



## Chapter 2 – Software Processes

# Topics covered

---



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

# The software process

amp



- ✧ 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

amp



- ✧ 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

*agile*  
*agility*  
*amp*



- ✧ 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



## ✧ 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.

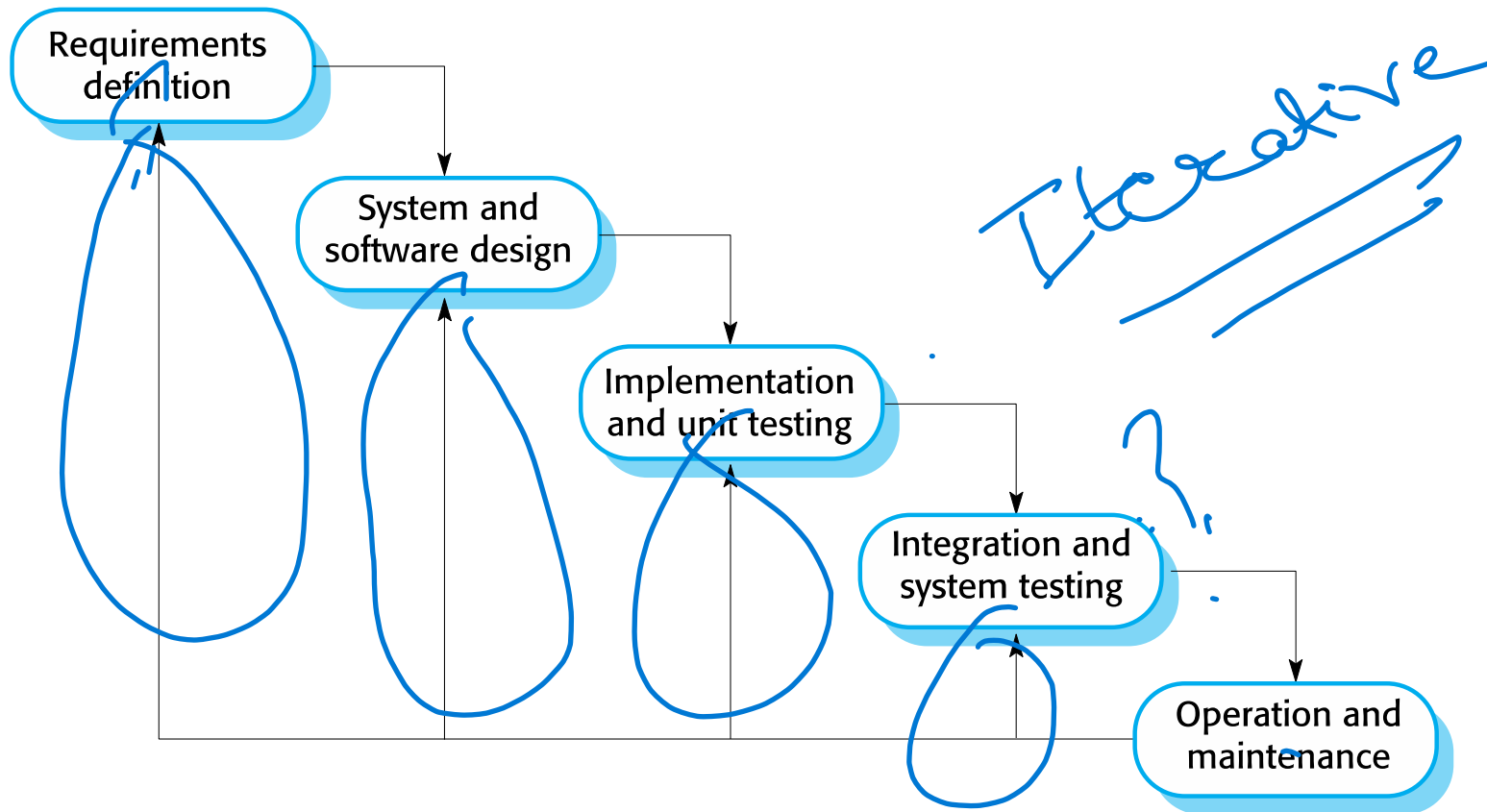
## ✧ Integration and configuration

- The system is assembled from existing configurable 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

amp





# 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.

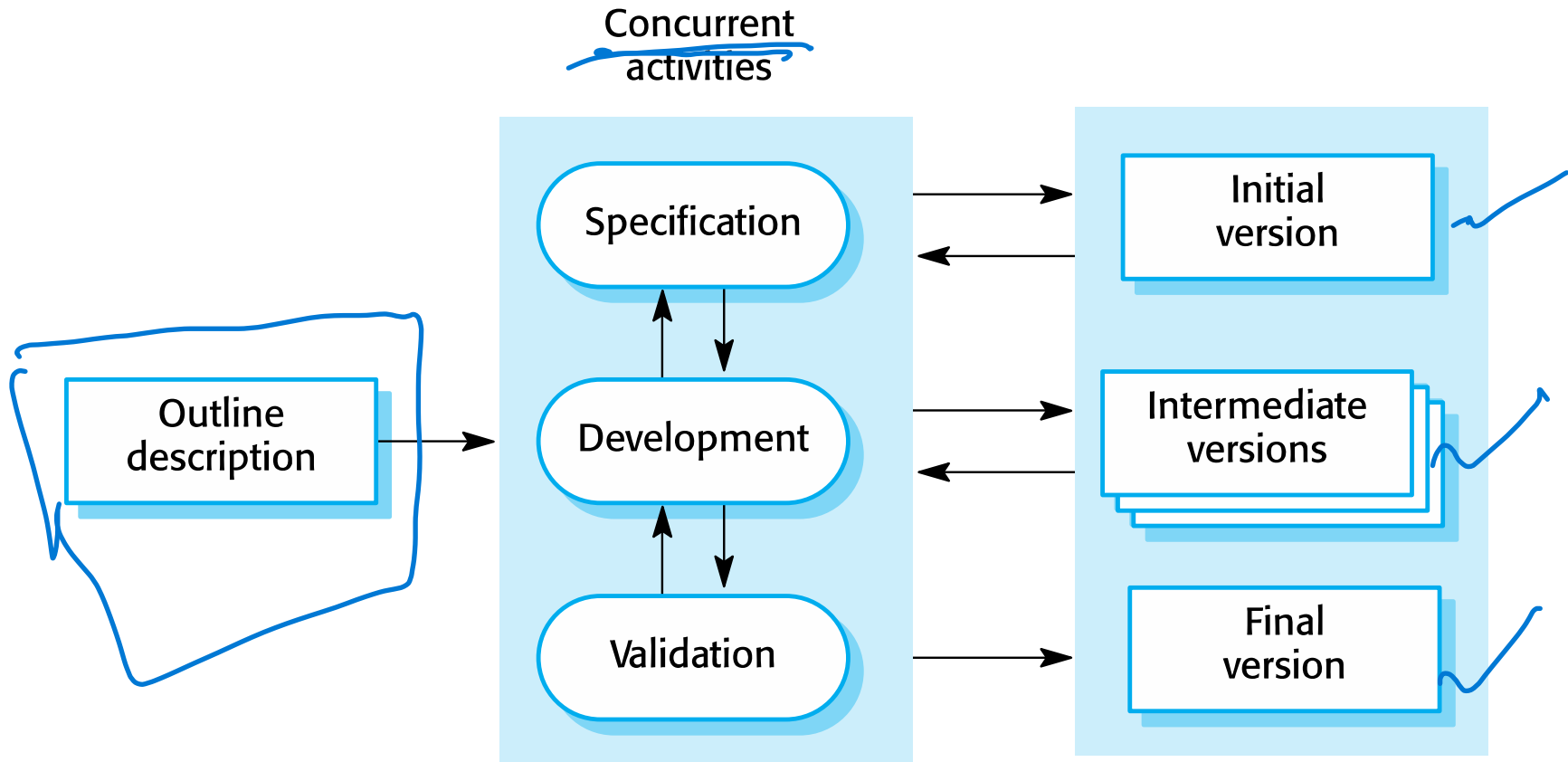
# Waterfall model problems



- ✧ 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.

# Incremental development

*amp*



# 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

amp



## ✧ 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 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
  - Reuse covered in more depth in Chapter 15.

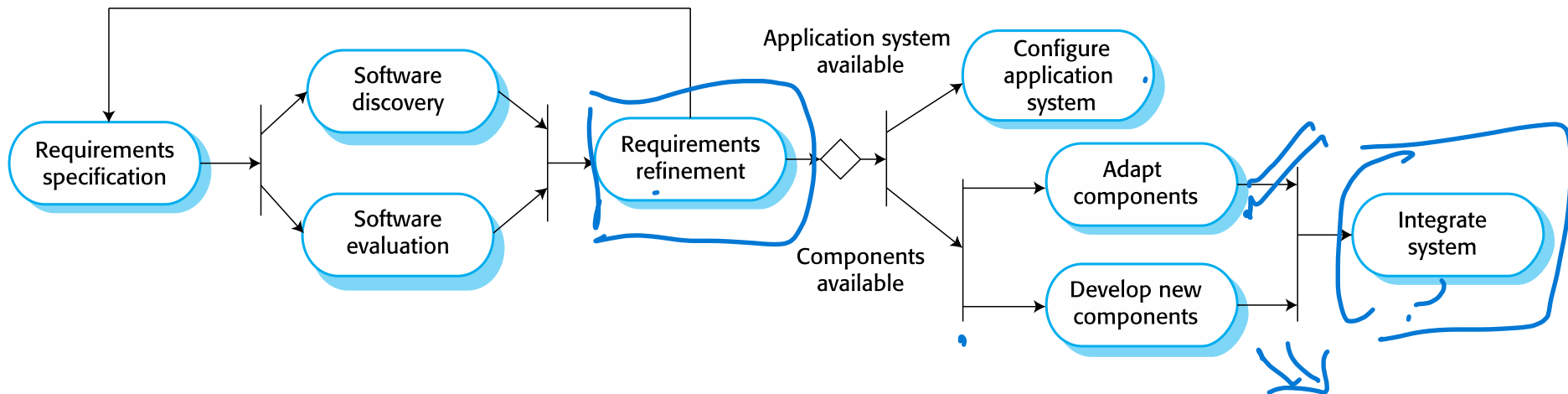
# Types of reusable software

---



- ✧ 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





# Key process stages

---



- ✧ Requirements specification
- ✧ Software discovery and evaluation
- ✧ Requirements refinement
- ✧ Application system configuration
- ✧ Component adaptation and integration

# 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

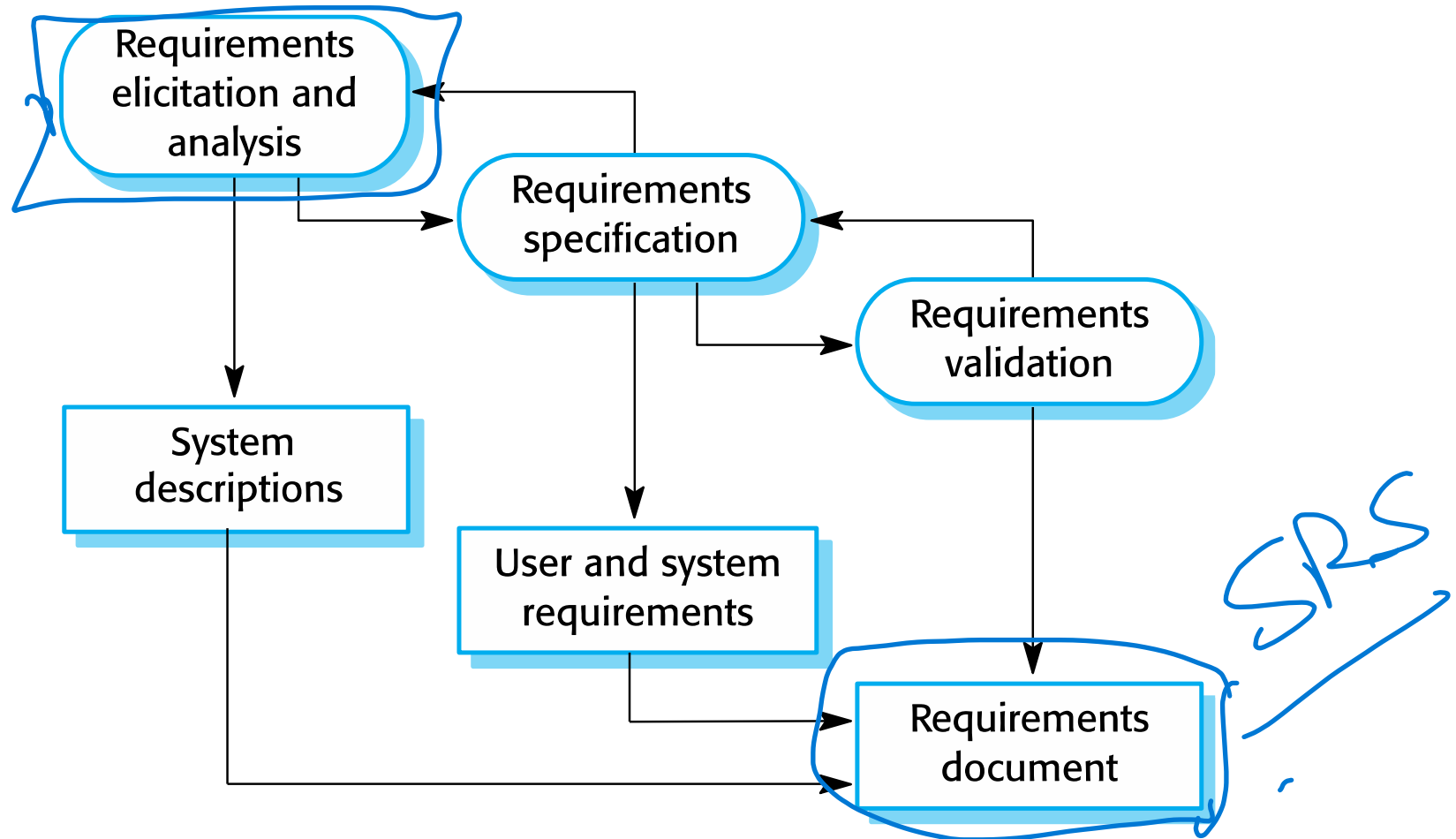
# Process activities



- ✧ 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.
- ✧ For example, in the waterfall model, they are organized in sequence, whereas in incremental development they are interleaved.



# The requirements engineering process



# Software specification

---



- ✧ The process of establishing what services are required and the constraints on the system's operation and development.
- ✧ Requirements engineering process
  - 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

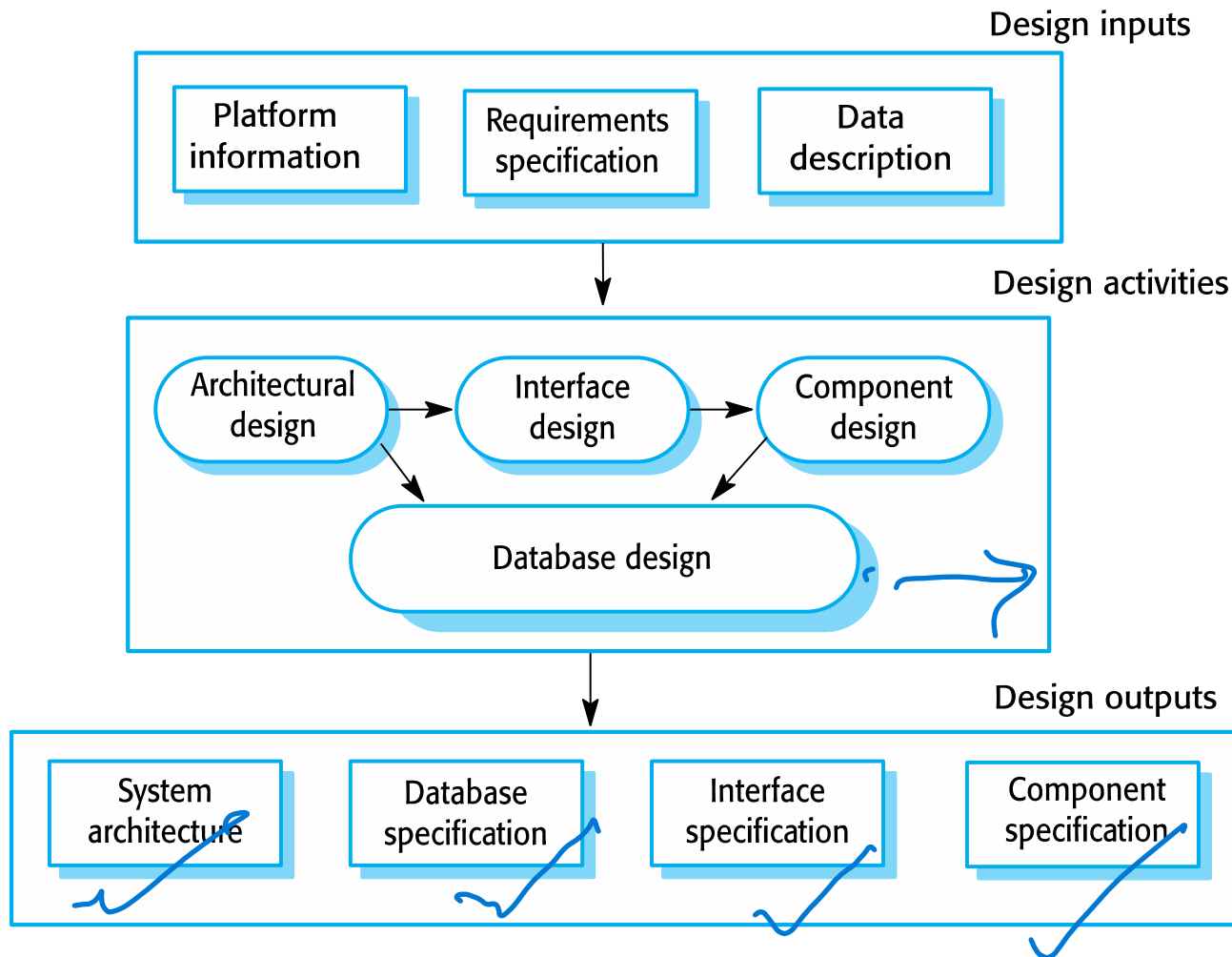
# Software design and implementation

SRS



- ✧ 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 (subsystems or modules), their relationships and how they are distributed.
- ✧ *Database design*, where you design the system data structures and how these are to be represented in a database.
- ✧ *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.

# 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.

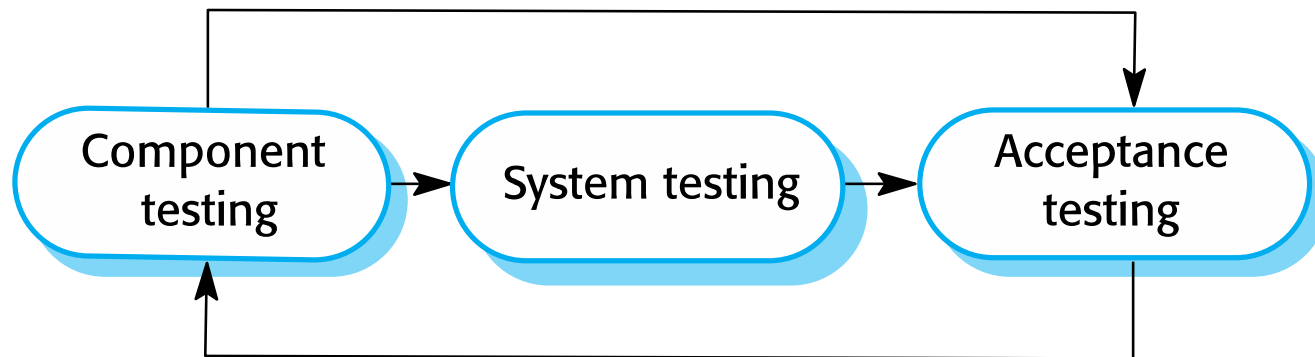
# Software validation

---



- ✧ 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.

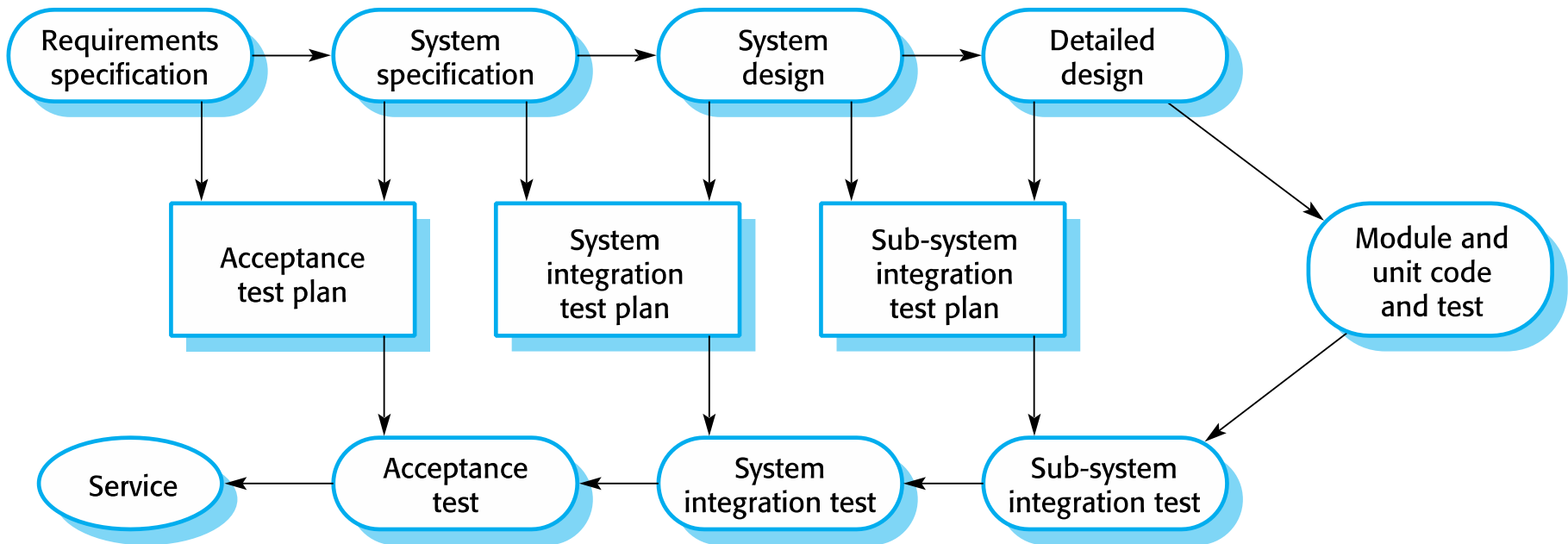
## ✧ System testing

- Testing of the system as a whole. Testing of emergent properties is particularly important.

## ✧ Customer 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)



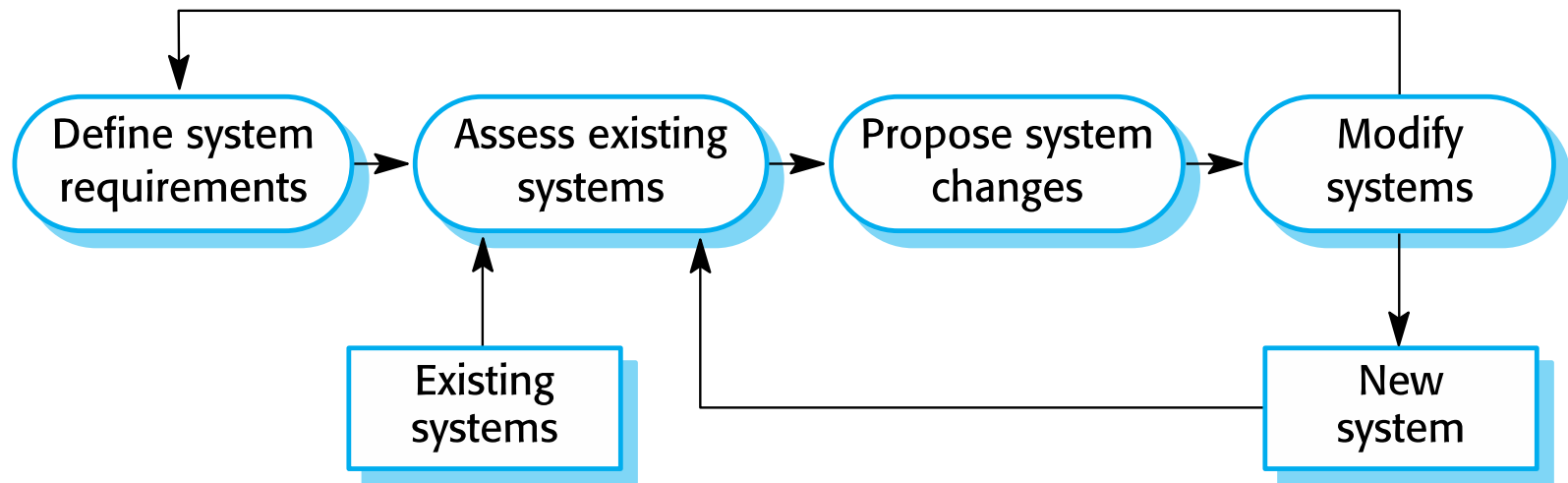
# Software evolution

---



- ✧ 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







# 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-analysing requirements) as well as the costs of implementing new functionality

# Reducing the costs of rework



- ✧ Change anticipation, 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

*amp*



- ✧ 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

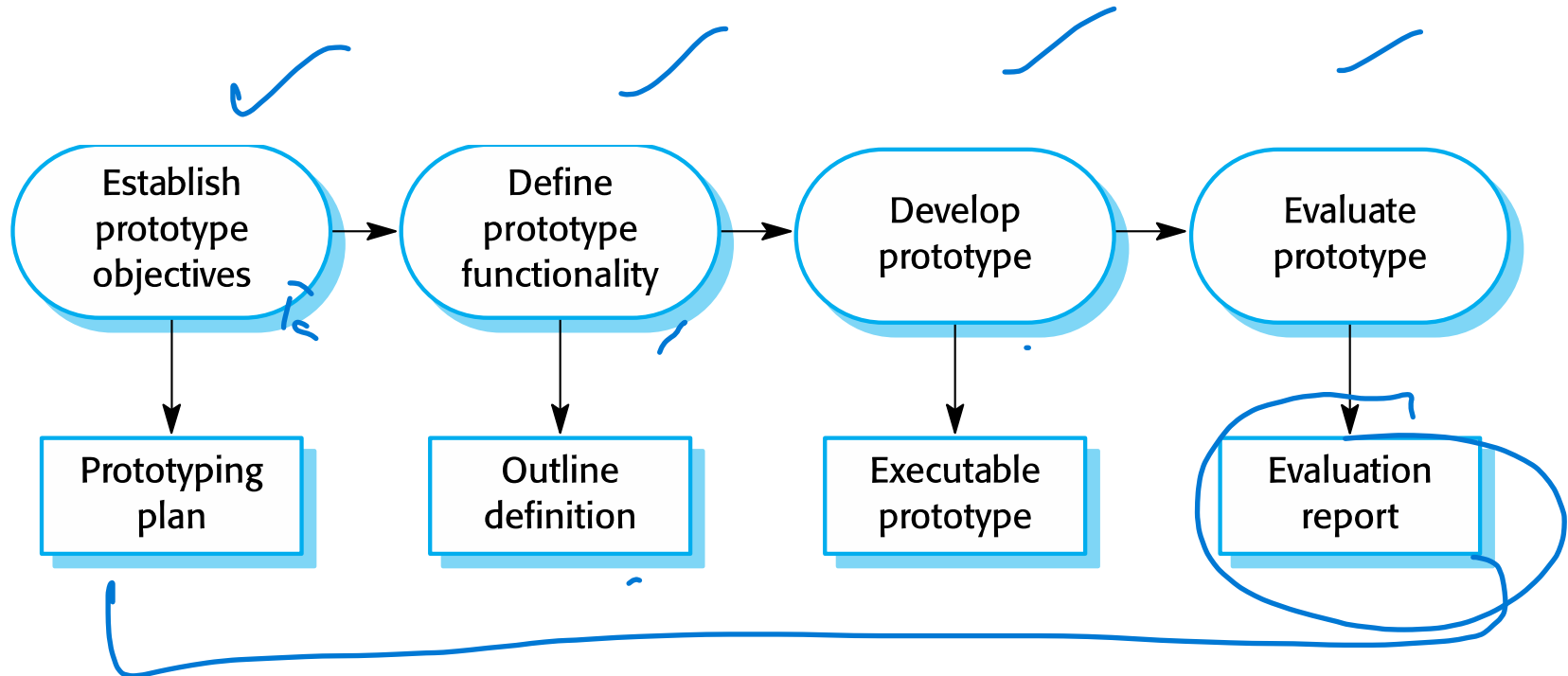
amp



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

# The process of prototype development

*amp*



# 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 organisational quality standards.

# 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

amp



## ✧ Incremental development

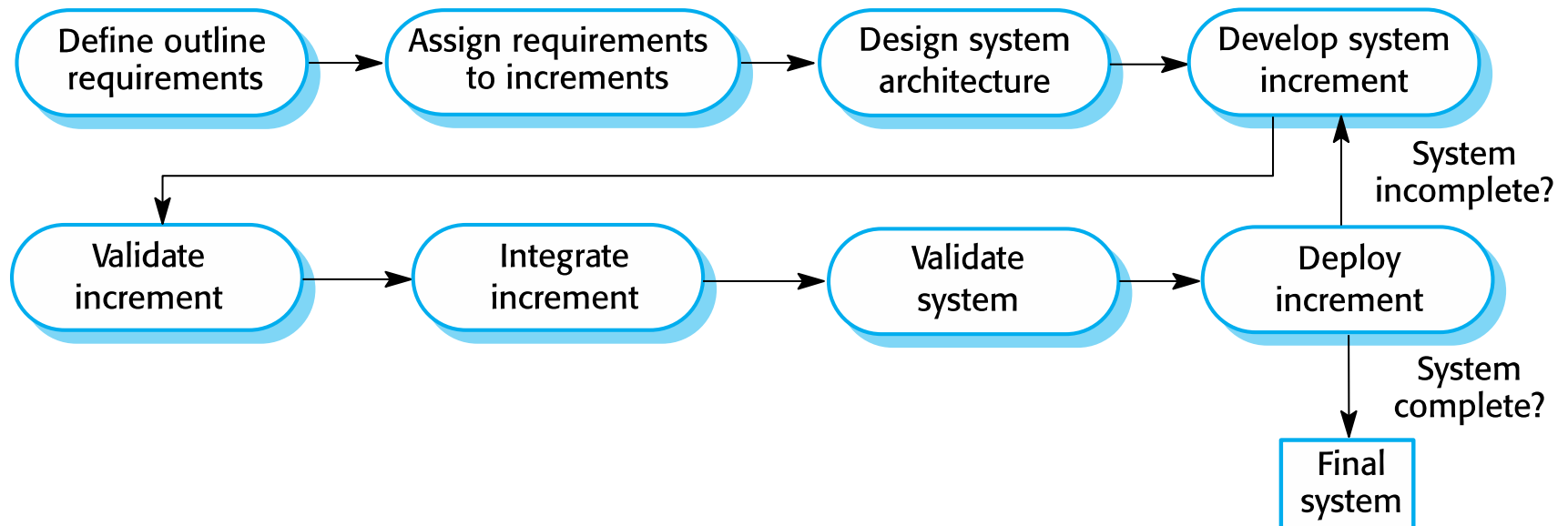
- Develop the system in increments and evaluate each increment before proceeding to the development of the next increment;
- Normal approach used in agile methods;
- Evaluation done by user/customer proxy.

## ✧ 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

amp



# 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.

# The Spiral Model



A heavy-weight, plan-driven, highly-structured approach for large projects.

Especially designed for those with higher chances of failure.

Combines iterative model, emphasizes risk assessment, customer participation, prototyping, and more

Definitely an iterative process.

**Source: After Boehm  
1988 (© 1988 IEEE)**

*amp*



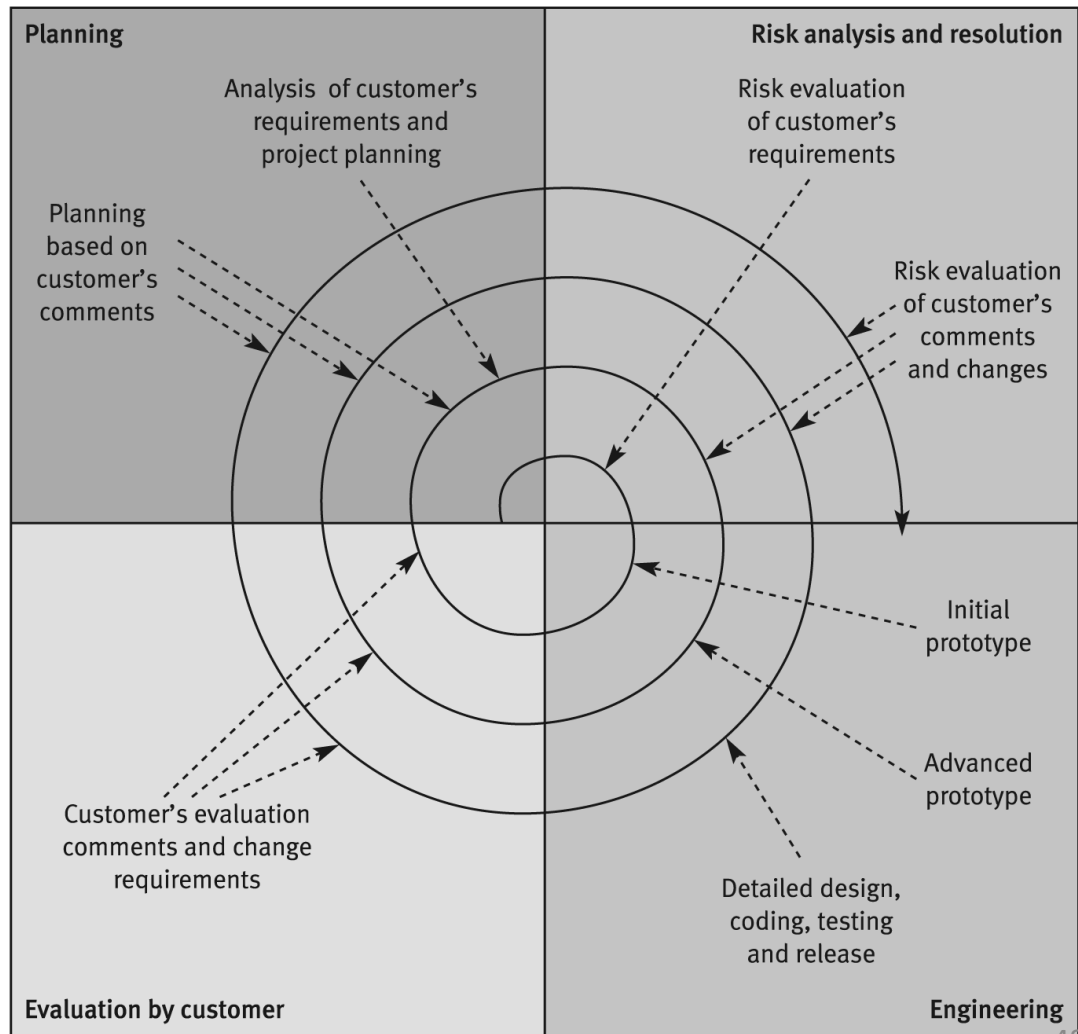
Can see each spiral  
includes:

Planning

Risk Analysis / Resolution

Engineering activities  
(design, code, test...)

Customer Evaluation  
(errors, changes, new  
requirements...)





# 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.
- ✧ Requirements engineering is the process of developing a software specification.

# Key points

---



- ✧ 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.
- ✧ Processes should include activities such as prototyping and incremental delivery to cope with change.

# Key points

---



- ✧ Processes may be structured for iterative development and delivery so that changes may be made without disrupting the system as a whole.
- ✧ The principal approaches to process improvement are agile approaches, geared to reducing process overheads, and maturity-based approaches based on better process management and the use of good software engineering practice.
- ✧ The SEI process maturity framework identifies maturity levels that essentially correspond to the use of good software engineering practice.