Chapter 1

Consequences of software failures

1. Increasing demands
2. Low expectations

What is software?

Computer programs and associated documentation. Software products may be developed for a particular customer or may be developed for a general market.

What are the attributes of good software?

Good software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable.

What is software engineering?

Software engineering is an engineering discipline that is concerned with all aspects of software production.

What are the fundamental software engineering activities?

- Software specification
- software development
- software validation and
- Software evolution.

What is the difference between software engineering and computer science?

Computer science focuses on theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.

What is the difference between software engineering and system engineering?

System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this more general process.

What are the key challenges facing software engineering?

Coping with increasing diversity, demands for reduced delivery times and developing trustworthy software.

What differences has the web made to software engineering?

The web has led to the availability of software services and the possibility of developing highly distributed service-based systems. Web-based systems development has led to important advances in programming languages and software reuse.

Difference between software engineering and computer science

Computer science focuses on the theory and fundamentals, software engineering is concerned with the practicalities of developing and delivering useful software.

Difference between software engineering and system engineering.

System engineering is concerned with all aspects of computer – based systems development including hardware, software and process engineering. Software engineering is part of the more general process.

Key challenges facing software engineering

- Coping with increasing diversity
- Demands for reduced delivery times.
- Developing trustworthy software

Attributes of a good software

Maintainability

Software should be written in such a way so that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.

Dependability and security

Software dependability includes a range of characteristics including reliability, security and safety. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system.

Efficiency

Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilisation, etc.

Acceptability

Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable and compatible with other systems that they use.

- ✧ Software costs more to maintain than it does to develop. For systems with a long life, maintenance costs may be several times development costs.
- ✧ Software engineering is concerned with cost-effective software development.

The two kinds of software products

- ✧ **Generic products**
    - Stand-alone systems that are marketed and sold to any customer who wishes to buy them.
    - Examples – PC software such as graphics programs, project management tools; CAD software; software for specific markets such as appointments systems for dentists.
- ✧ **Customized products or Bespoke**
    - Software that is commissioned by a specific customer to meet their own needs.
    - Examples – embedded control systems, air traffic control software, traffic monitoring systems.

Differences between generic and customized products

- ✧ **Generic products**
    - The specification of what the software should do is owned by the software developer and decisions on software change are made by the developer.
- ✧ **Customized products**

    - The specification of what the software should do is owned by the customer for the software and they make decisions on software changes that are required.

Software engineering

Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use.

The two key phrases in the definition

- ✧ Engineering discipline
    - Using appropriate theories and methods to solve problems bearing in mind organizational and financial constraints.
- ✧ All aspects of software production
    - Not just technical process of development. Also project management and the development of tools, methods etc. to support software production.

Two importance of software engineering

- ✧ More and more, individuals and society rely on advanced software systems. We need to be able to produce reliable and trustworthy systems economically and quickly.
- ✧ It is usually cheaper, in the long run, to use software engineering methods and techniques for software systems rather than just write the programs as if it was a personal programming project. For most types of system, the majority of costs are the costs of changing the software after it has gone into use.

Software process activities

The systematic approach that is used in software engineering is sometimes called a **software process**.

A **software process** is a sequence of activities that leads to the production of a software product.

There are **four fundamental activities** that are common to all software processes. These activities are:

- ✧ **Software specification**, where customers and engineers define the software that is to be produced and the constraints on its operation.
- ✧ **Software development**, where the software is designed and programmed.
- ✧ **Software validation**, where the software is checked to ensure that it is what the customer requires.
- ✧ **Software evolution**, where the software is modified to reflect changing customer and market requirements.

Software engineering is related to both computer science and systems engineering:

1. Computer science is concerned with the theories and methods that underlie computers and software systems, whereas software engineering is concerned with the practical problems of producing software. Some knowledge of computer science is essential for software engineers in the same way that some knowledge of physics is essential for electrical engineers. Computer science theory, however, is often most applicable to relatively small programs. Elegant theories of computer science cannot always be applied to large, complex problems that require a software solution.

2. System engineering is concerned with all aspects of the development and evolution of complex systems where software plays a major role. System engineering is therefore concerned with hardware development, policy and process design and system deployment, as well as software engineering. System engineers are involved in specifying the system, defining its overall architecture, and then integrating the different parts to create the finished system. They are

less concerned with the engineering of the system components (hardware, software, etc.)

General issues that affect most software

- ✧ **Heterogeneity**
  - Increasingly, systems are required to operate as distributed systems across networks that include different types of computer and mobile devices.
- ✧ **Business and social change**
  - Business and society are changing incredibly quickly as emerging economies develop and new technologies become available. They need to be able to change their existing software and to rapidly develop new software.
- ✧ **Security and trust**
  - As software is intertwined with all aspects of our lives, it is essential that we can trust that software.

Software engineering diversity

- ✧ There are many different types of software system and there is no universal set of software techniques that is applicable to all of these.
- ✧ The software engineering methods and tools used depend on the type of application being developed, the requirements of the customer and the background of the development team.

Software application types

- ✧ **Stand-alone applications**
  - These are application systems that run on a local computer, such as a PC. They include all necessary functionality and do not need to be connected to a network.
- ✧ **Interactive transaction-based applications**
  - Applications that execute on a remote computer and are accessed by users from their own PCs or terminals. These include web applications such as e-commerce applications.

- **Embedded control systems**
    - These are software control systems that control and manage hardware devices. Numerically, there are probably more embedded systems than any other type of system.
- **Batch processing systems**
    - These are business systems that are designed to process data in large batches. They process large numbers of individual inputs to create corresponding outputs.
- **Entertainment systems**
    - These are systems that are primarily for personal use and which are intended to entertain the user.
- **Systems for modeling and simulation**
    - These are systems that are developed by scientists and engineers to model physical processes or situations, which include many, separate, interacting objects.
- **Data collection systems**
    - These are systems that collect data from their environment using a set of sensors and send that data to other systems for processing.
- **Systems of systems**
    - These are systems that are composed of a number of other software systems.

Software engineering fundamentals

- Some fundamental principles apply to all types of software system, irrespective of the development techniques used:
    - Systems should be developed using a managed and understood development process. Of course, different processes are used for different types of software.
    - Dependability and performance are important for all types of system.
    - Understanding and managing the software specification and requirements (what the software should do) are important.

        - Where appropriate, you should reuse software that has already been developed rather than write new software.

Software engineering ethics

- ✧ Software engineering involves wider responsibilities than simply the application of technical skills.
- ✧ Software engineers must behave in an honest and ethically responsible way if they are to be respected as professionals.
- ✧ Ethical behaviour is more than simply upholding the law but involves following a set of principles that are morally correct.
- ✧ Confidentiality
    - ✧ Engineers should normally respect the confidentiality of their employers or clients irrespective of whether or not a formal confidentiality agreement has been signed.
- ✧ Competence
    - ✧ Engineers should not misrepresent their level of competence. They should not knowingly accept work which is outwith their competence.

Exercises

1. **Explain why professional software is not just the programs that are developed for a customer.**
   Professional software is not just the program developed for the customer. In fact professional software along with executable code will also contain documentation, configuration of data – all that is required to make the operation of programs correct and accurate. Generally such a software do contain separate files for program and configuration purposes. Documentation of the professional software consists of several topics such as structure of the system, user documentation and all the information required for the user to work on that.

2. **What is the most important difference between generic software product development and custom software development? What might this mean in practice for users of generic software products?**

   The essential difference is that in generic software product development, the specification is owned by the product developer. For custom product development, the specification is owned and controlled by the customer. The implications of this are significant – the developer can quickly decide to change the specification in response to some external change (e.g. a competing product) but, when the customer owns the specification, changes have to be negotiated between the customer and the developer and may have contractual implications.

   For users of generic products, this means they have no control over the software specification so cannot control the evolution of the product. The developer may decide to include/exclude features and change the user interface. This could have implications for the user's business processes and add extra training costs when new versions of the system are installed. It also may limit the customer's flexibility to change their own business processes.

3. **What are the four important attributes that all professional software should have? Suggest four other attributes that may sometimes be significant**.
   i. Four important attributes are
      - Maintainability
      - Dependability
      - performance and
      - usability.
   ii. Other attributes that may be significant could be
      - reusability (can it be reused in other applications)
      - distributability (can it be distributed over a network of processors)

        - portability (can it operate on multiple platforms e.g laptop and mobile platforms)
        - and inter-operability (can it work with a wide range of other software systems)

4. **Apart from the challenges of heterogeneity, business and social change, and trust and security, identify other problems and challenges that software engineering is likely to face in the 21st century (Hint: think about the environment).**

   Problems and challenges for software engineering
   - Developing systems that are energy-efficient. This makes them more usable on low power mobile devices and helps reduce the overall carbon footprint of IT equipment.
   - Developing validation techniques for simulation systems (which will be essential in predicting the extent and planning for climate change).
   - Developing systems for multicultural use
   - Developing systems that can be adapted quickly to new business needs
   - Designing systems for outsourced development
   - Developing systems that are resistant to attack
   - Developing systems that can be adapted and configured by end-users
   - Finding ways of testing, validating and maintaining end-user developed system.

5. **Based on your own knowledge of some of the application types discussed in section 1.1.2, explain, with examples, why different application types require specialized software engineering techniques to support their design and development.**

   Different application types require the use of different development techniques for a number of reasons:
   - Costs and frequency of change. Some systems (such as embedded systems in consumer devices) are extremely expensive to change; others, must change frequently in response to changing requirements (e.g. business systems). Systems which are very expensive to change

need extensive upfront analysis to ensure that the requirements are consistent and extensive validation to ensure that the system meets its specification. This is not cost effective for systems that change very rapidly.

- The most important 'non-functional' requirements. Different systems have different priorities for non-functional requirements. For example, a real-time control system in an aircraft has safety as its principal priority; an interactive game has responsiveness and usability as its priority. The techniques used to achieve safety are not required for interactive gaming; the extensive UI design required for games is not needed in safety-critical control systems.
  3. The software lifetime and delivery schedule. Some software systems have a relatively short lifetime (many web-based systems), others have a lifetime of tens of years (large command and control systems). Some systems have to be delivered quickly if they are to be useful. The techniques used to develop short-lifetime, rapid delivery systems (e.g. use of scripting languages, prototyping, etc.) are inappropriate for long-lifetime systems which require
  techniques that allow for long-term support such as design modelling

6. **Explain why there are fundamental ideas of software engineering that apply to all types of software systems.**
   Because of all software systems have common quality attributes including availability, modifiability, performance, security and safety, testability and usability, the fundamental software ideas provide common solutions or tactics to support those qualities.

7. **Explain how the universal use of the Web has changed software systems.**
   Enhance technological innovation. Services like learning activities in which engineer can share technology of developing software
   The web was created for research and military use at first where only internal companies where able to use it, but now it has evolved where ordinary users can access information of all kinds and purchase items for prominent companies and entrepreneurs worldwide. This is where Software Developer comes in developing ecommerce systems and much more.

Chapter 2

Software processes

A structured set of activities required to develop a software system.

A **software process** is a set of related activities that leads to the production of a software product. These activities may involve the development of software from scratch in a standard programming language like Java or C. However, business applications are not necessarily developed in this way.

Fundamental software processes:

- **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.

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 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

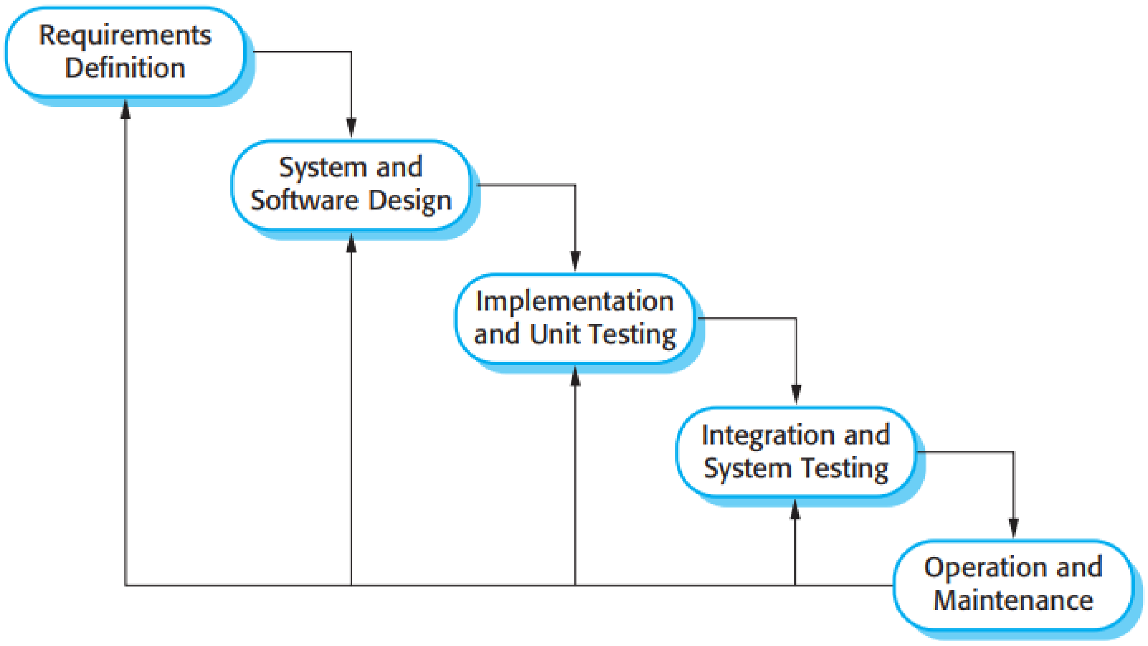
Software process model.

A **software process model** is an abstract representation of a process. It presents a description of a process from some particular perspective.

Software process model

✧ **The waterfall model**

- Plan-driven model. Separate and distinct phases of specification and development.
- This takes the fundamental process activities of specification, development, validation, and evolution and represents them as separate process phases such as requirements specification, software design, implementation, testing, and so on.


Requirements Definition
System and Software Design
Implementation and Unit Testing
Integration and System Testing
Operation and Maintenance

- ✧ **Incremental development**
    - Specification, development and validation are interleaved. May be plan-driven or agile.
    - This approach interleaves the activities of specification, development, and validation. The system is developed as a series of versions (increments), with each version adding functionality to the previous version
- ✧ **Reuse-oriented software engineering**
    - The system is assembled from existing components. May be plan-driven or agile.
    - This approach is based on the existence of a significant number of reusable components. The system development process focuses on integrating these components into a system rather than developing them from scratch.

The waterfall model.

Because of the cascade from one phase to another, this model is known as the 'waterfall model' or software life cycle. The waterfall model is an example of a plan-driven process—in principle, you must plan and schedule all of the process activities before starting work on them.

The principal stages of the waterfall model directly reflect the fundamental development activities:

- ***Requirements analysis and definition*** The system's services, constraints, and goals are established by consultation with system users. They are then defined in detail and serve as a system specification.

- ***System and software design*** The systems design process allocates the requirements to either hardware or software systems by establishing an overall system architecture. Software design involves identifying and describing the fundamental software system abstractions and their relationships.

- ***Implementation and unit testing*** During this stage, the software design is realized as a set of programs or program units. Unit testing involves verifying that each unit meets its specification.

- ***Integration and system testing*** The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the customer.

- ***Operation and maintenance*** Normally (although not necessarily), this is the longest life cycle phase. The system is installed and put into practical use. Maintenance involves correcting errors which were not discovered in earlier stages of the life cycle, improving the implementation of system units and enhancing the system's services as new requirements are discovered.

✧ **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.

Other problems of waterfall or disadvantages

✧ Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.

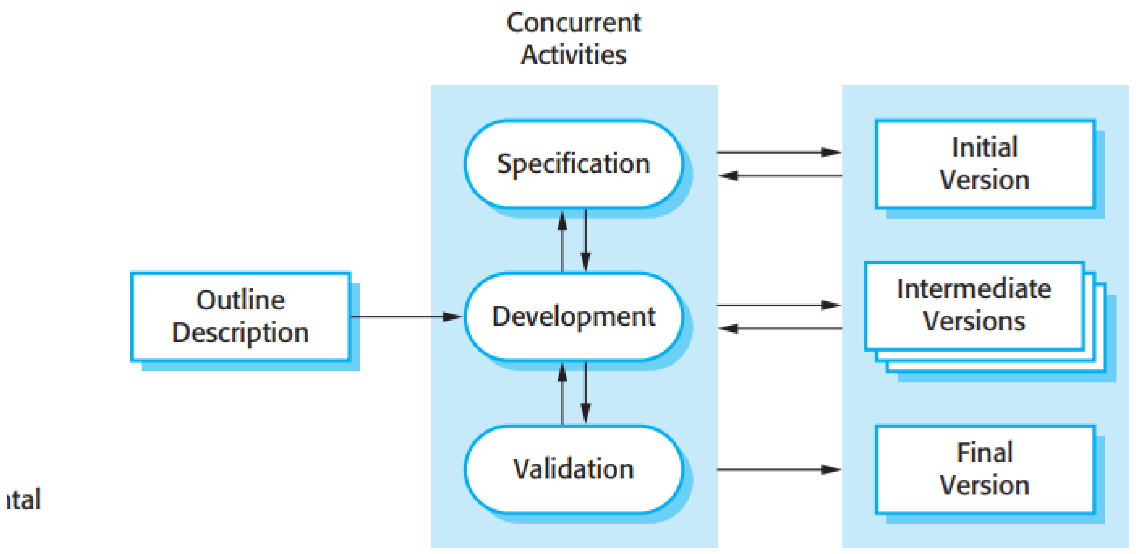
- Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
- Few business systems have stable requirements.

✧ 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

Incremental development is based on the idea of developing an initial implementation, exposing this to user comment and evolving it through several versions until an adequate system has been developed.



Incremental software development, which is a fundamental part of agile approaches, is better than a waterfall approach for most business, e-commerce, and personal systems.

Incremental development benefits

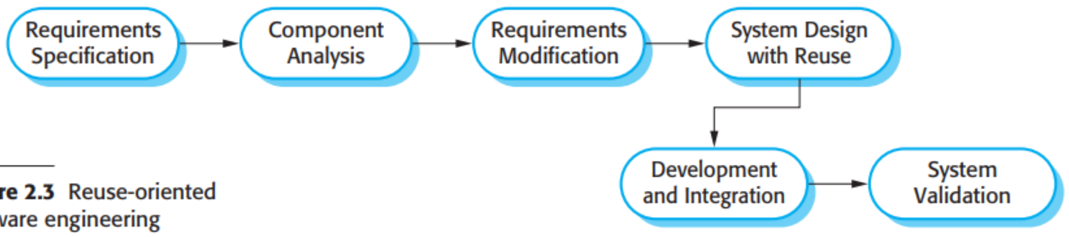
- ✧ The cost of accommodating changing customer requirements is reduced.
    - ▪ The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model.
- ✧ It is easier to get customer feedback on the development work that has been done.
    - ▪ Customers 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.

In the majority of software projects, there is some software reuse. This often happens informally when people working on the project know of designs or code that are similar to what is required. They look for these, modify them as needed, and incorporate them into their system.


Requirements Specification
Component Analysis
Requirements Modification
System Design with Reuse
Development and Integration
System Validation

ure 2.3 Reuse-oriented tware engineering

These stages are:

1. ***Component analysis*** Given the requirements specification, a search is made for components to implement that specification. Usually, there is no exact match and the components that may be used only provide some of the functionality required.

2. ***Requirements modification*** During this stage, the requirements are analyzed using information about the components that have been discovered. They are then

modified to reflect the available components. Where modifications are impossible, the component analysis activity may be re-entered to search for alternative solutions.

3**. *System design with reuse*** During this phase, the framework of the system is designed or an existing framework is reused. The designers take into account the components that are reused and organize the framework to cater for this. Some new software may have to be designed if reusable components are not available.

4. ***Development and integration*** Software that cannot be externally procured is developed, and the components and COTS systems are integrated to create the new system. System integration, in this model, may be part of the development process rather than a separate activity.

There are three types of software component that may be used in a reuse-oriented process:

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

2. Collections of objects that are developed as a package to be integrated with a component framework such as .NET or J2EE.

3. Stand-alone software systems that are configured for use in a particular environment.

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.

In the waterfall model, they are organized in sequence, whereas in incremental development they are inter-leaved.

Software specification

- **Software specification or requirements engineering** is the process of understanding and defining what services are required from the system and identifying the constraints on the system's operation and development.

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

There are four main activities in the requirements engineering process:

- **Feasibility study**
  - Is it technically and financially feasible to build the system?
  - An estimate is made of whether the identified user needs may be satisfied using current software and hardware technologies.
- **Requirements elicitation and analysis**
  - What do the system stakeholders require or expect from the system?
  - This is the process of deriving the system requirements through observation of existing systems, discussions with potential users and procurers, task analysis, and so on.
- **Requirements specification**
  - Defining the requirements in detail
  - Two types of requirements may be included in this document.
  - User requirements are abstract statements of the system requirements for the customer and end-user of the system;
  - system requirements are a
    more detailed description of the functionality to be provided

- **Requirements validation**
    - Checking the validity of the requirements
    - This activity checks the requirements for realism, consistency, and completeness. During this process, errors in the requirements document are inevitably discovered. It must then be modified to correct these problems.

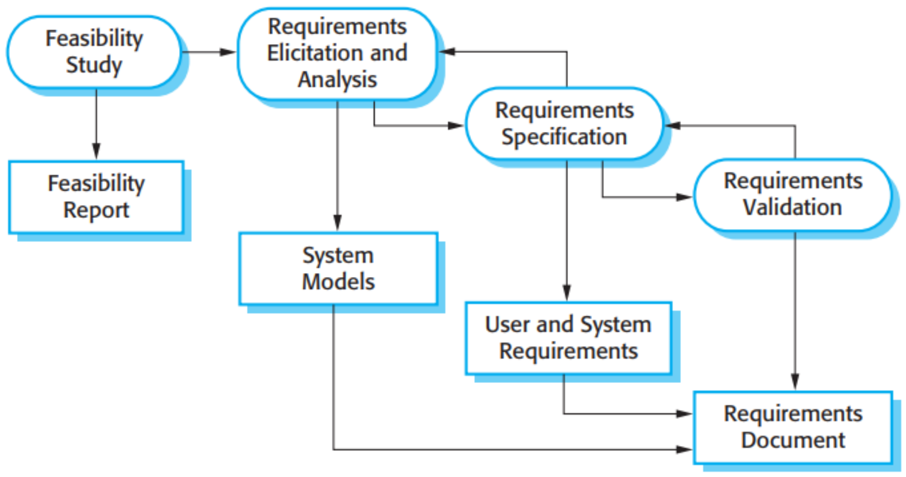
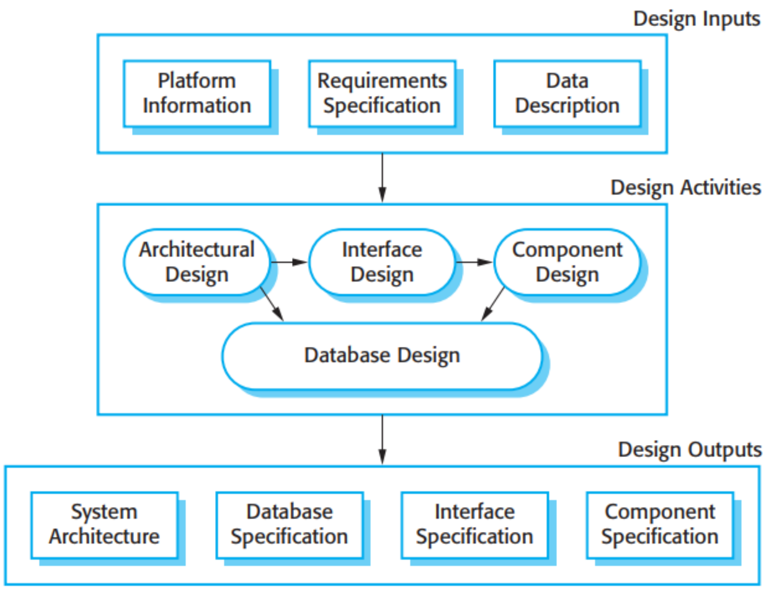


Figure 2.4 The requirements engineering process

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.



igure 2.5 A general nodel of the design process

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.

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.

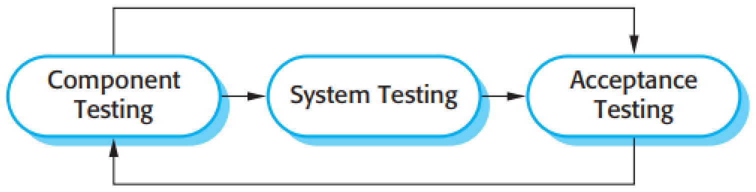
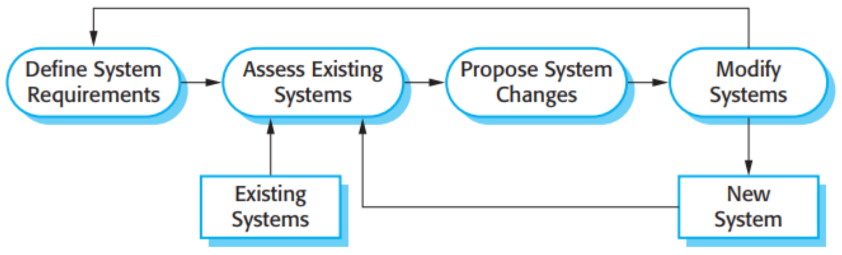


Figure 2.6 Stages of testing

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.
- ✧ **Acceptance testing**
    - Testing with customer data to check that the system meets the customer's needs.

Software evolution



igure 2.8 System volution

- ✧ 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.
- ✧ **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.
- ✧ **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.

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 cost of Work

Change avoidance, where the software process includes activities that can anticipate possible changes before significant rework is required.

Change tolerance, where the process is designed so that changes can be accommodated at relatively low cost.

Two ways of coping with change and changing system requirements.

These are:
1. **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 some design decisions. This supports change avoidance 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.

2. **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 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.

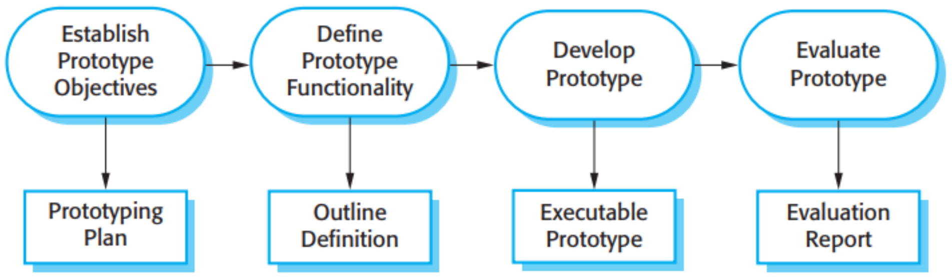


Figure 2.9 The process of prototype development

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 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

- ✧ **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.
  -

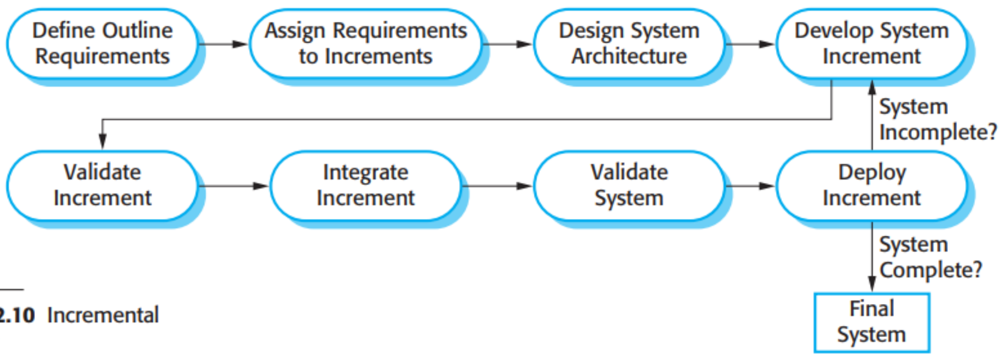


Figure 2.10 Incremental delivery

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.
- The essence of iterative processes is that the specification is developed in conjunction with the software.

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.

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.

Advantages

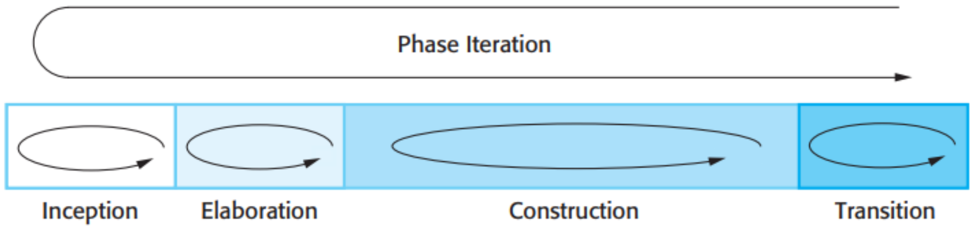
- ✧ Spiral model has been very influential in helping people think about iteration in software processes and introducing the risk-driven approach to development.

The Rational Unified Process

- ✧ A modern generic process derived from the work on the UML and associated process.
- ✧ **Normally described from 3 perspectives**
    - ▪ A dynamic perspective that shows phases over time;
    - ▪ A static perspective that shows process activities;
    - ▪ A practivce perspective that suggests good practice.

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.

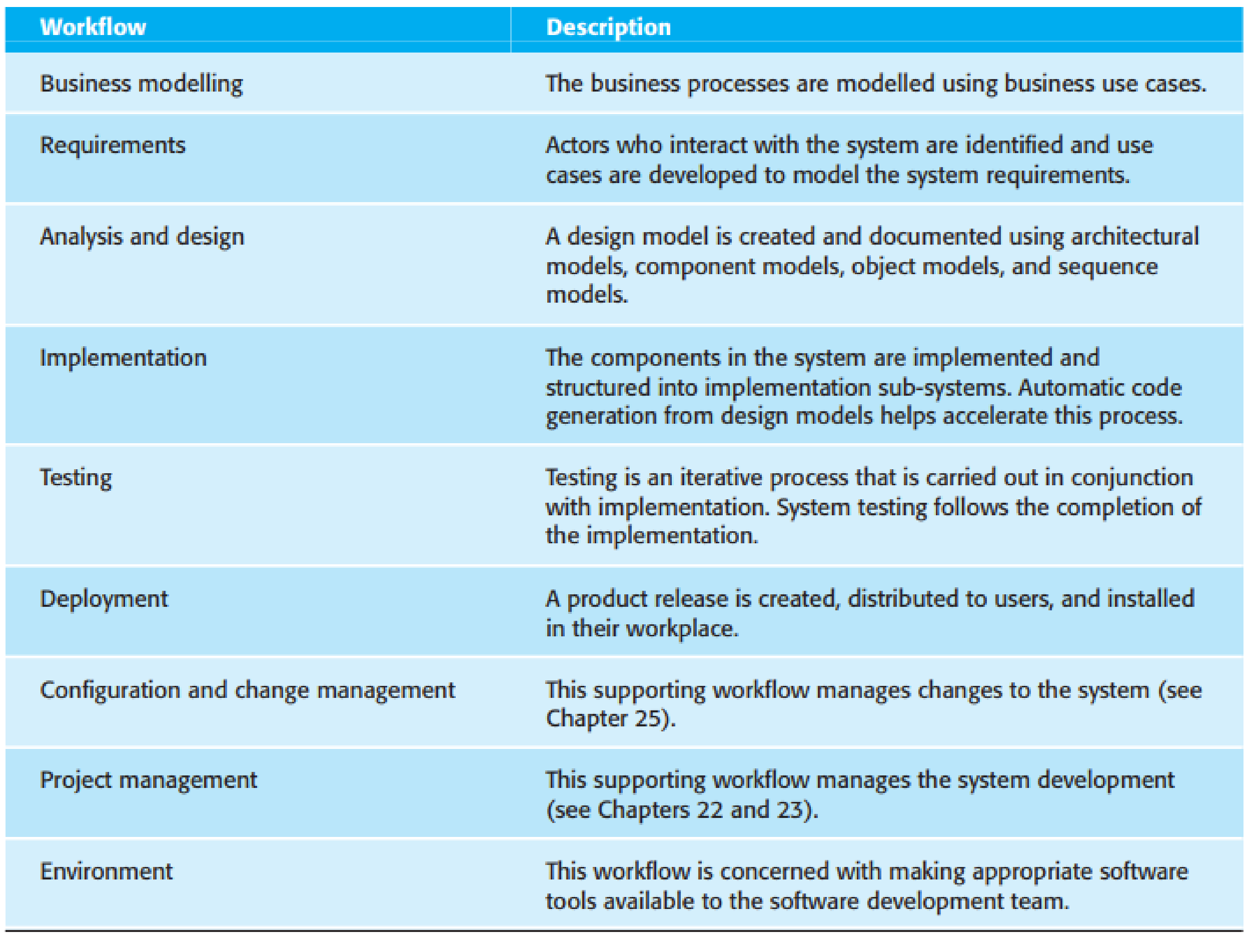


gure 2.12 Phases in
ıe Rational Unified
ocess

Rup iterations

- ✧ 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 workflow in 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.
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 (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

- ✧ 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.
- ✧ 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.

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.**

Exercises

1. Giving reasons for your answer based on the type of system being developed, suggest the most appropriate generic software process model that might be used as a basis for managing the development of the following systems:

1. A system to control anti-lock braking in a car
2. A virtual reality system to support software maintenance
3. A university accounting system that replaces an existing system manual paper based system.
4. An interactive travel planning system that helps users plan journey with the lowest environment impact
5. The ATM system
6. MTN mobile money system
7. A university accounting system that replaces an existing system.

Answer

1. This is a safety-critical system so requires a lot of up-front analysis before implementation. It certainly needs a plan-driven approach to development with the requirements carefully analyzed. **A waterfall model is therefore the most appropriate approach to use, perhaps with formal transformations between the different development stages.**
2. This is a system where the requirements will change and there will be an extensive user interface components. **Incremental development with, perhaps, some UI prototyping is the most appropriate model. An agile process may be used.**
3. This is a system whose requirements are fairly well-known and which will be used in an environment in conjunction with lots of other systems such as a research grant management system. **Therefore, a reuse-based approach is likely to be appropriate for this.**
4. Interactive travel planning system with a complex user interface but which must be stable and reliable. **An incremental development approach is the most appropriate as the system requirements will change as real user experience with the system is gained.**
5.
6.
7. **Reuse-based approach**

2. Explain why incremental development is the most effective approach for developing business software systems. Why is this model less appropriate for real-time systems engineering?

   In a reuse based process, you need two requirements engineering activities because it is essential to adapt the system requirements according to the capabilities of the system/components to be reused. These activities are:

   1. An initial activity where you understand the function of the system and set out broad requirements for what the system should do. These should be expressed in sufficient detail that you can use them as a basis for deciding of a system/component satisfies some of the requirements and so can be reused.

   2. Once systems/components have been selected, you need a more detailed requirements engineering activity to check that the features of the reused software meet the business needs and to identify changes and additions that are required.

3. **Explain why design conflicts might arise when designing an architecture for which both availability and security requirements are the most important non-functional requirements**.

Fundamentally, to provide availability, you need to have (a) replicated components in the architecture so that in the event of one component failing, you can switch immediately to a backup component.

You also need to have several copies of the data that is being processed. Security requires minimizing the number of copies of the data and, wherever possible, adopting an architecture where each component only knows as much as it needs to, to do its job. This reduces the chance of intruders accessing the data.

Therefore, there is a fundamental architectural conflict between availability (replication, several copies) and security (specialization, minimal copies). The system architect has to find the best compromise between these fundamentally opposing requirements.

4. **Why are iterations usually limited when the waterfall model is used?**

The waterfall model is a document-driven model with documents produced at the end of each phase. Because of the cost of producing and approving documents, iterations and costly and involve significant rework. Hence, they are limited.

5. **Distinguish between the waterfall model and incremental model and give an example each of suitable software application that each might be used to develop type?**

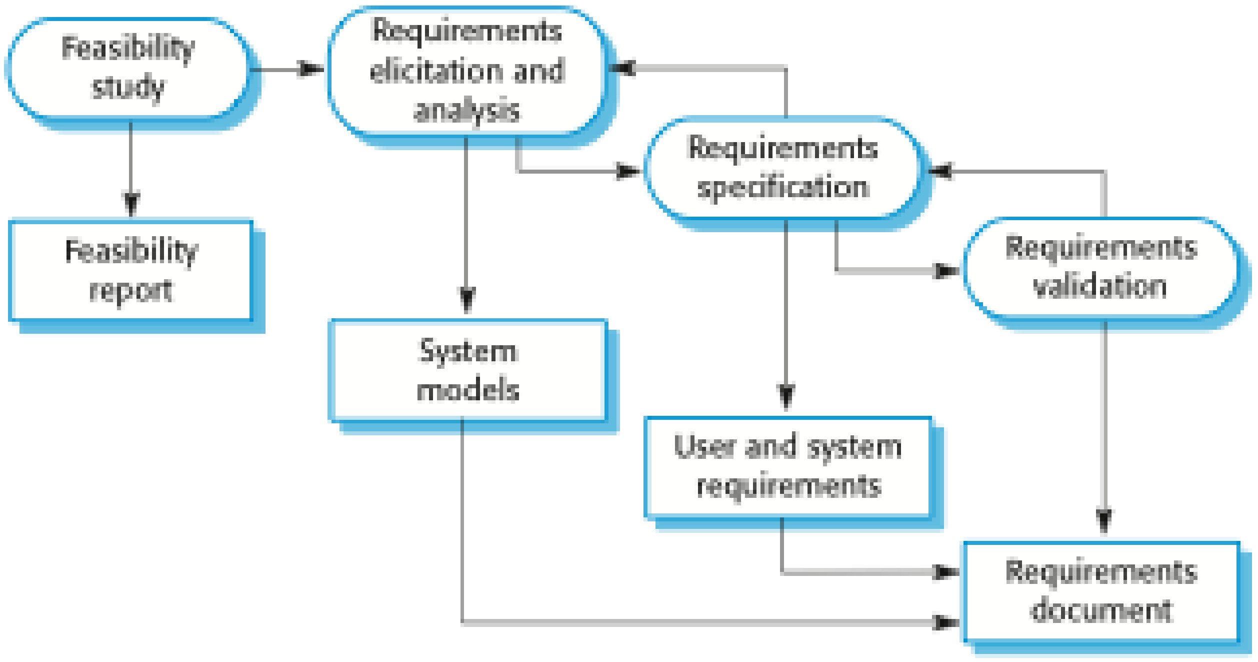
The waterfall model is a document-driven model with documents produced at the end of each phase.
This takes the fundamental process activities of specification, development, validation, and evolution and represents them as separate process phases such as requirements specification, software design, implementation, testing, and so on.
Example of software suitable is anti-lock braking system.

While Incremental model approach interleaves the activities of specification, development, and validation. The system is developed as a series of versions (increments), with each version adding functionality to the previous version.
Example of suitable software is the virtual reality system.

6. **Briefly explain the principal stages of the requirements engineering process with a suitable diagram?**

✧ 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


Feasibility study
Requirements elicitation and analysis
Requirements specification
Requirements validation
Feasibility report
System models
User and system requirements
Requirements document