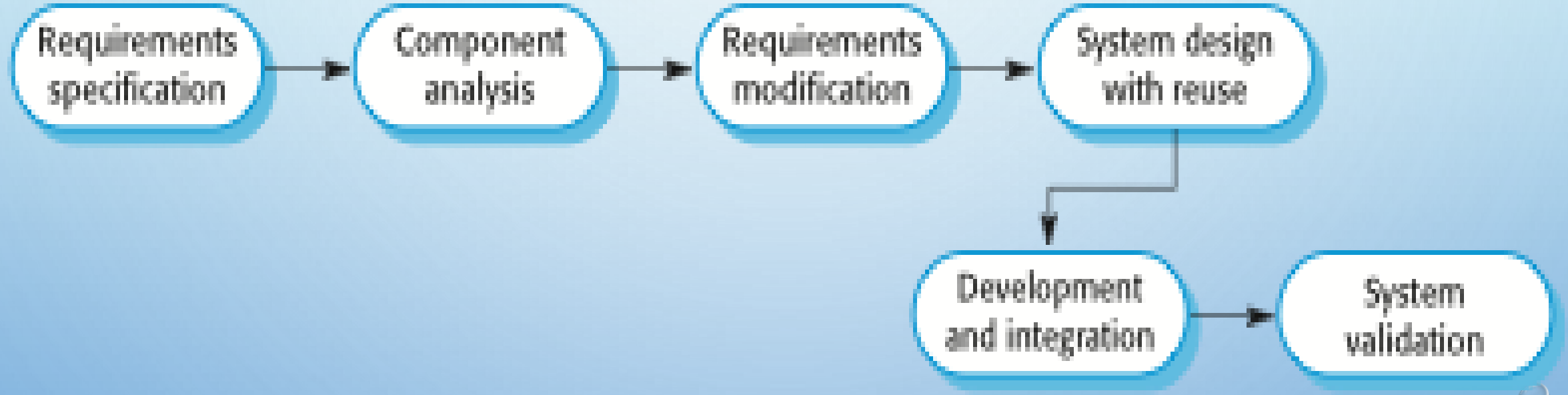
Reuse-Oriented Software Engineering (Integration and configuration)

- Based on software reuse where systems are integrated from existing components or application systems (sometimes called 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

- Web services that are developed according to service standards and which are available for remote invocation.
- 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.

Reuse-oriented software engineering



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

Thank you ☺