

Chapter 2 – Software Processes

Topics covered



✧ Software process models

✧ Process activities

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.

Example of a process



✧ Preparing for a birthday party

✧ Let's practice!

Software process descriptions



- ✧ When we describe and discuss processes, we usually **talk about the activities** in these processes such as specifying a data model, designing a user interface, etc. **and the ordering of these activities.**
- ✧ Process descriptions may also include:
 - Products, which are the outcomes of a process activity;
 - Roles, which reflect the responsibilities of the people involved in the process;
 - Pre- and post-conditions, which are statements that are true before and after a process activity has been enacted or a product produced.

Plan-driven and agile processes

✧ Plan-driven processes

- All activities are **planned in advance** and **progress is measured against this plan**.

✧ Agile processes

- **planning is incremental**
 - **Easier to change** the process to reflect changing customer requirements.

✧ In practice, most practical processes include elements of both plan-driven and agile approaches.

✧ There are no right or wrong software processes.

Software process models



✧ The waterfall model

- Plan-driven model. Separate and distinct phases of specification and development.

✧ Incremental development

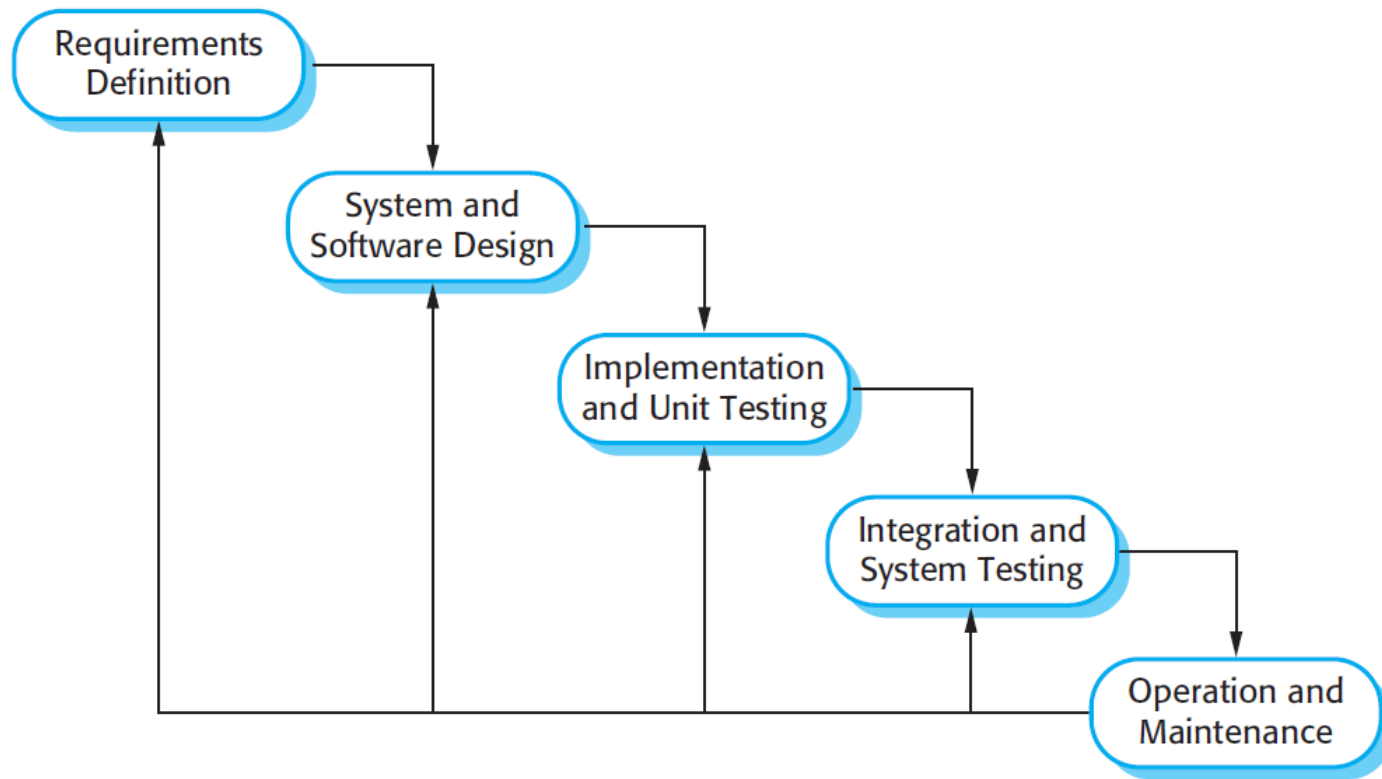
- Specification, development and validation are interleaved. May be **plan-driven or agile**.

✧ Reuse-oriented software engineering

- The system is assembled from existing components. May be **plan-driven or agile**.

✧ In practice, most large systems are developed using a process that incorporates elements from all of these models.

The waterfall model



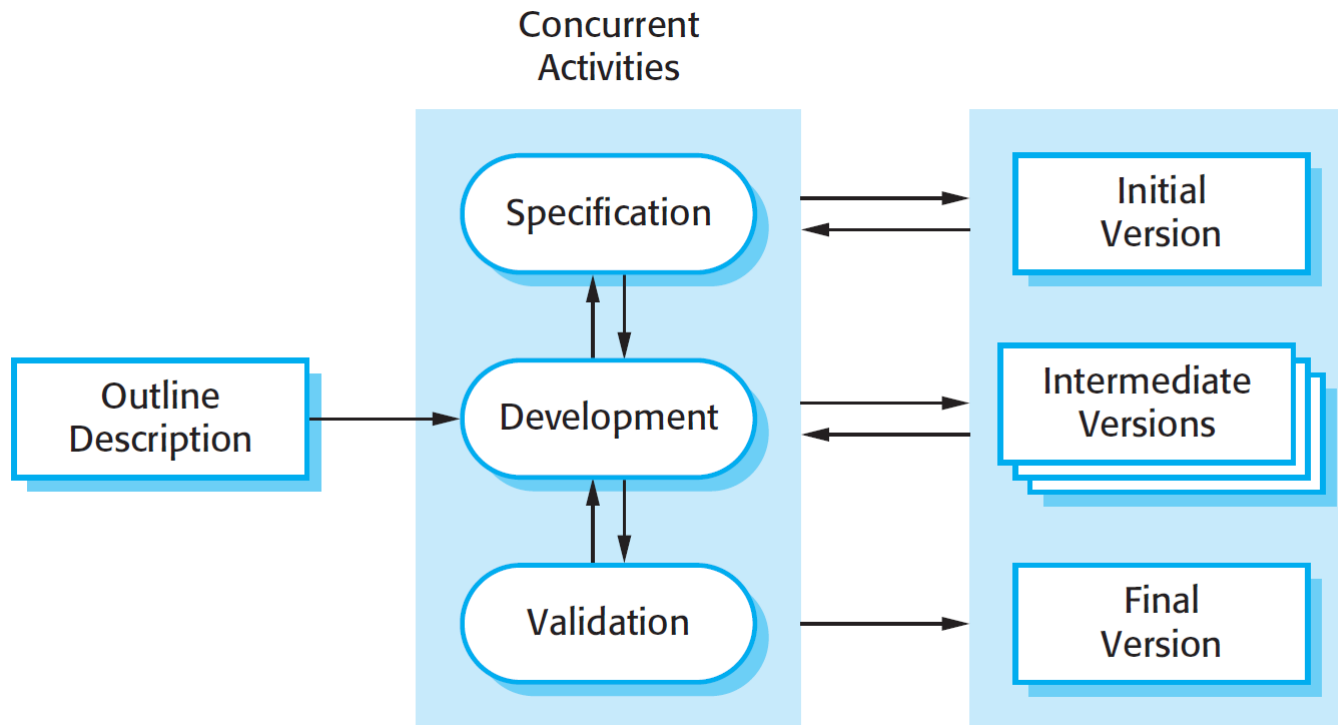
Waterfall model phases

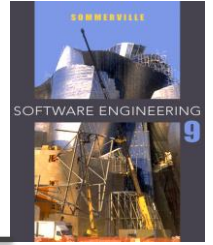
- ✧ There are separate identified phases in the waterfall model:
 - Requirements analysis and definition
 - System and software design
 - Implementation and unit testing
 - Integration and system testing
 - Operation and maintenance
- ✧ Main drawback
 - Difficulty of accommodating change after the process is underway.
 - 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.
 - Appropriate when
 - Requirements are well-understood
 - Changes will be fairly limited during the design process
 - Few business systems have stable requirements.
- ✧ Mostly used for
 - large systems engineering projects
 - System is developed at several sites
 - The plan-driven nature of the waterfall model helps coordinate the work

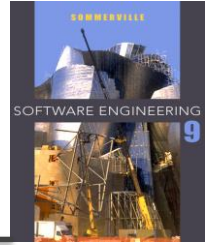
Incremental development





Incremental development benefits

- ✧ **Lower cost to accommodate changes.**
 - The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.
- ✧ **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.

Incremental development problems

✧ Not suitable for systems that are

- Large
- Complex
- Long-lifetime
- Different teams develop different parts

✧ Large complex systems need

- Stable framework
- Clear division of responsibilities

Reuse-oriented software engineering



- ✧ Based on **systematic reuse** where systems are integrated from **existing components or COTS** (Commercial-off-the-shelf) systems.
- ✧ Needs large base of reusable software components
 - Software components can be systems

Reuse-oriented software engineering

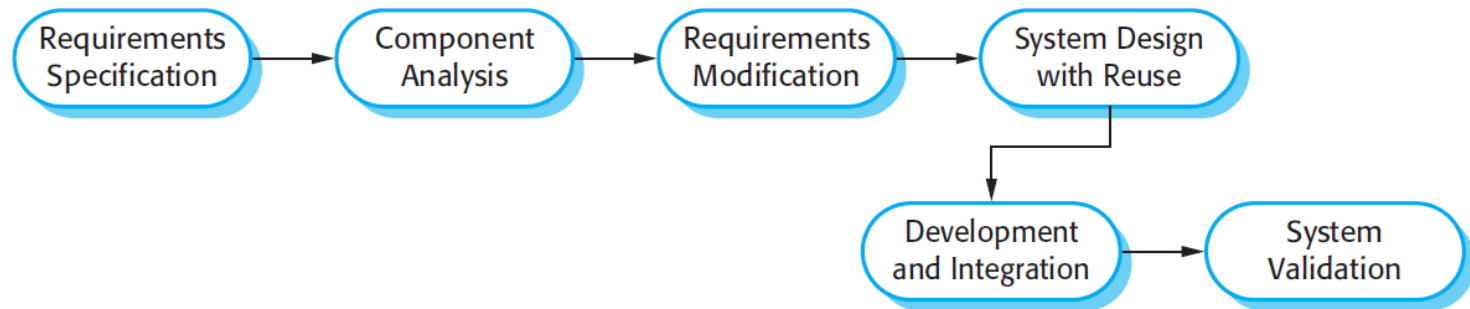


✧ Process stages

- Component analysis;
- Requirements modification;
- System design with reuse;
- Development and integration.

✧ Reuse is now the standard approach for building many types of business system

Reuse-oriented software engineering



Types of software component

- ✧ Web services
- ✧ 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.

Reuse-oriented software engineering



✧ Benefits:

- Reduces amount of software to be developed
 - Reduced cost
 - Reduced risk
 - Usually results in faster delivery of software

✧ Problems:

- Requirements compromises
 - Software may not meet user needs
 - No control over component evolution

Process activities



✧ Four basic process activities:

- Specification, development, validation and evolution are organized differently in different development processes.
- In the waterfall model, **they are organized in sequence**
- In incremental development, **they are inter-leaved**

✧ Real software processes are inter-leaved sequences

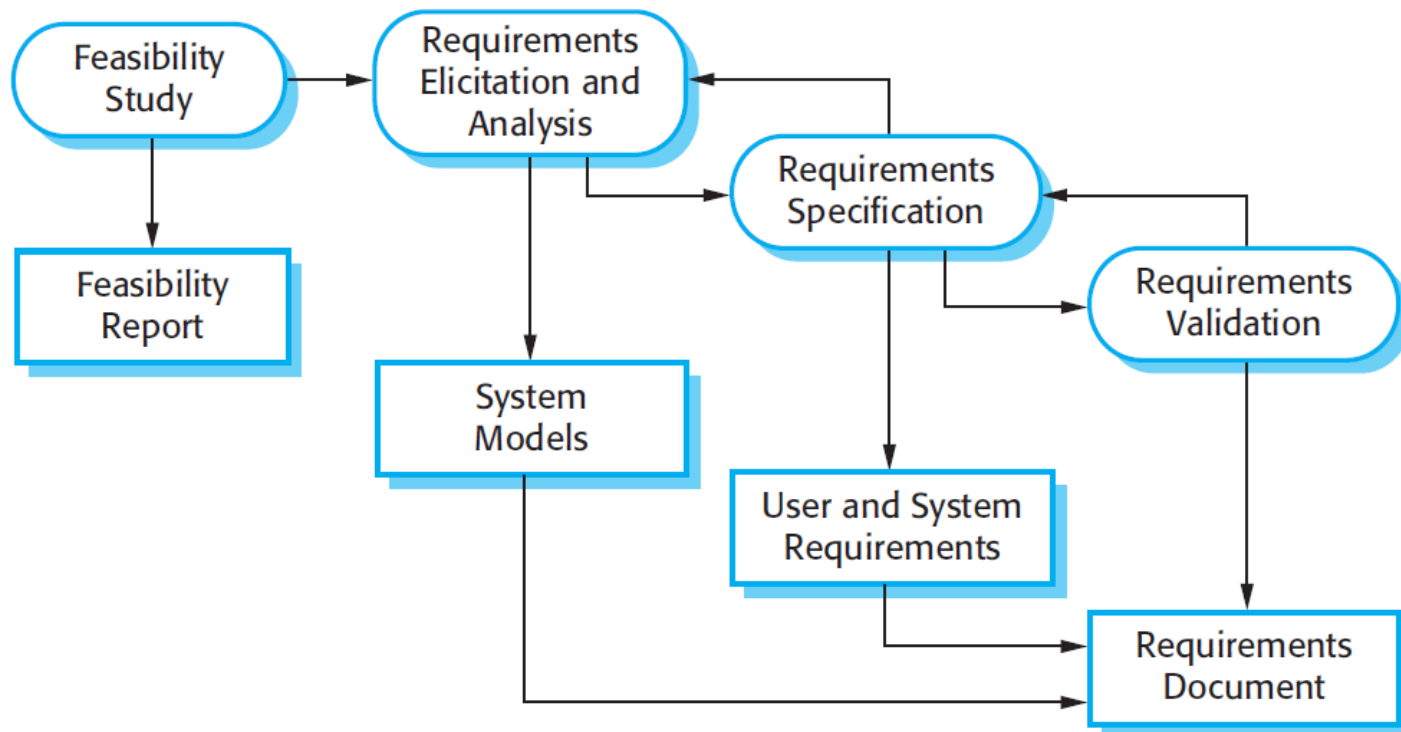
- Technical, collaborative, managerial activities
- To develop a software

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
 - Is it technically and financially feasible to build the system?
 - Requirements elicitation and analysis
 - What do the system stakeholders require or expect from the system?
 - Requirements specification
 - Defining the requirements in detail
 - Requirements validation
 - Checking the validity of the requirements

The requirements engineering process



Software design and implementation



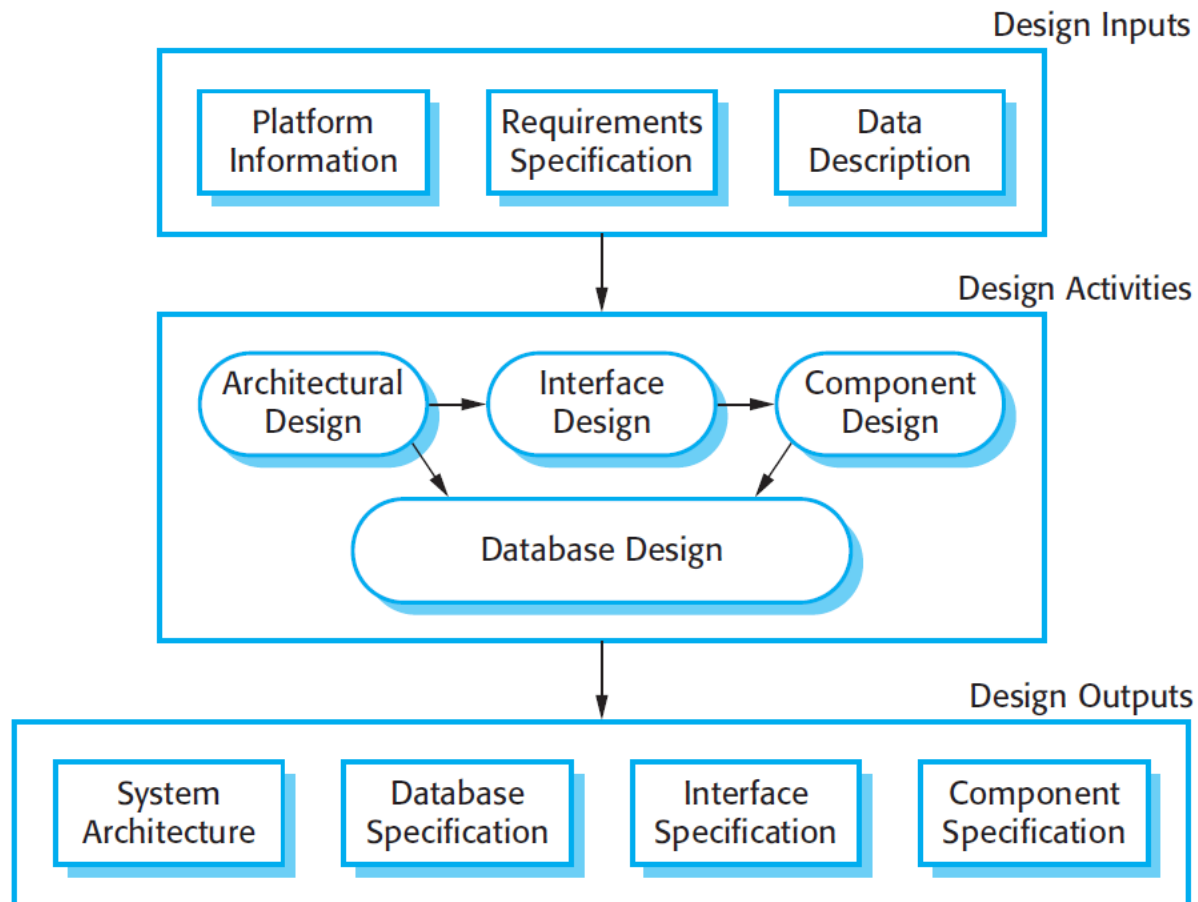
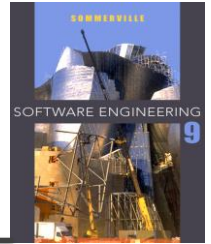
- ✧ The process of **converting the system specification into an executable system.**
- ✧ Software design
 - Design a software structure that realises the specification;
- ✧ Implementation
 - Translate this structure into an executable program;
- ✧ The activities of design and implementation are **closely related** and may be inter-leaved.

Design activities



- ✧ *Architectural design*, where you identify the **overall structure of the system**, the **principal components** (sometimes called sub-systems or modules), **their relationships** and **how they are distributed**.
- ✧ *Interface design*, where you define the **interfaces between system components**.
- ✧ *Component design*, where you take each system component and design how it will operate.
- ✧ *Database design*, where you design the system **data structures** and how these are to be represented in a database.

A general model of the design process



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.

Testing stages

✧ Development or component testing

- Individual components are tested independently;
- Components may be functions or objects or coherent groupings of these entities.

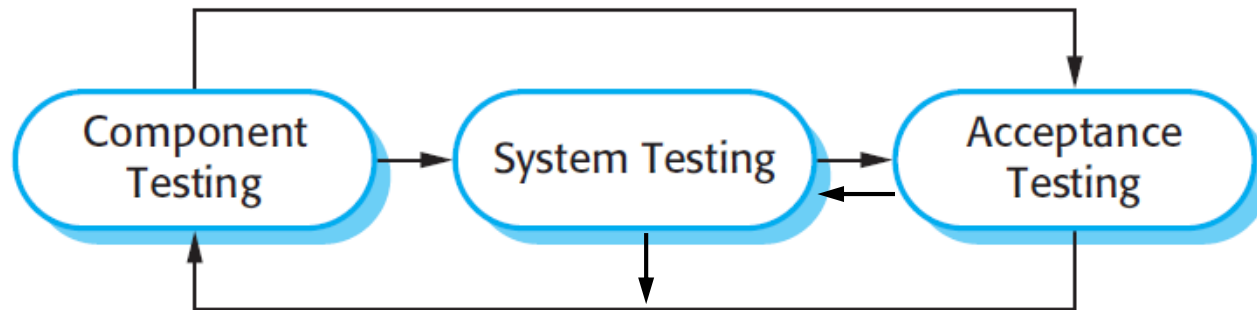
✧ System testing

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

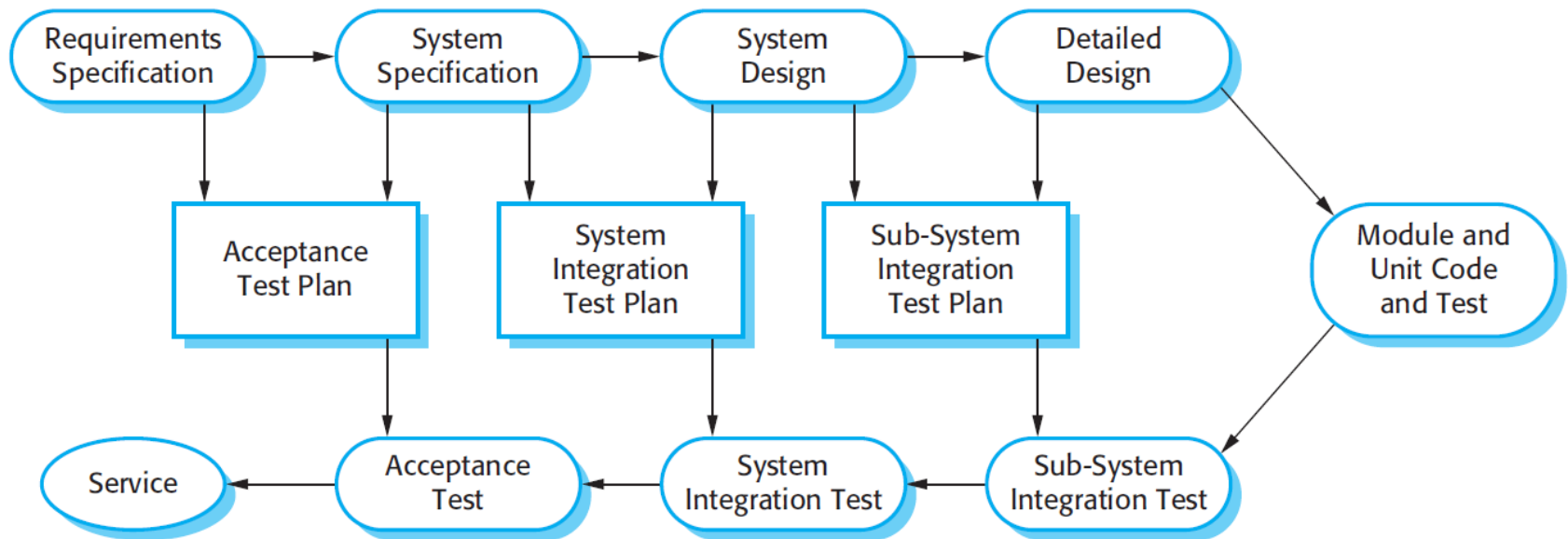
✧ Acceptance testing

- Testing with customer data to check that the system meets the customer's needs.

Stages of testing



Testing phases in a plan-driven software process

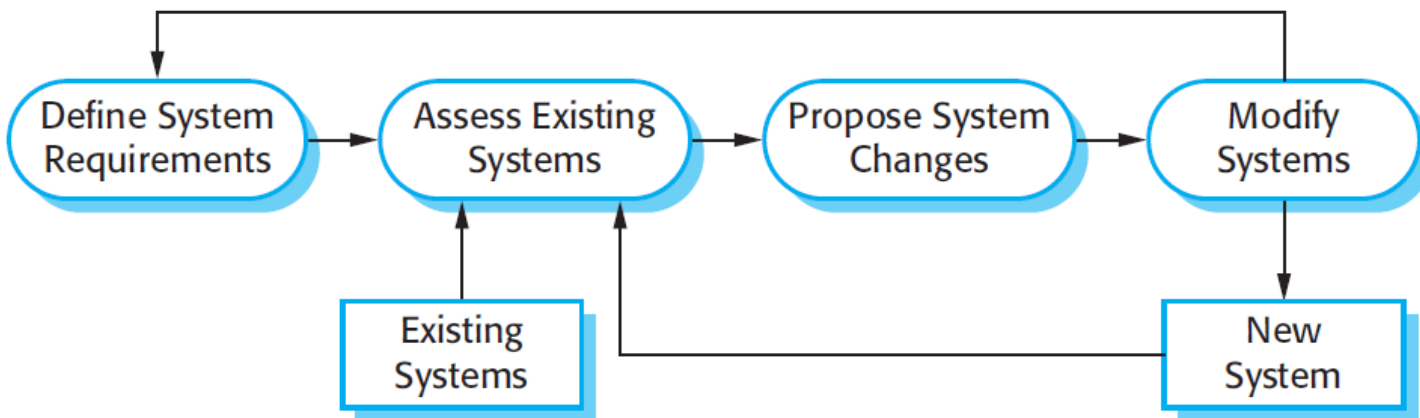


Software evolution



- ✧ Software is inherently **flexible and can change**.
- ✧ As requirements change through **changing business circumstances**, the software that supports the business must also **evolve and change**.
- ✧ Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new.

System evolution



Key points



- ✧ General process models describe the organization of software processes.
 - Examples
 - the 'waterfall' model
 - incremental development
 - reuse-oriented development

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.

Chapter 2 – Software Processes

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

- 1) **Change avoidance**, where the software process includes activities that can anticipate possible changes before significant rework is required.
 - For example, a **prototype** system may be developed to show some key features of the system to customers.



Software prototyping



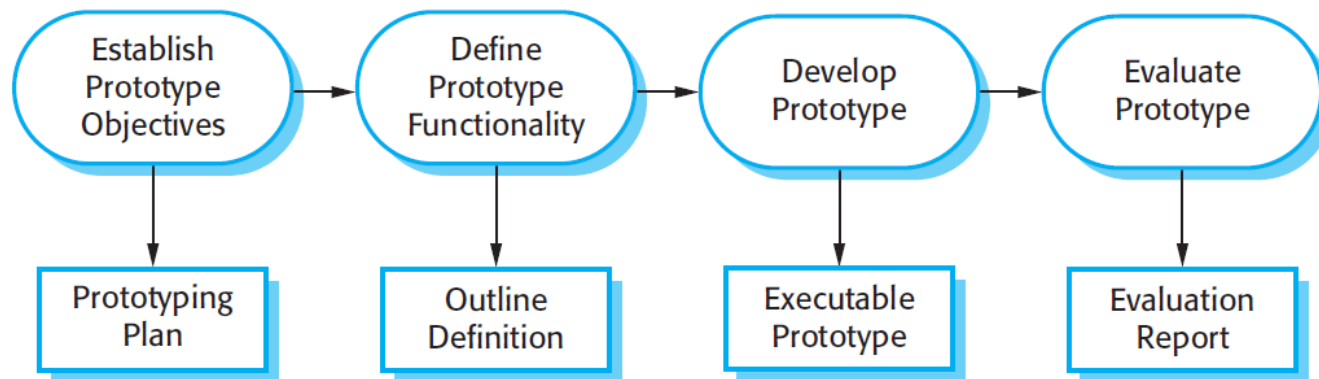
- ✧ A prototype is an **initial version** of a system used to demonstrate **concepts** and try out **design options**.

- ✧ 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**;

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.

Reducing the costs of rework



- 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**.
 - Proposed **changes may be implemented in increments** that have not yet been developed.
 - If this is impossible, then only a single or few increments (a small part of the system) may have be altered to incorporate the change.

Incremental delivery



- ✧ Rather than deliver the system as a single delivery
 - Development and delivery is **broken down into increment**
 - Each increment **delivering part of the required functionality**
- ✧ User requirements are **prioritised**
 - Highest priority requirements are included in early increments
- ✧ Once the development of an increment is started
 - Requirements are frozen
 - Requirements for later increments can continue to evolve

Incremental development and delivery

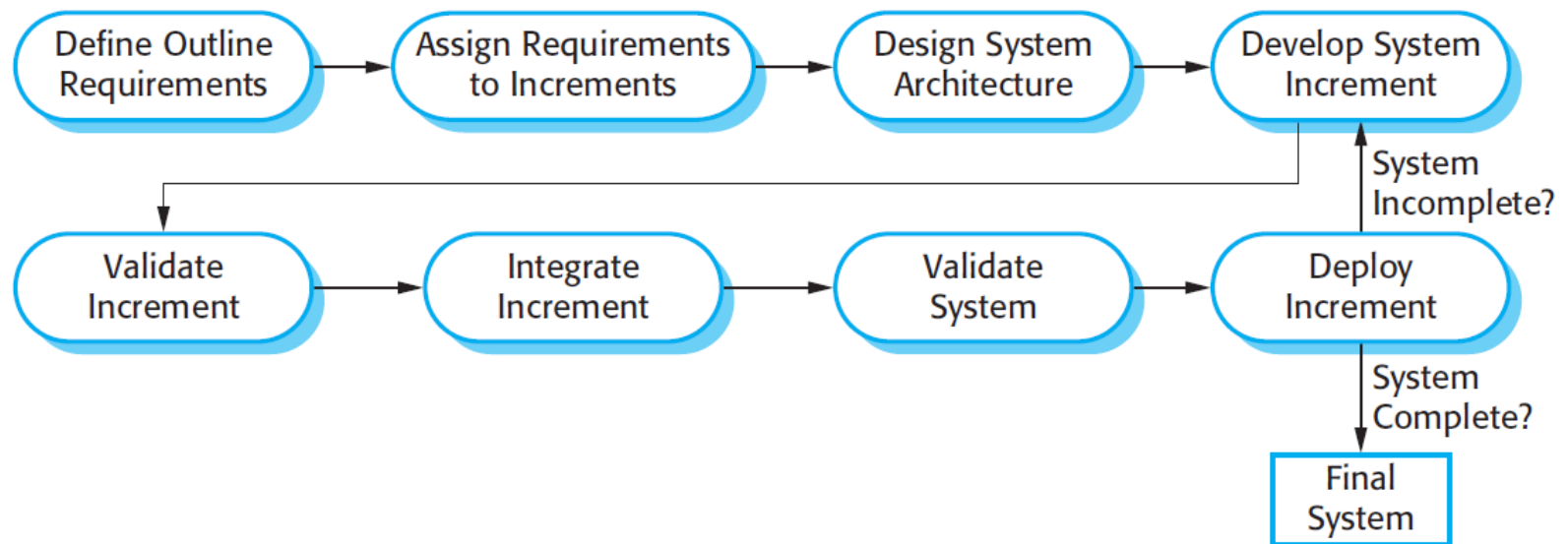
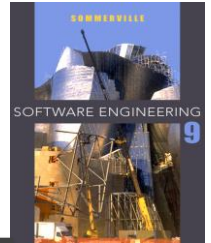
✧ Incremental development

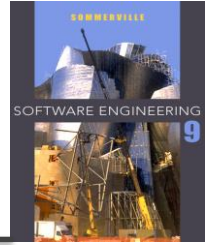
- Develop the system in increments
 - Evaluate each increment before proceeding to the development of the next increment.
- Normal approach used in agile methods
- Evaluation done by user/customer proxy.

✧ Incremental delivery

- Deploy an increment for **use** by end-users.
- More **realistic evaluation** about practical use of software.
- Difficult to implement for replacement systems
 - 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**.

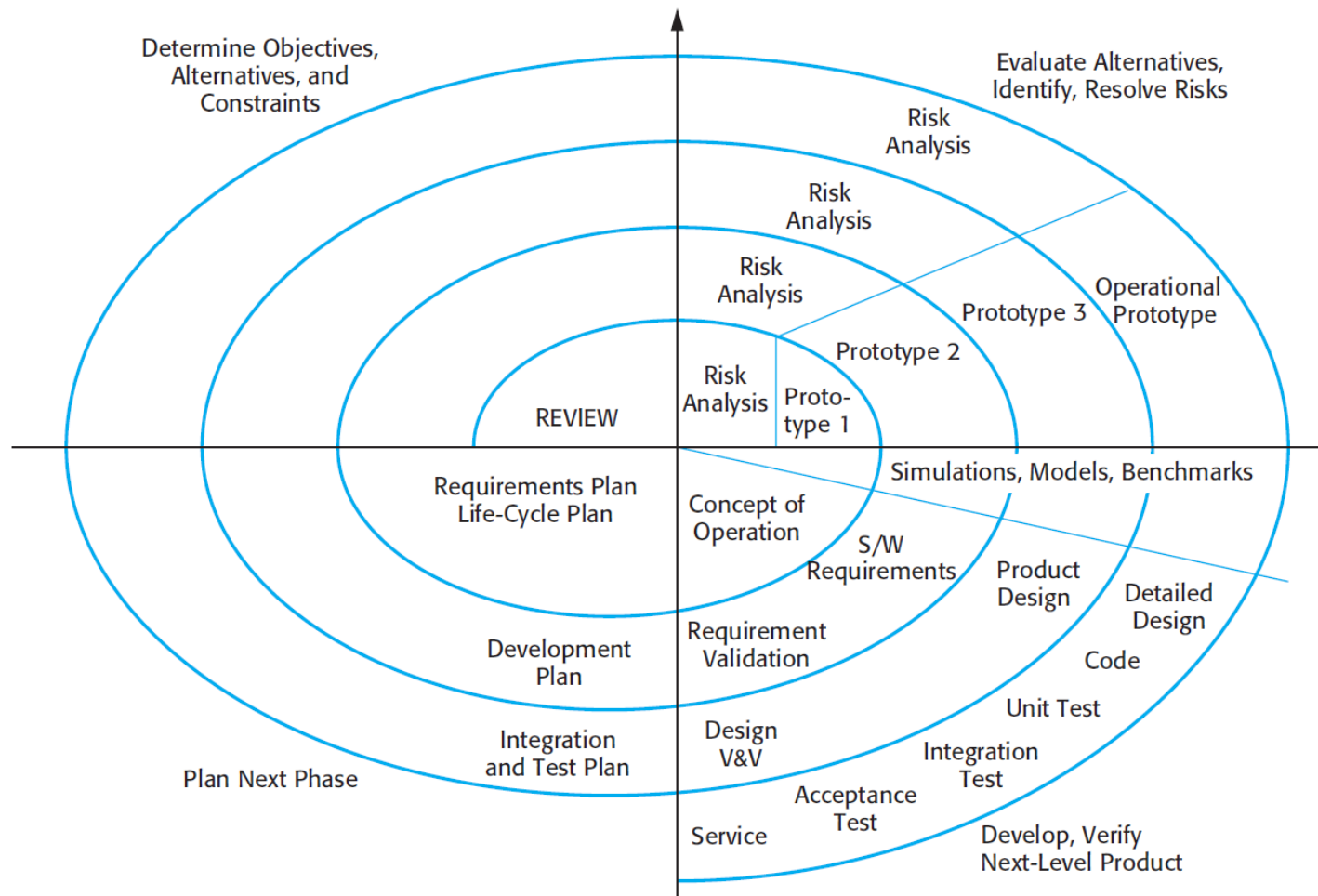
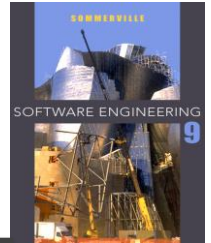
Incremental delivery problems

- ✧ 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.

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.
- Risks are identified.

✧ Risk assessment and reduction

- Identified risks are assessed
- Activities put in place to reduce the key risks.
 - e.g. Inappropriate requirements -> prototype may be developed.

Spiral model sectors



✧ 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

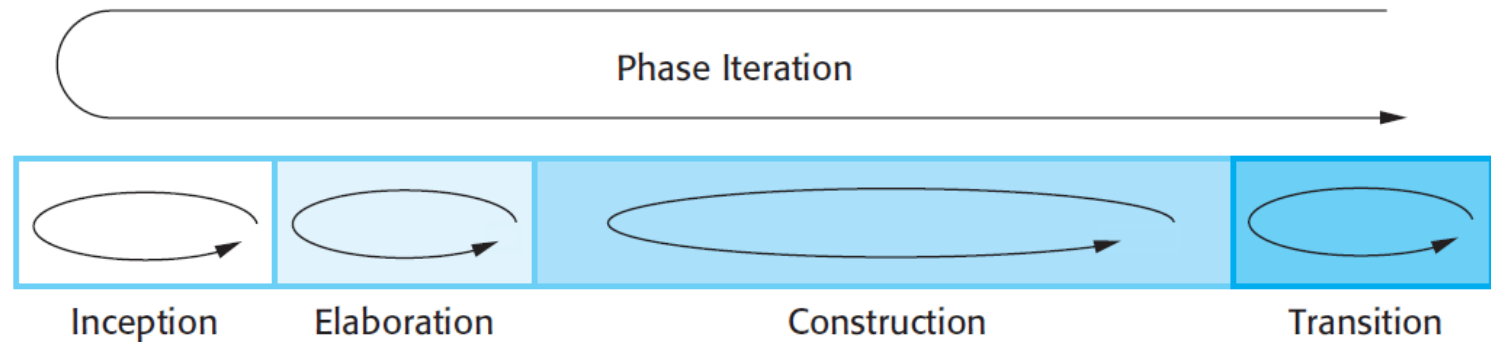


- ✧ Spiral model has been very influential in helping people
 - **Think about iteration** in software processes.
 - Introducing the **risk-driven approach** to development.
 - Safety risks are the main consideration?
 - Development based on formal transformations may be used
 - Main identified risk is sub-system integration?
 - The waterfall model may be the best development model to use.

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 **practive** perspective that suggests good practice.

Phases in the Rational Unified Process



RUP phases



✧ Inception

- Establish the business case for the system.

✧ Elaboration

- Develop an understanding of the problem domain and the system architecture. Requirement models are prepared.

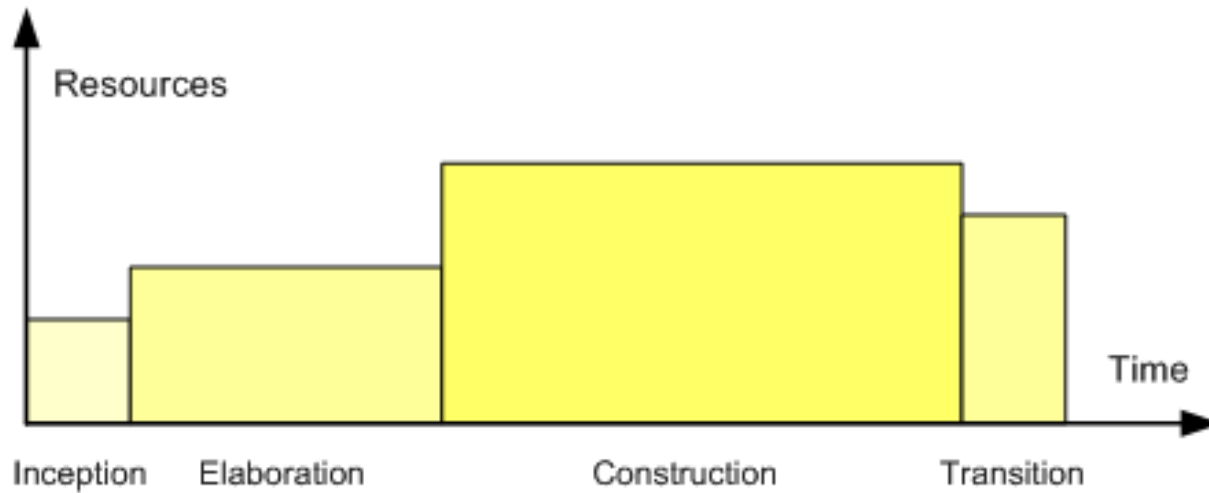
✧ Construction

- System design, programming and testing.

✧ Transition

- Deploy the system in its operating environment.

RUP phases



© GFLewis

RUP iteration



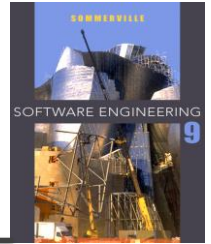
✧ In-phase iteration

- Each phase is iterative with results developed incrementally.

✧ Cross-phase iteration

- As shown by the loop in the RUP model, the whole set of phases may be enacted incrementally.

Static workflows in the Rational Unified Process



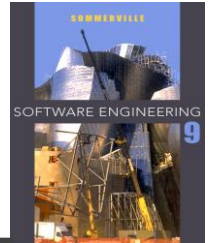
Workflow	Description
Business modelling	The business processes are modelled using business use cases.
Requirements	Actors who interact with the system are identified and use cases are developed to model the system requirements.
Analysis and design	A design model is created and documented using architectural models, component models, object models and sequence models.
Implementation	The components in the system are implemented and structured into implementation sub-systems. Automatic code generation from design models helps accelerate this process.

Static workflows in the Rational Unified Process



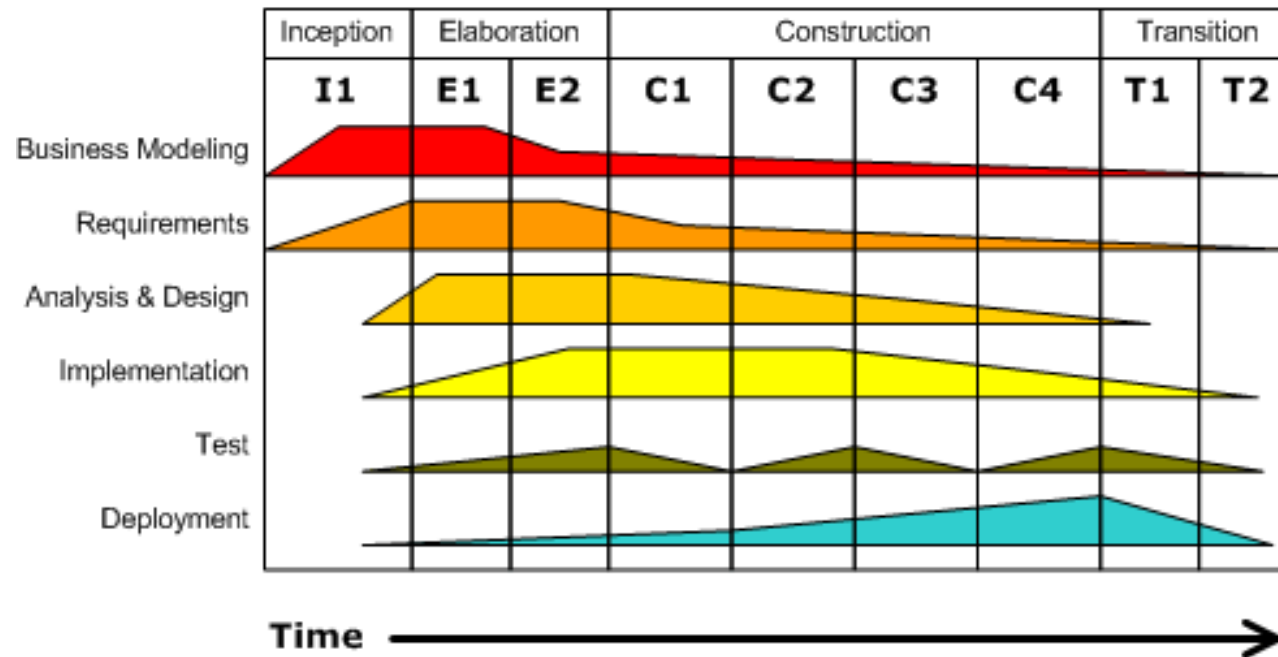
Workflow	Description
Testing	Testing is an iterative process that is carried out in conjunction with implementation. System testing follows the completion of the implementation.
Deployment	A product release is created, distributed to users and installed in their workplace.
Configuration and change management	This supporting workflow managed changes to the system (see Chapter 25).
Project management	This supporting workflow manages the system development (see Chapters 22 and 23).
Environment	This workflow is concerned with making appropriate software tools available to the software development team.

Core static workflows



Iterative Development

Business value is delivered incrementally in time-boxed cross-discipline iterations.



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.
- Analyze impact of change before accepting them

✧ 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.

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 **incremental development and delivery** so that changes may be made without disrupting the system as a whole.
- ✧ 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.