Chapter 2 – Software Processes

Topics covered

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

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 (or deliverables), 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 (conditions) 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.
  - The right process depends on the customer and regulatory requirements, the environment where the software will be used, and the type of software being developed
- Software processes can be improved by process standardization (helping process communication, training, automation, and to include good SE practices)

Software process models

Software process models

✧ A **software process model** (sometimes called a **Software Development Life Cycle** or **SDLC model**) is a simplified representation of a software process.

✧ Each process model represents a process from a particular perspective and thus provides partial information about that process

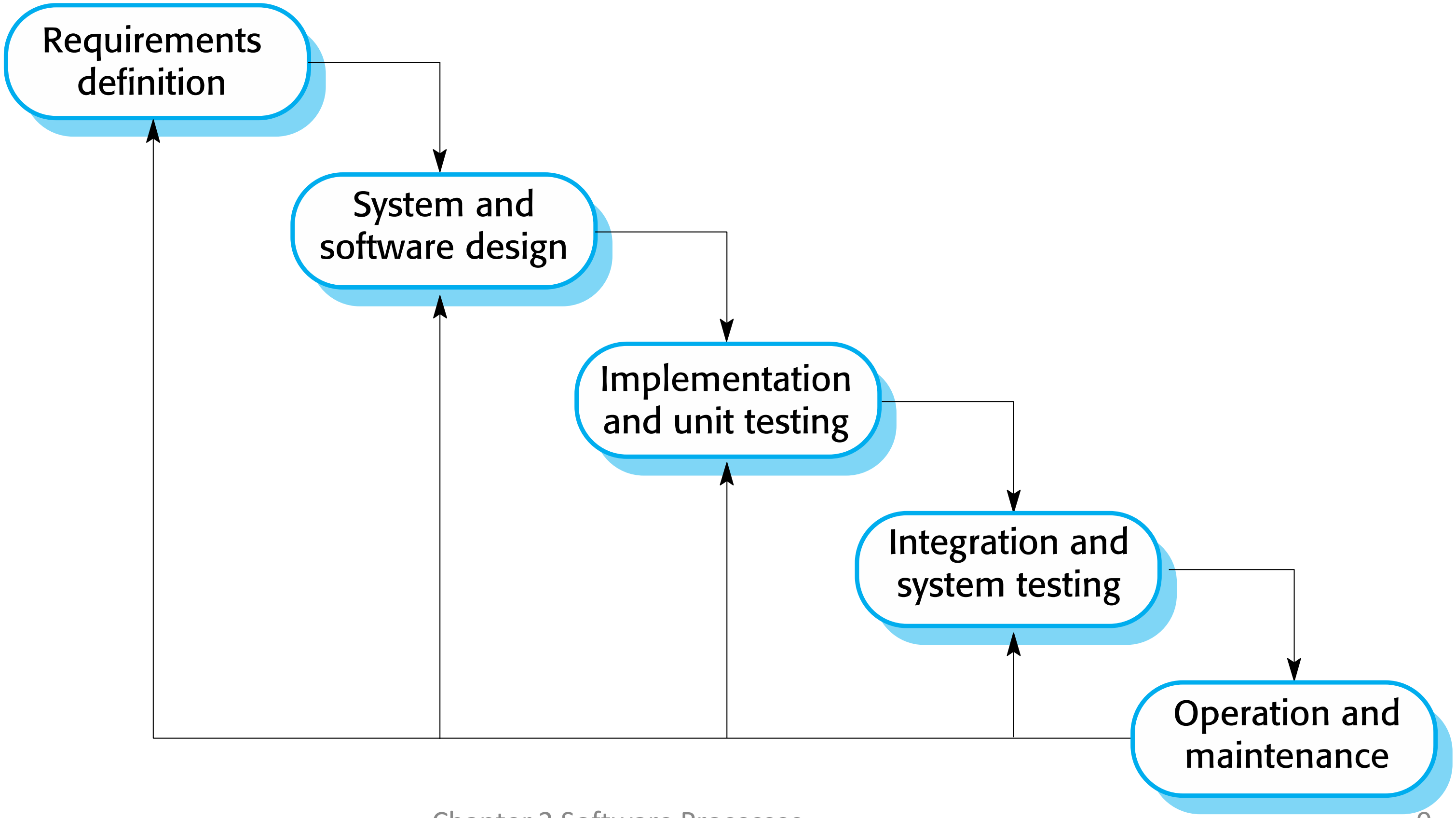
- For example, a process activity model show the activities and their sequence but may not show the roles of people involved in these activities

✧ This chapter covers a number of very general process models (sometimes called *process paradigms*)

Software process models
(from an architectural perspective - process framework)

- ✧ The waterfall model (software life cycle)
  - Plan-driven model. Separate and distinct phases of specification and development.
    - **Plan** and **schedule** all of the process activities before starting software development
- ✧ Incremental development
  - Specification, development and validation are interleaved. The system is developed as a series of versions (increments) by adding functions incrementally. May be plan-driven or agile.
- ✧ Integration and configuration (Reuse-oriented software engineering)
  - 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



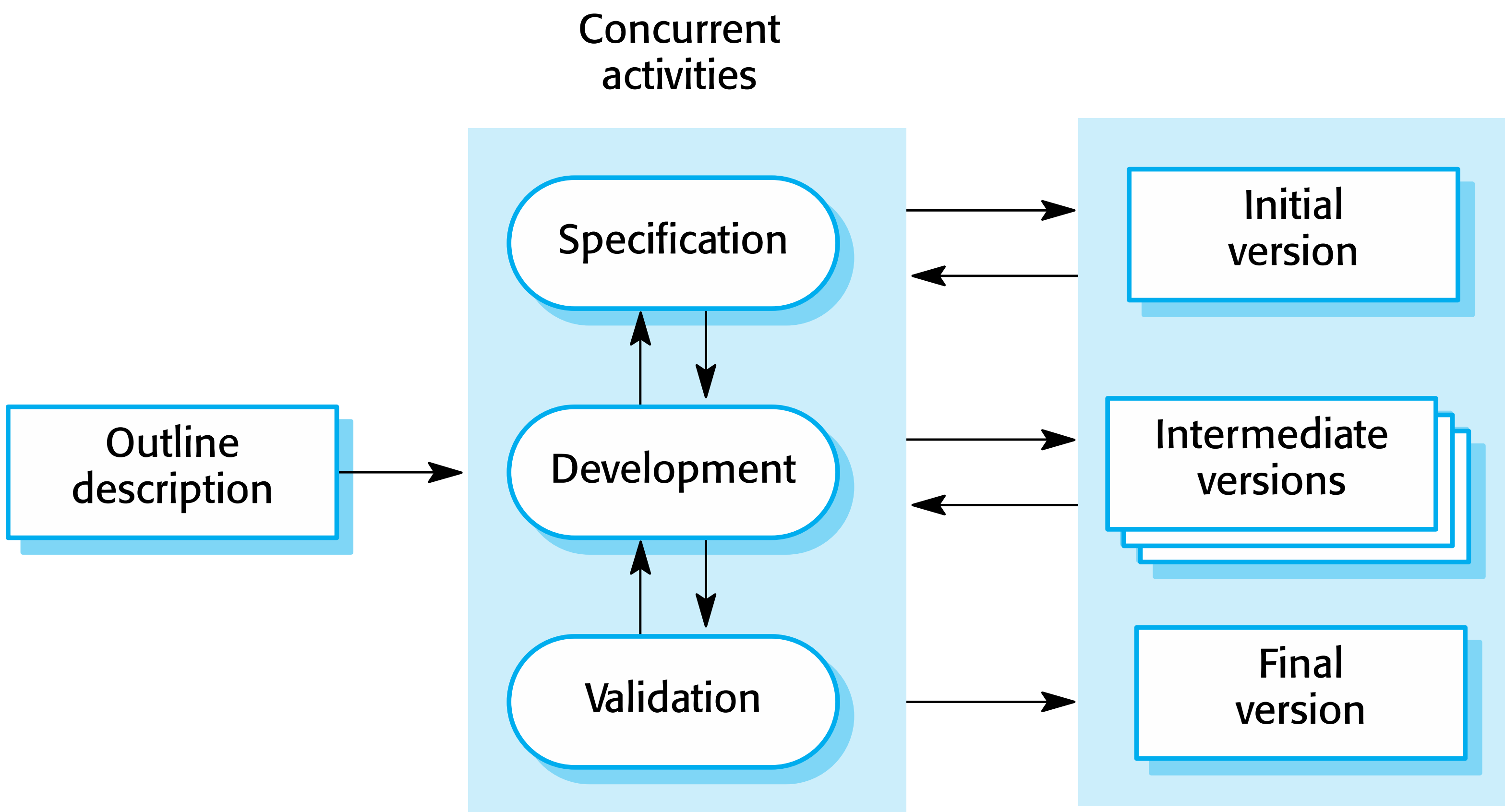
Waterfall model phases

- 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. In practice, the waterfall process is never a simple linear model but involves feedback from one phase to another

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.
  - It is appropriate for embedded systems (the requirements are inflexible to change), critical systems (the need for extensive requirement analysis and design), and large software system (the need for independent development of different subsystems)
  - 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



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. (code & fix)

- 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.

✧ The problems become particularly acute for large, complex, long-lifetime systems, where different teams develop different parts of the system

Incremental development

- ✧ Incremental development in some form is now the most common approach for the development of application systems
- ✧ Incremental development can be either plan-driven, agile or, more usually, a mixture of both
  - Plan-driven: the system increments are identified in advance
  - Agile: the early increments are identified but the development of later increments depends on progress and customer priorities

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.

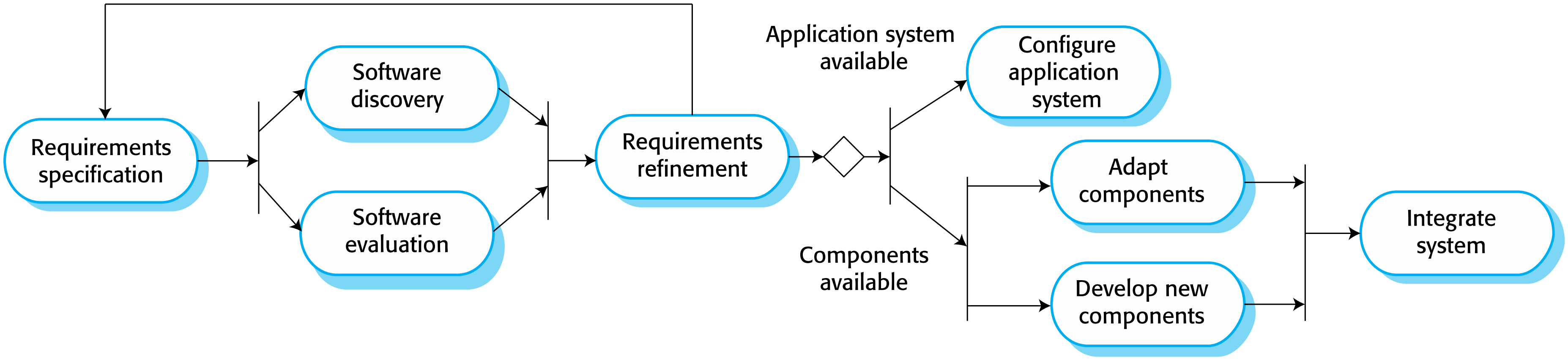
Types of reusable software

- ✧ 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.

Key process stages

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

Reuse-oriented software engineering



Advantages and disadvantages

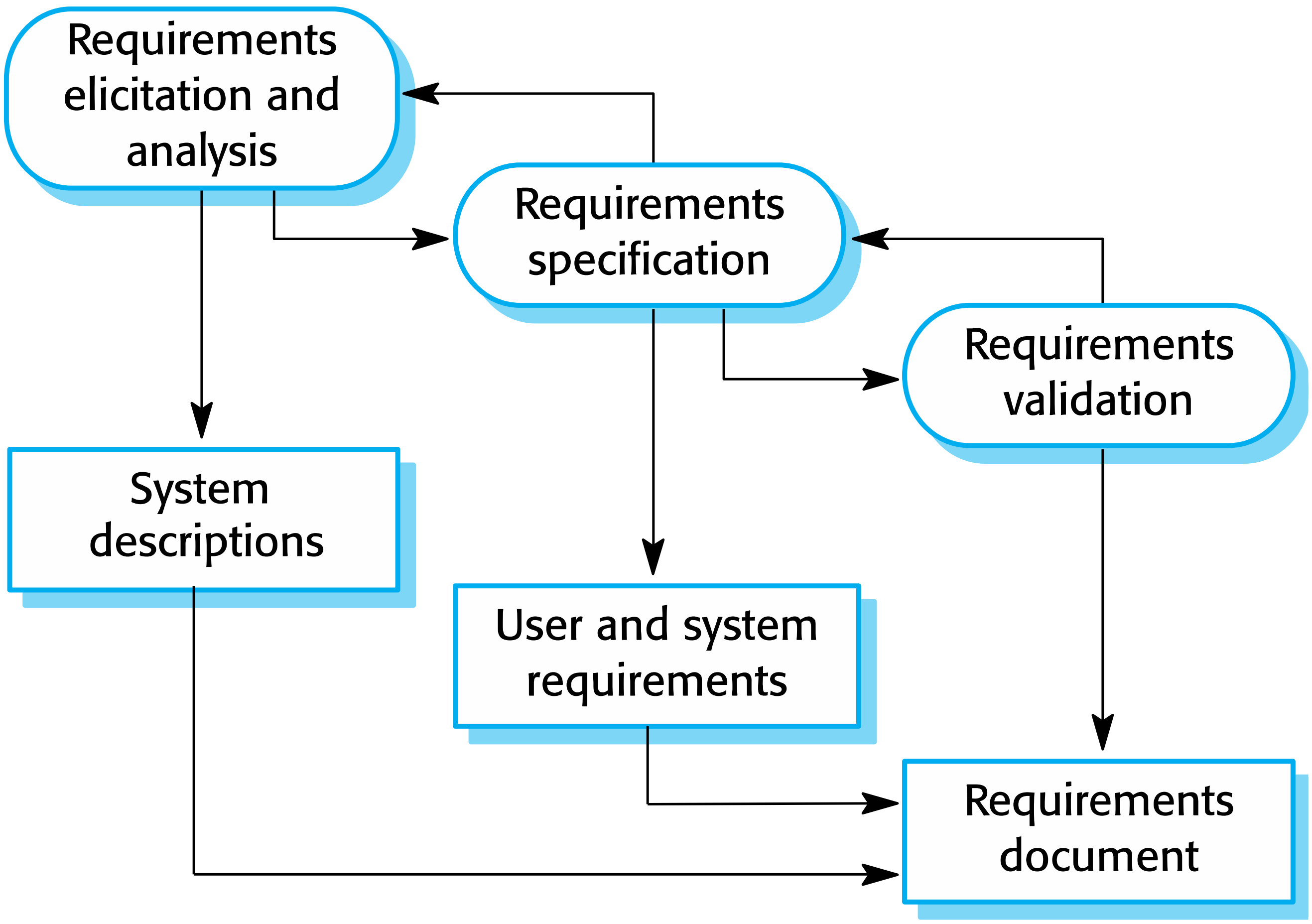
- Advantages
  - Reduced costs and risks as less software is developed from scratch
  - Faster delivery and deployment of system
- Disadvantages
  - But requirements compromises are inevitable so system may not meet real needs of users
  - Loss of control over evolution of reused system elements

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.
- How these activities are carried out depends on the type of software being developed, the experienced and competence of the developers, and the type of organization developing the software

The requirements engineering process



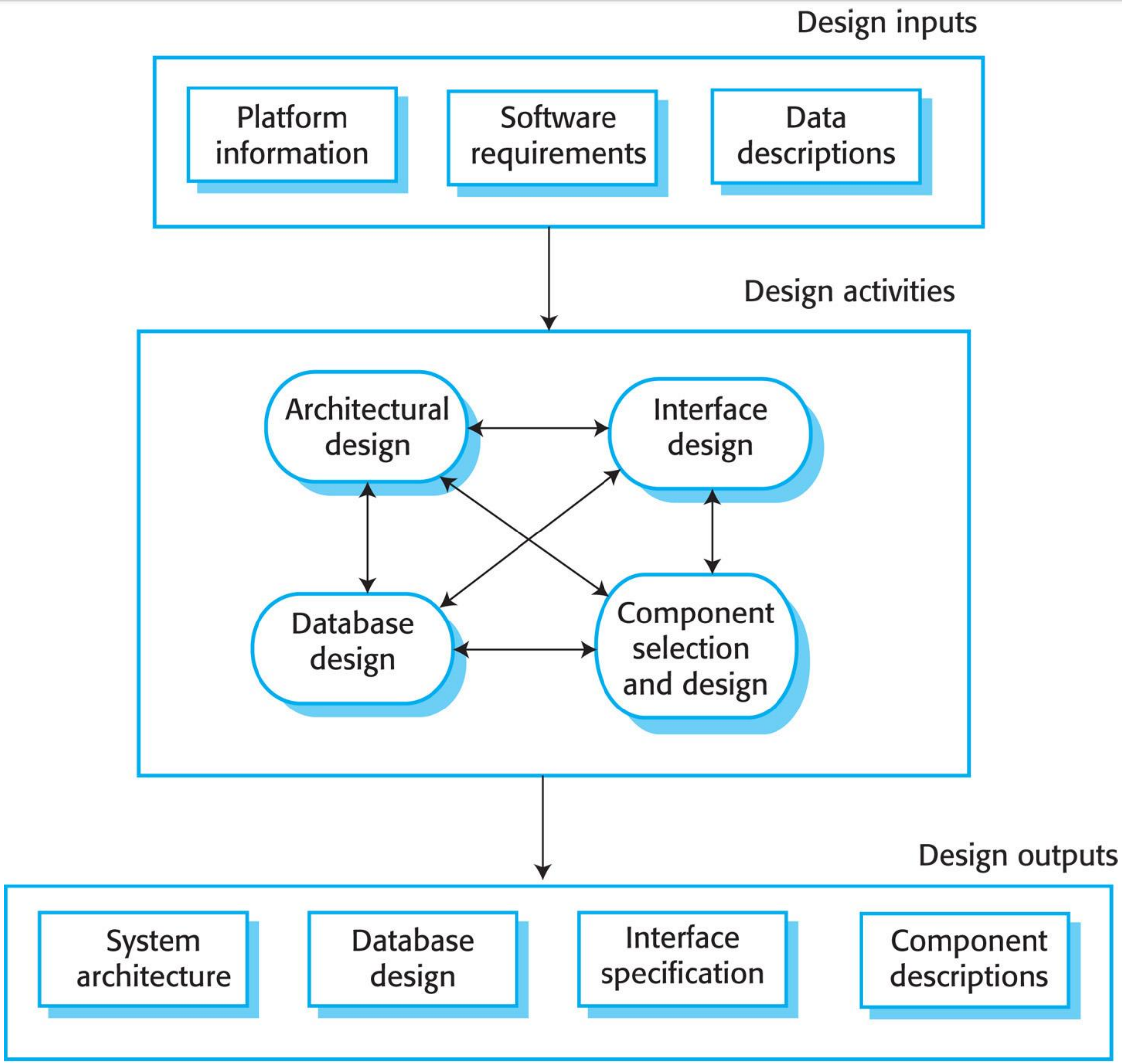
Software specification

- 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

- ✧ 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 <u>design</u> and <u>implementation</u> are closely related and <u>may be inter-leaved</u>.

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.

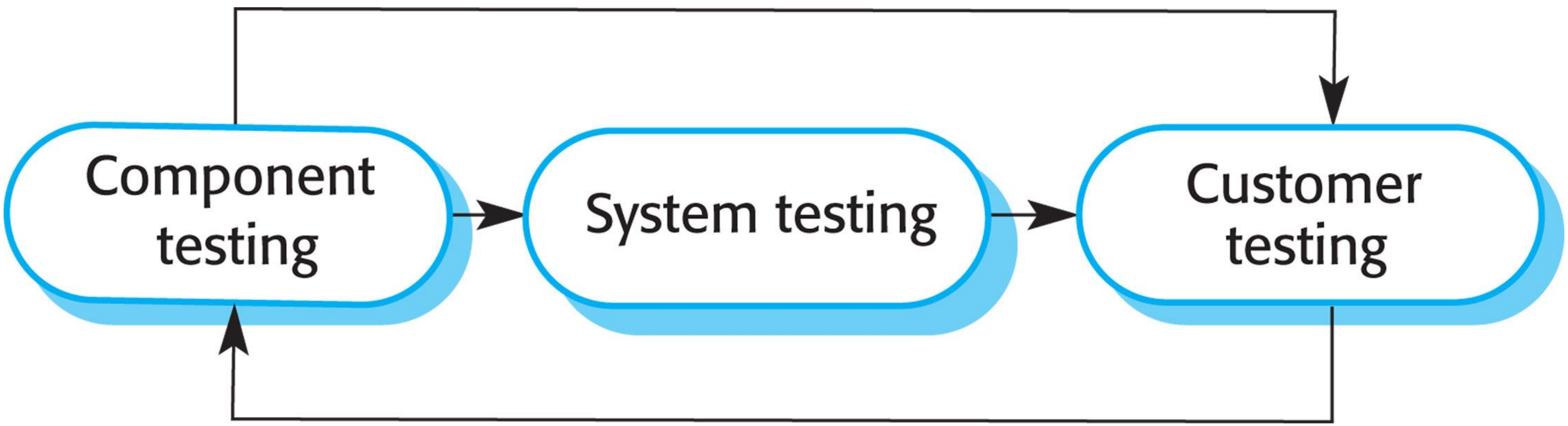
System implementation

- 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



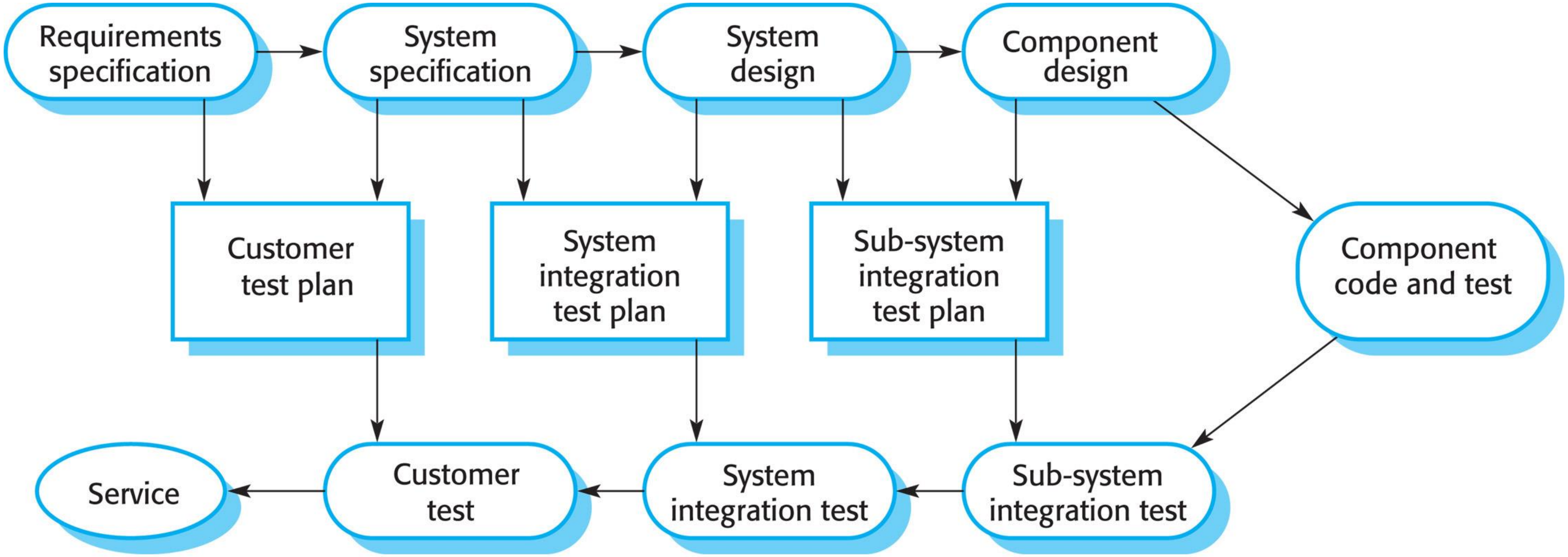
Testing stages

✧ 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.
- For products that are sold as applications, customer testing is sometimes called beta testing

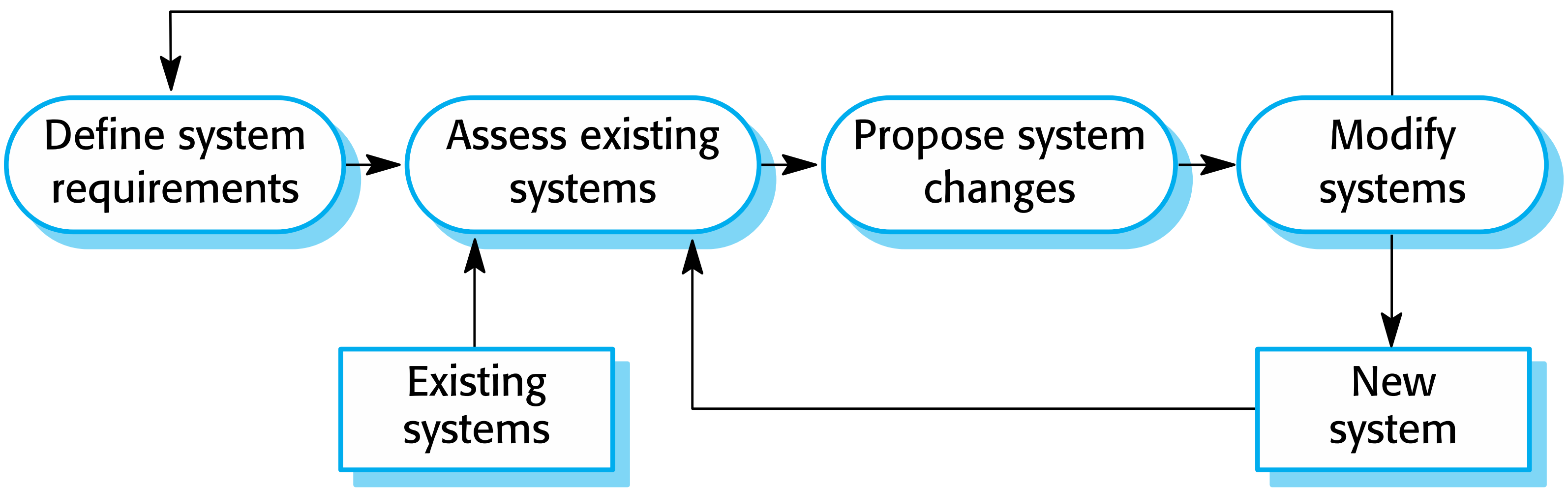
Testing phases in a plan-driven software process (V-model)



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.
  - Refactoring – improving the structure and organization of a program (see Chapter 3)

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 as it allows users to experiment with the system before delivery and so refine their requirements. The number of requirements change proposals made after delivery is therefore likely to be reduced
- Incremental delivery, where system increments are delivered to the customer for comment and experimentation. This supports both change avoidance and change tolerance. It avoids the premature commitment to requirements for the whole system and allows changes to be incorporated into later increments at relatively low cost

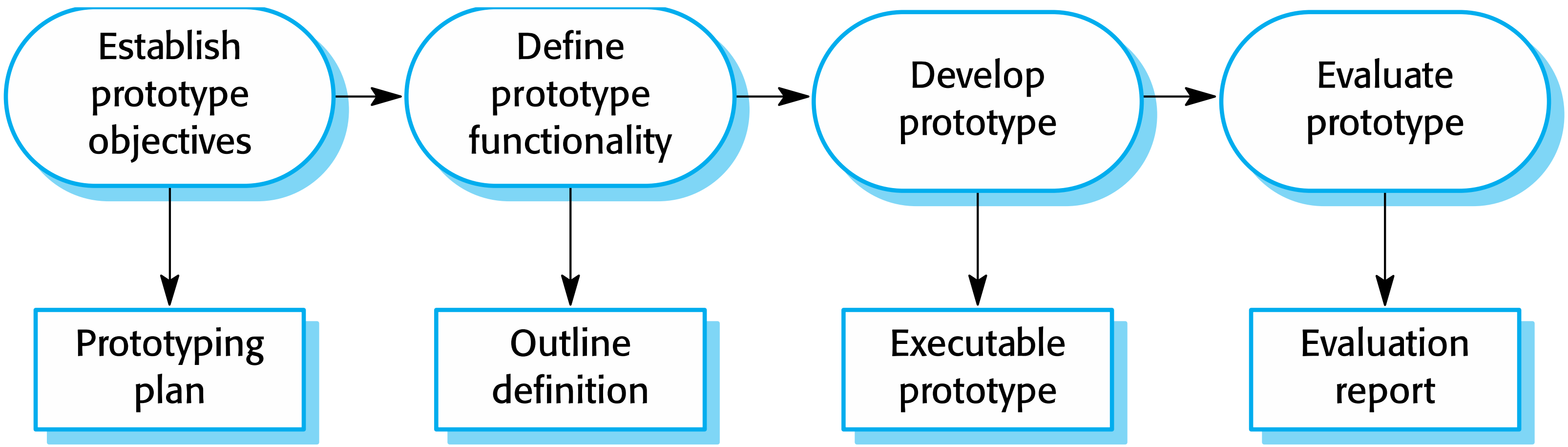
Software prototyping

- A prototype is an initial version of a system used to demonstrate concepts, try out design options, and find out more about the problem and its possible solutions
- 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.
    - For software subject to parallel implementation, back-to-back testing is the execution of a test on the similar implementations and comparing the results.

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 (to reduce prototyping costs and accelerate the delivery schedule)
  - Prototype should focus on areas of the product that are not well-understood; (leave some functionality out)
  - Error checking and recovery may not be included in the prototype; (ignore error handling and reduce reliability and quality standards)
  - Focus on functional rather than non-functional requirements such as reliability and security (relax non-functional requirements)

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; (…has quality problems)
- Prototypes are normally undocumented;
- The prototype structure is usually degraded through rapid change;
- The prototype probably will not meet normal organisational quality standards.

✧ Throw-away prototyping vs. Evolutionary prototyping

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

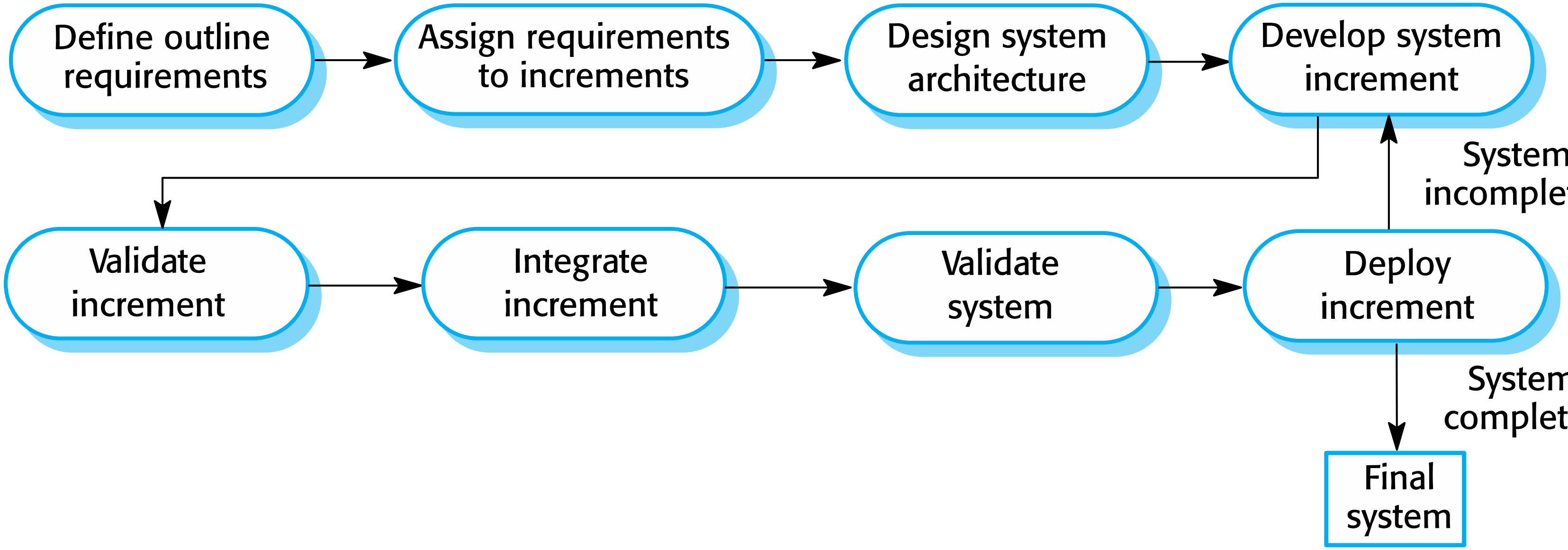
✧ Incremental development

- 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.
- ✧ It should be relatively easy to incorporate changes into the system

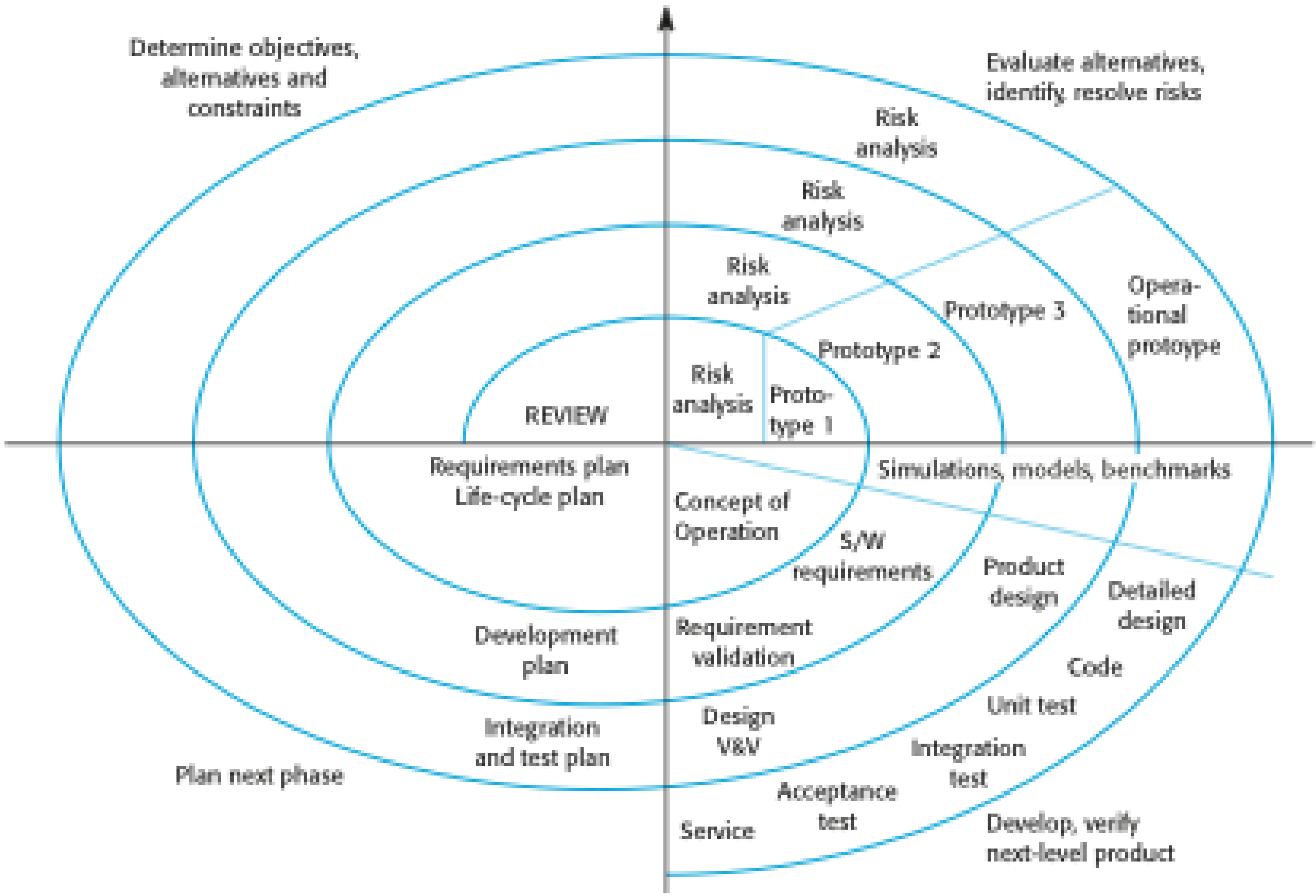
Incremental delivery problems

- 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.
- Can be difficult for developing a replacement system
- 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**.

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. (also include constraints, risks, and alternative strategies)

✧ 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. (..waterfall, prototype, formal method)

✧ Planning

- The project is reviewed and the next phase of the spiral is planned.

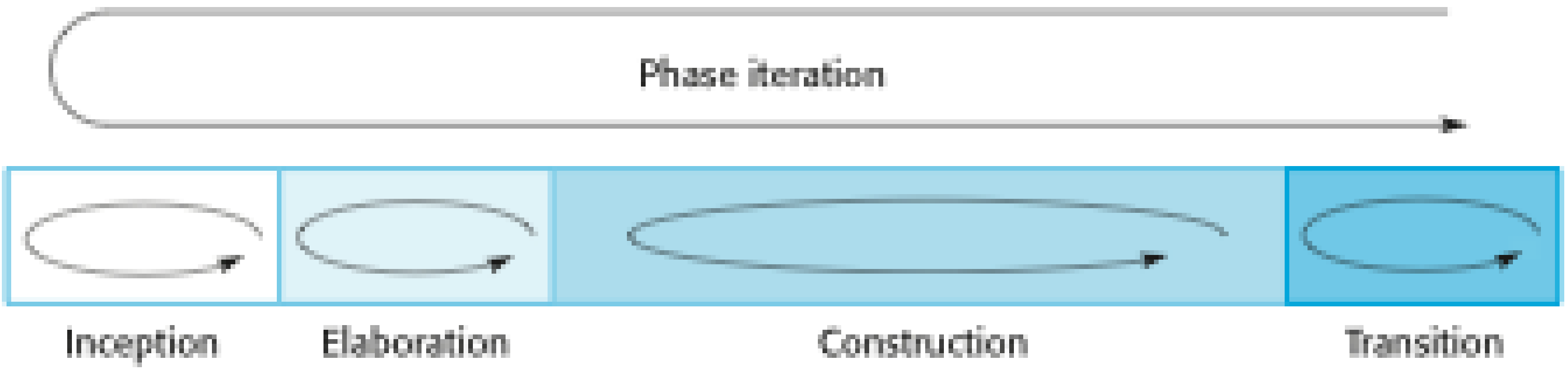
Spiral model usage

- ✧ 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

- ✧ 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 Rational Unified Process



The phases in the RUP are more closely related to business rather than technical concerns

RUP phases

✧ Inception

- Establish the business case for the system.

✧ Elaboration

- Develop an understanding of the problem domain, establish the system architecture, develop project plan, and identify risks

✧ Construction

- System design, programming and testing.

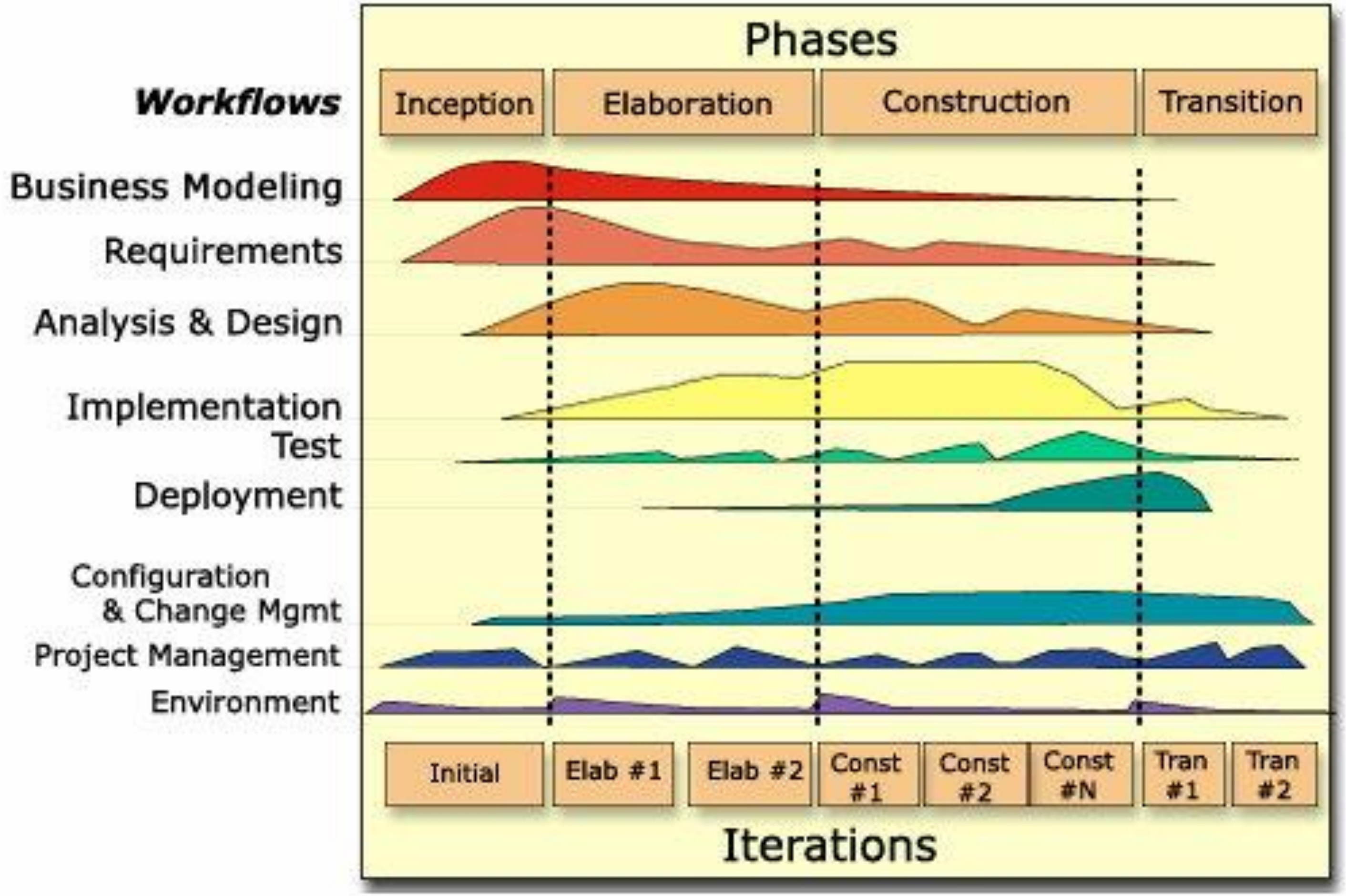
✧ Transition

- Deploy the system in its operating environment. (something ignored in most software process but is, in fact, an expensive and sometimes problematic activity)

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.

RUP Phases and Disciplines



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.

6 core engineering workflow and 3 core supporting workflow

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 (six fundamental practices)

- ✧ 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 (six fundamental practices)

✧ 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 improvement

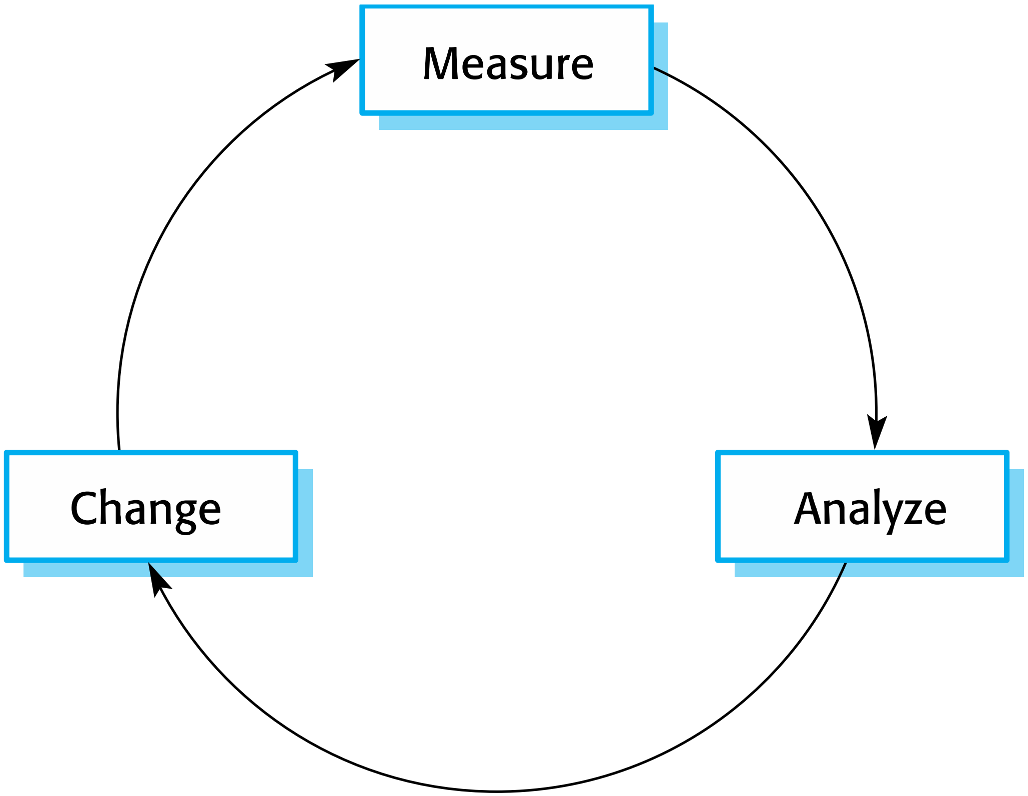
Process improvement

- How to deliver a cheaper and better software quickly?
  - Improve software process is a way to enhance quality, reduce cost, or accelerate development of software
- Many software companies have turned to **software process improvement** (SPI) as a way of enhancing the quality of their software, reducing costs or accelerating their development processes.
  - Software Engineering Institute's **CMMI** (Capability Maturity Model Integration)
- Process improvement means understanding existing processes and changing these processes to increase product quality and/or reduce costs and development time.
- Process improvement is a long-term and continuous activity

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 primary goals of the approach are improved product quality and process predictability
- 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.
  - The improvement philosophy is that the best processes are those with lowest overheads and agile approaches can achieve this

The process improvement cycle



Process improvement activities

✧ *Process measurement*

- You measure one or more attributes of the software process or product. These measurements forms a baseline that helps you decide if process improvements have been effective.

✧ *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.

✧ *Process change*

- Process changes 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.

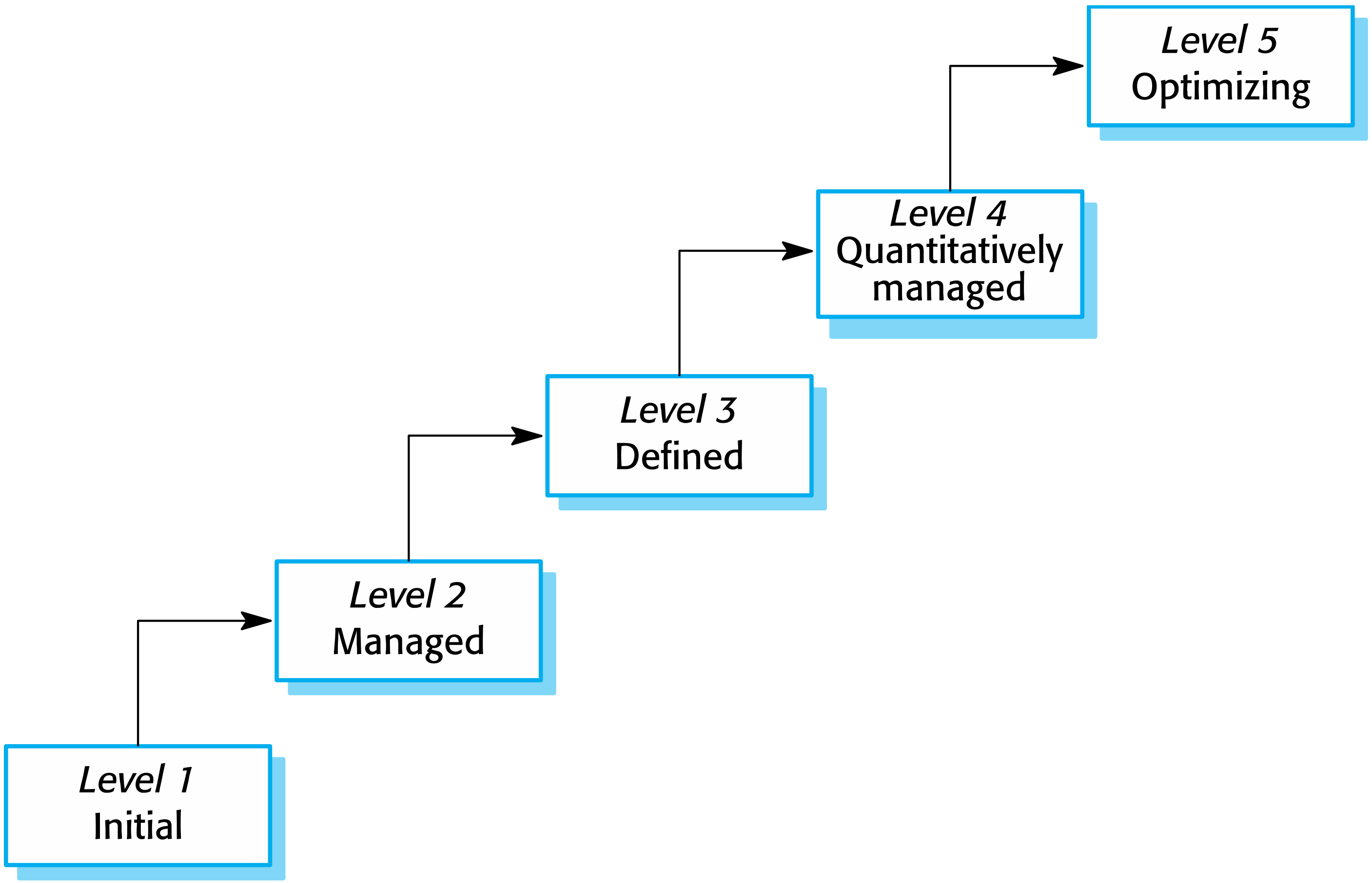
Process measurement

- Wherever possible, quantitative process data should be collected
  - However, where organisations do not have clearly defined process standards this is very difficult as you don't know what to measure. A process may have to be defined before any measurement is possible.
- Process measurements should be used to assess process improvements
  - But this does not mean that measurements should drive the improvements. The improvement driver should be the **organizational objectives**.

Process metrics

- ✧ Time taken for process activities to be completed
  - E.g. Calendar time or effort to complete an activity or process.
- ✧ Resources required for processes or activities
  - E.g. Total effort in person-days.
- ✧ Number of occurrences of a particular event
  - E.g. Number of defects discovered.

Capability maturity levels



The SEI capability maturity model

✧ Initial

- Essentially uncontrolled

✧ Managed (Repeatable)

- Product management procedures defined and used

✧ Defined

- Process management procedures and strategies defined and used

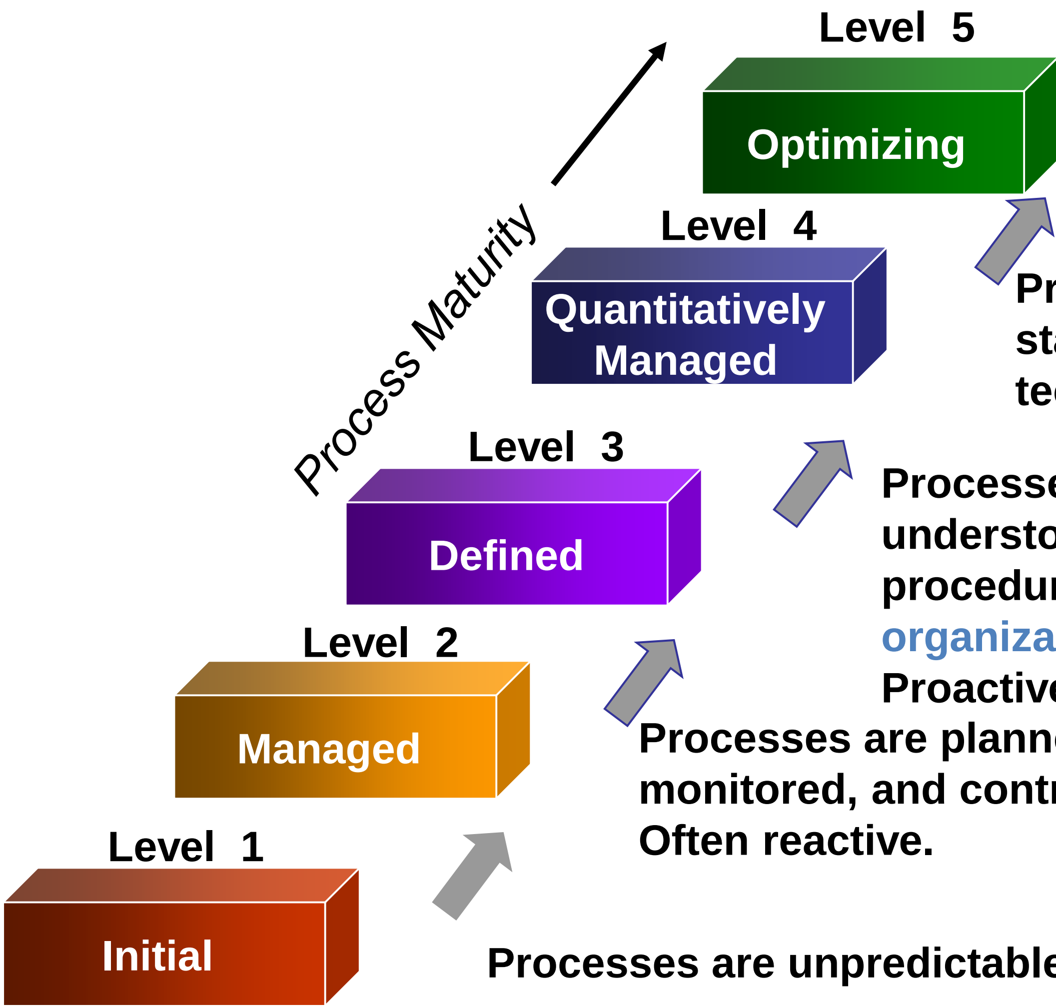
✧ Quantitatively Managed

- Quality management strategies defined and used

✧ Optimising

- Process improvement strategies defined and used

CMMI Staged Representation - 5 Maturity Levels



Process performance continually improved through incremental and innovative technological improvements.

Processes are controlled using statistical and other quantitative techniques.

Processes are well characterized and understood. Processes, standards, procedures, tools, etc. are defined at the organizational (Organization X) level. Proactive.

Processes are planned, documented, performed, monitored, and controlled at the project level. Often reactive.

Processes are unpredictable, poorly controlled, reactive.

Behaviors at the Five Levels

Maturity Level	Process Characteristics	Behaviors
5 Optimizing	Focus is on continuous quantitative improvement	Focus on "fire prevention"; improvement anticipated and desired, and impacts assessed.
4 Quantitatively Managed	Process is measured and controlled	Greater sense of teamwork and inter-dependencies
3 Defined	Process is characterized for the organization and is proactive	Reliance on defined process. People understand, support and follow the process.
2 Managed	Process is characterized for projects and is often reactive	Over reliance on experience of good people – when they go, the process goes. “Heroics.”
1 Initial	Process is unpredictable, poorly controlled, and reactive	Focus on "fire fighting"; effectiveness low – frustration high.

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.