SOFTWARE ENGINEERING

Dr. Emad Ali Ahmed

Contents

1 Introduction to Software Engineering

Objectives.

understand what software engineering is and why it is important

By the end of this lecture, you will...

- ☒ Understand what software engineering is.
- ☒ Understand why software engineering is important.
- ☒ Know answers to key questions related to the software engineering discipline.

Preface

We can't run the modern world without software. National infrastructures and utilities are controlled by computer-based systems and most electrical products include a computer and controlling software. Industrial manufacturing and distribution are completely computerized, as is the financial system. Entertainment, including the music industry, computer games, and film and television, is software intensive. Therefore, software engineering is essential for the functioning of national and inter- national societies.

Software systems are abstract and intangible. They are not constrained by the properties of materials, governed by physical laws, or by manufacturing processes. This simplifies software engineering, as there are no natural limits to the potential of software. However, because of the lack of physical constraints, software systems can quickly become extremely complex, difficult to understand, and expensive to change. There are many different types of software systems, from simple embedded systems to complex, worldwide information systems. It is pointless to look for universal notations, methods, or techniques for software engineering because different types of software require different approaches. Developing an organizational information system is completely different from developing a controller for a scientific instrument. Neither of these systems has much in common with a graphics-intensive computer game. All of these applications need software engineering; they do not all need the same software engineering techniques.

Software engineers can be rightly proud of their achievements. Of course, we still have problems developing complex software but, without software engineering, we would not have explored space, would not have the Internet or modern telecommunications. All forms of travel would be more dangerous and expensive. Software engineering has contributed a great deal and I am convinced that its contributions in the 21st century will be even greater.

Activity

Think about all the devices and systems that you encounter in your everyday life which have software controlling them…

List as many as you can

Virtually all countries depend on complex computer-based systems.

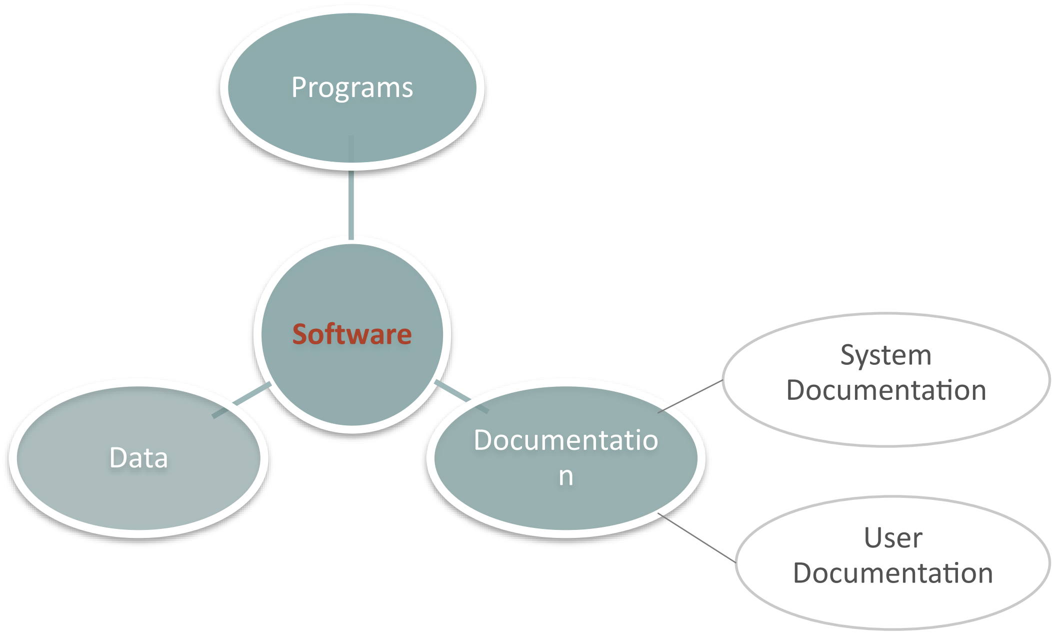
Professional software development

Lots of people write programs. People in business write spreadsheet programs to simplify their jobs, scientists and engineers write programs to process their experimental data, and hobbyists write programs for their own interest and enjoyment. However, the vast majority of software development is a professional activity where software is developed for specific business purposes, for inclusion in other devices, or as software products such as information systems, CAD systems, etc. Professional software, intended for use by someone apart from its developer, is usually developed by teams rather than individuals. It is maintained and changed throughout its life.

Software engineering is intended to support professional software development, rather than individual programming. It includes techniques that support program specification, design, and evolution, none of which are normally relevant for per- sonal software development.

However, when we are talking about software engineering, software is not just the programs themselves but also all associated documentation and configuration data that is required to make these programs operate correctly. A professionally developed software system is often more than a single program. The system usually consists of a number of separate programs and configuration files that are used to set up these programs. It may include system documentation, which describes the picture of the system; user documentation, which explains how to use the system, and web- sites for users to download recent product information.

Many people think that software is simply another word for computer programs. The following figure shows software components:



This is one of the important differences between professional and amateur software development. If you are writing a program for yourself, no one else will use it and you don't have to worry about writing program guides, documenting the program design, etc. However, if you are writing software that other people will use and other engineers will change then you usually have to provide additional information as well as the code of the program.

To help you to get a broad view of what software engineering is about, the following questions summarized some frequently asked questions.

☒ 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?

a) Software specification. *What does the customer need? What are the constraints?*
b) Software development. *Design & programming.*
c) Software validation. *Checking whether it meets requirements.*
d) Software evolution. *Modifications (e.g. customer/market).*

☒ 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 are the costs of software engineering?**

Roughly 60% of software costs are development costs; 40% are testing costs. For custom software, evolution costs often exceed development costs.

☒ **What are the best software engineering techniques and methods?**

While all software projects have to be professionally managed and developed, different techniques are appropriate for different types of system. For example, games should always be developed using a series of prototypes whereas safety critical control systems require a complete and analyzable specification to be developed. You can't, therefore, say that one method is better than another.

☒ 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.

Software engineers are concerned with developing software products (i.e., software which can be sold to a customer). There are two kinds of software products:

1. ***Generic products*** These are stand-alone systems that are produced by a development organization and sold on the open market to any customer who is able to buy them. Examples of this type of product include software for PCs such as databases, word processors, drawing packages, and project-management tools. It also includes, so called vertical applications designed for some specific purpose such as library information systems, accounting systems, or systems for maintaining dental records.
2. ***Customized (or bespoke) products*** These are systems that are commissioned by a particular customer. A software contractor develops the software especially for that customer. Examples of

this type of software include control systems for electronic devices, systems written to support a particular business process, and air traffic control systems.

An important difference between these types of software is that, in generic products, the organization that develops the software controls the software specification. For custom products, the specification is usually developed and controlled by the organization that is buying the software. The software developers must work to that specification.

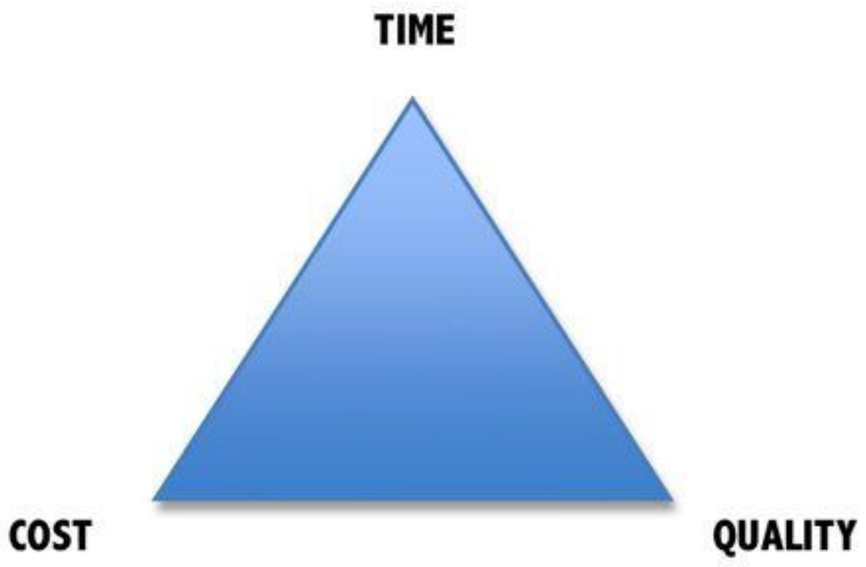
However, the distinction between these system product types is becoming increasingly blurred. More and more system are now being built with a generic product as a base, which is then adapted to suit the requirements of a customer. Enterprise Resource Planning (ERP) systems, such as the SAP system, are the best examples of this approach. Here, a large and complex system is adapted 1or a company by incorporating information about business rules and processes, reports required, and so on.

When we talk about the quality of professional software, we have to take into account that the software is used and changed by people apart from its developers. Quality is therefore not just concerned with what the software does. Rather, it has to include the software's behavior while it is executing and the structure and organization of the system programs and associated documentation. This is reflected in so-called quality or non-functional software attributes. Examples of these attributes are the software's response time to a user query and the under standability of the program code.

The specific set of attributes that you might expect from a software system obviously depends on its application. Therefore, a banking system must be secure, an interactive game must be responsive, a telephone switching system must be reliable, and so on.

The important of Software Engineering:

Complex systems need a disciplined approach for designing, developing and managing them.



Software Development Crises

Projects were: Late - Over budget – Unreliable -Difficult to maintain - Performed poorly.

Software errors….the cost.

Errors in computer software can have devastating effects.

Activity

Try to collect or mention some of the software crises around the world

Therefore…

A well-disciplined approach to software development and management is necessary. This is called engineering.

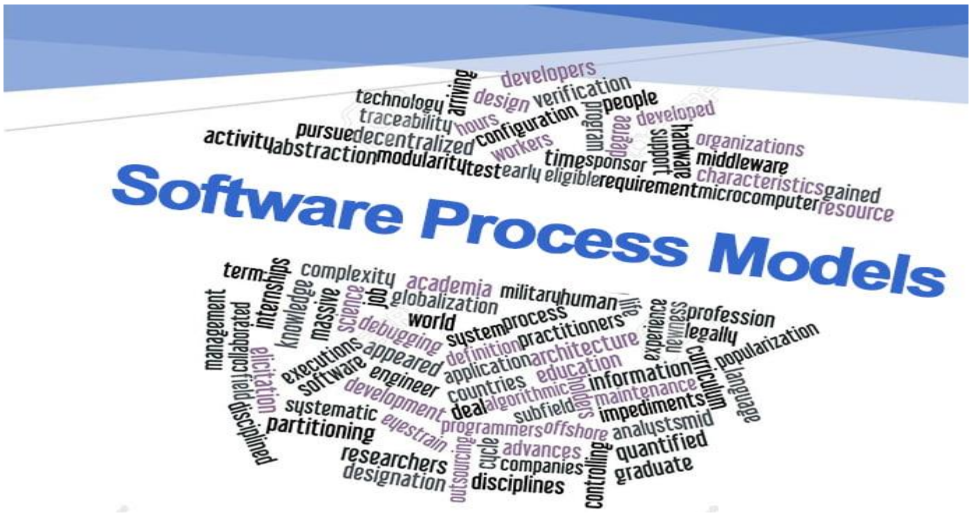
2 Software Processes

Objectives.

understand the idea of a software process- a coherent set of activities for software production.

By the end of this lecture, you will...

- ☒ Know about the fundamental process activities of software requirements, software development, testing and evolution.
- ☒ Know about Components of Software.
- ☒ Understand the concepts of software processes and software processes models.



Software is the set of instructions in the form of programs to govern the computer system and to process the hardware components. To produce a software product the set of activities is used. This set is called a software process.

Software processes in software engineering refer to the methods and techniques used to develop and maintain software.

There are many different software processes but all must include four activities that are fundamental to software engineering.

There four basic key process activities are:

- ***Software Specifications:*** In this process, detailed description of a software system to be developed with its functional and non-functional requirements .
- ***Software Development:*** In this process, designing, programming, documenting, testing, and bug fixing is done .
- ***Software Validation:*** In this process, evaluation software product is done to ensure that the software meets the business requirements as well as the end users needs .
- ***Software Evolution:*** It is a process of developing software initially, then timely updating it for various reasons.

Components of Software: There are three main components of the software:

1. ***Program:*** A computer program is a list of instructions that tell a computer what to do .
2. ***Documentation:*** Source information about the product contained in design documents, detailed code comments….. etc .
3. ***Operating Procedures:*** Set of step-by-step instructions compiled by an organization to help workers carry out complex routine operations.

Other Software Components are:

1. ***Code:*** the instructions that a computer executes in order to perform a specific task or set of tasks.
2. ***Data:*** the information that the software uses or manipulates.
3. ***User interface:*** the means by which the user interacts with the software, such as buttons, menus, and text fields.
4. ***Libraries:*** pre-written code that can be reused by the software to perform common tasks.
5. ***Documentation:*** information that explains how to use and maintain the software, such as user manuals and technical guides.
6. ***Test cases:*** a set of inputs, execution conditions, and expected outputs that are used to test the software for correctness and reliability.
7. ***Configuration files:*** files that contain settings and parameters that are used to configure the software to run in a specific environment.

8. ***Build and deployment scripts:*** scripts or tools that are used to build, package, and deploy the software to different environments.
9. ***Metadata:*** information about the software, such as version numbers, authors, and copyright information.

All these components are important for software development, testing and deployment.

Software Process Model

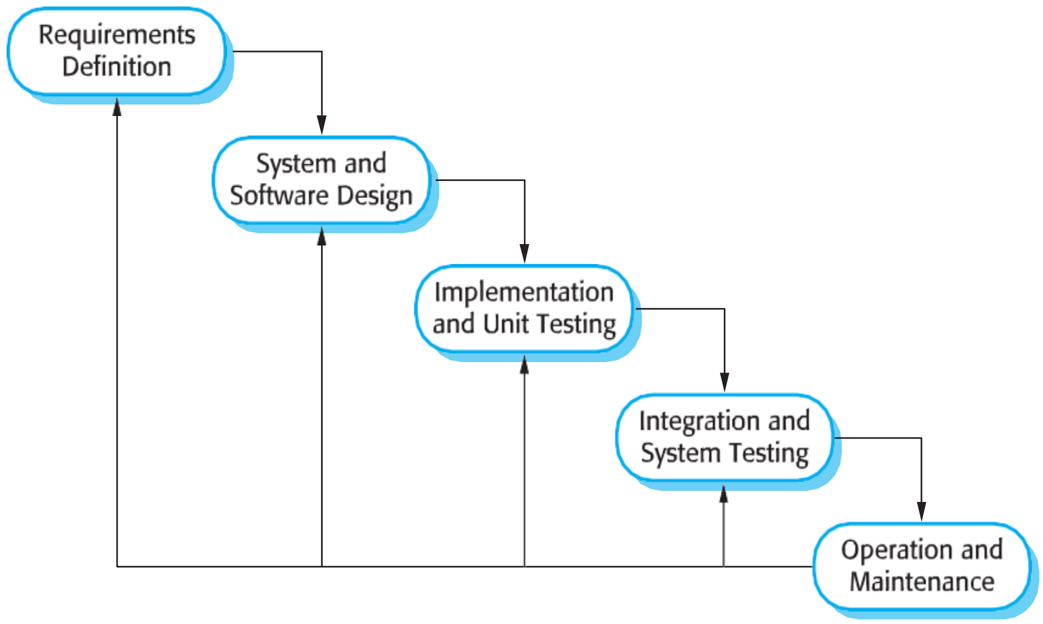
A software process model is an abstraction of the actual process, which is being described. It can also be defined as a simplified representation of a software process. Each model represents a process from a specific perspective.

Need for Process Model:

The software development team must decide the process model that is to be used for software product development and then the entire team must adhere to it. This is necessary because the software product development can then be done systematically. Each team member will understand what is the next activity and how to do it. Thus, process model will bring the definiteness and discipline in overall development process. Every process model consists of definite entry and exit criteria for each phase. Hence the transition of the product through various phases is definite.

The process models that I cover here are:

The Waterfall Model: It is a sequential design process in which progress is seen as flowing steadily downwards.


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

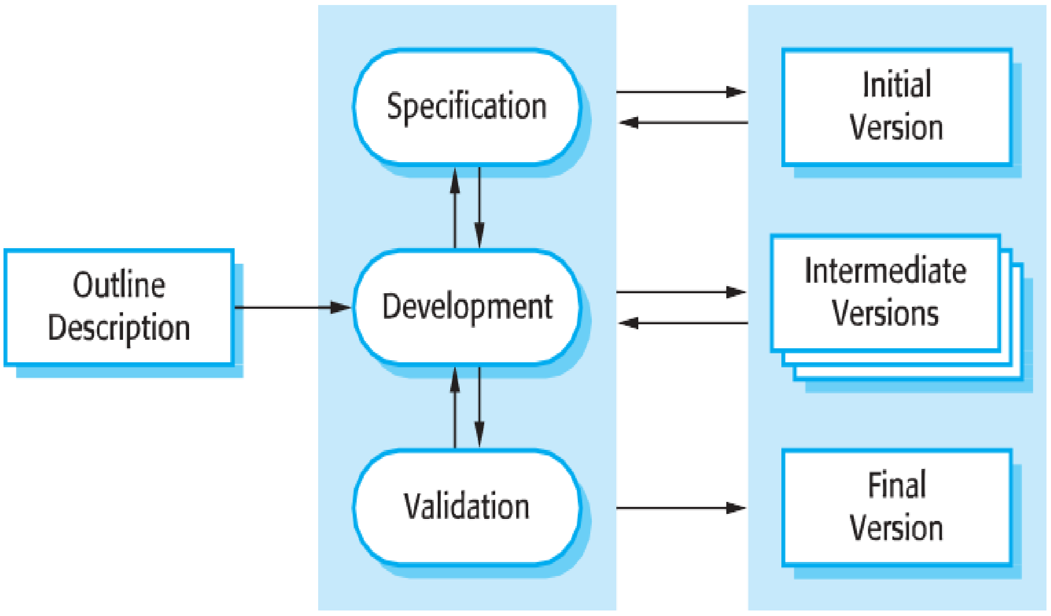
Advantages of waterfall model are:

- Clear and defined phases of development make it easy to plan and manage the project.
- It is well-suited for projects with well-defined and unchanging requirements.

Disadvantages of waterfall model are:

- Changes made to the requirements during the development phase can be costly and time-consuming.
- It can be difficult to know how long each phase will take, making it difficult to estimate the overall time and cost of the project.
- It does not have much room for iteration and feedback throughout the development process.

The Incremental Process Model is also known as the Successive version model. This article focuses on discussing the Incremental Process Model in detail.



The Incremental Model develops the software in smaller, iterative cycles, delivering a working version early and adding features incrementally. This approach offers more flexibility and lower risk.

In Incremental Model Multiple development cycles take place and these cycles are divided into more smaller modules. Generally, a working software in incremental model is produced during first module Each subsequent release of the module adds function to the previous release. In incremental model, process continues till the complete system is achieved.

Characteristics of Incremental Process Model

1. System development is divided into several smaller projects.

2. To create a final complete system, partial systems are constructed one after the other.
3. Priority requirements are addressed first.
4. The requirements for that increment are frozen once they are created.

Advantages of the Incremental Process Model

1. Prepares the software fast.
2. Clients have a clear idea of the project.
3. Changes are easy to implement.
4. Provides risk handling support, because of its iterations.
5. Adjusting the criteria and scope is flexible and less costly.
6. Comparing this model to others, it is less expensive.
7. The identification of errors is simple.

Disadvantages of the Incremental Process Model

1. A good team and proper planned execution are required.
2. Because of its continuous iterations the cost increases.
3. Issues may arise from the system design if all needs are not gathered upfront throughout the program lifecycle.
4. Every iteration step is distinct and does not flow into the next.
5. It takes a lot of time and effort to fix an issue in one unit if it needs to be corrected in all the units.

The incremental model is an understandable alternative to the waterfall model. There are several iterations of smaller cycles comprising requirements, design, programming, and testing, each resulting in a software prototype.

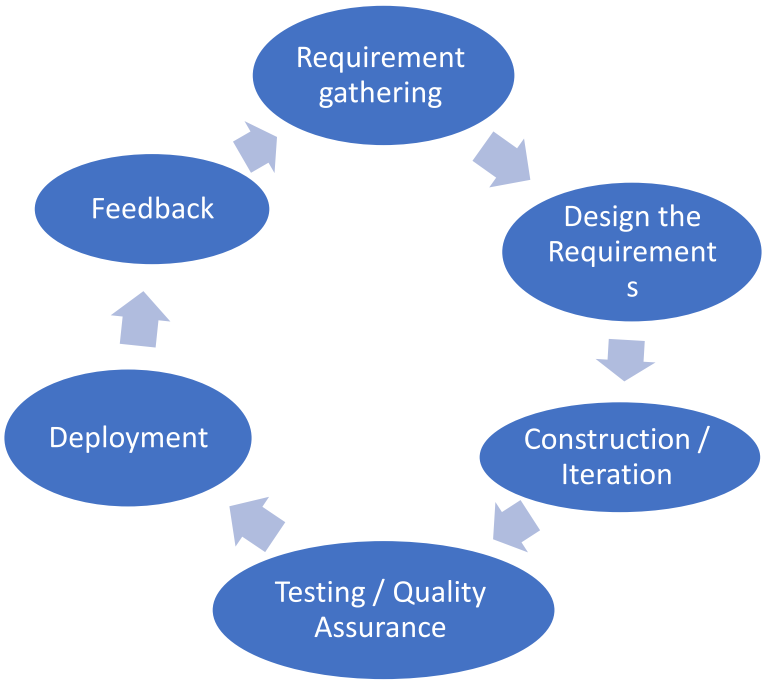
Differences between Waterfall Model and Incremental Model

S. No.	Waterfall Model	Incremental Model
Documentation	Need for Detailed Documentation in the waterfall model is Necessary.	The need for Detailed Documentation in the incremental model is Necessary but not too much.
Planning	In the waterfall model, early stage planning is necessary.	In an incremental model, early-stage planning is also necessary.
Risk	There is a high amount of risk in the waterfall model.	There is a low amount of risk in the incremental model.
Waiting Time for Running Software	There is a long waiting time for running software in the waterfall model.	There is a short waiting time for running software in the incremental model.
Handling Large Projects	The waterfall model can't handle large projects.	The incremental model also can't handle large projects.
Flexibility to Change	Flexibility to change in the waterfall model is Difficult.	Flexibility to change in incremental model is Easy.
Cost	The cost of the Waterfall model is Low.	The cost of the incremental model is also Low.
Testing	Testing is done in the waterfall model after the completion of the coding phase.	Testing is done in the incremental model after every iteration of the phase.
Returning to Previous Phases	Returning to the previous stage/phase in the waterfall model is not possible.	Returning to the previous stage/phase in the incremental model is possible.
Team Size	In the waterfall model, a large team is required.	In an incremental model large team is not required.
Phases Overlapping	In the waterfall model overlapping of phases is not possible.	In incremental model overlapping of phases is possible.
Development Cycles	There is only one cycle in the waterfall model.	Multiple development cycles take place in the incremental model.
Customer Involvement	The customer is involved only at the beginning of development.	In incremental model, customer involvement is intermediate.
Framework Type	The linear framework type is used.	Linear with iterative framework type is used.
Customer Control	The customer is having least control over the administrator.	The customer has more control over the administrator in comparison to the waterfall model.
Reusability	Reusability is the least possible.	Reusability is possible to some extent.

The Agile Model was primarily designed to help a project adapt quickly to change requests. So, the main aim of the Agile model is to facilitate quick project completion. To accomplish this task, agility is required. Agility is achieved by fitting the process to the project and removing activities that may not be essential for a specific project. Also, anything that is a waste of time and effort is avoided. The Agile Model refers to a group of development processes. These processes share some basic characteristics but do have certain subtle differences among themselves.

Steps in the Agile Model

The agile model is a combination of iterative and incremental process models. The steps involve in agile **SDLC** models are :



When To Use the Agile Model?

- ✓ When frequent modifications need to be made, this method is implemented.
- ✓ When a highly qualified and experienced team is available.
- ✓ When a customer is ready to have a meeting with the team all the time.
- ✓ when the project needs to be delivered quickly.
- ✓ Projects with few regulatory requirements or not certain requirements.
- ✓ projects utilizing a less-than-strict current methodology
- ✓ Those undertakings where the product proprietor is easily reachable
- ✓ Flexible project schedules and budgets.

Advantages of the Agile Model

1. Working through Pair programming produces well-written compact programs which have fewer errors as compared to programmers working alone.
2. It reduces the total development time of the whole project.
3. Agile development emphasizes face-to-face communication among team members, leading to better collaboration and understanding of project goals.
4. Customer representatives get the idea of updated software products after each iteration. So, it is easy for him to change any requirement if needed.

5. Agile development puts the customer at the center of the development process, ensuring that the end product meets their needs.

Disadvantages of the Agile Model

1. The lack of formal documents creates confusion and important decisions taken during different phases can be misinterpreted at any time by different team members.
2. It is not suitable for handling complex dependencies.
3. The agile model depends highly on customer interactions so if the customer is not clear, then the development team can be driven in the wrong direction.
4. Agile development models often involve working in short sprints, which can make it difficult to plan and forecast project timelines and deliverables. This can lead to delays in the project and can make it difficult to accurately estimate the costs and resources needed for the project.
5. Agile development models require a high degree of expertise from team members, as they need to be able to adapt to changing requirements and work in an iterative environment. This can be challenging for teams that are not experienced in agile development practices and can lead to delays and difficulties in the project.

6. Due to the absence of proper documentation, when the project completes and the developers are assigned to another project, maintenance of the developed project can become a problem.

Agile development models prioritize flexibility, collaboration, and customer satisfaction. They focus on delivering working software in short iterations, allowing for quick adaptation to changing requirements. While Agile offers advantages like faster delivery and customer involvement, it may face challenges with complex dependencies and lack of formal documentation. Overall, Agile is best suited for projects requiring rapid development, continuous feedback, and a highly skilled team.

Conclusion

Software processes provide structured methods for developing and maintaining software. They include approaches like Waterfall for linear projects, Agile for flexibility, Scrum for teamwork, and DevOps for automation and collaboration. Each has unique strengths, tailored to different project needs. Choosing the right process enhances efficiency and product quality.

questions

What are the four 4 main activities of the software process?

What are the three types of software process models?

What is a software scope?

A well-defined range that includes every action taken to create and distribute the software product is known as the software scope.

Is waterfall iterative or incremental?

A waterfall model is neither iterative nor incremental model.

Which model is incremental?

The incremental model is an understandable alternative to the waterfall model. There are several iterations of smaller cycles comprising requirements, design, programming, and testing, each resulting in a software prototype.

What is the second name of incremental model?

Which of the following is not one of the principles of the agile software development method?

(A) Following the plan

(B) Embrace change

(C) Customer involvement

(D) Incremental delivery

3 Software Requirements

Objectives.

Introduce software requirements and to discuss the processes involved in discovering and documenting these requirements.

By the end of this lecture, you will...

- ☒ understand the concepts of user and system requirements and why these requirements should be written in different ways.
- ☒ understand the differences between functional and nonfunctional software requirements.
- ☒ understand the principal requirements engineering activities of elicitation, analysis and validation, and the relationships between these activities.

The requirements for a system are the descriptions of what the system should do the services that it provides and the constraints on its operation. These requirements reflect the needs of customers for a system that serves a certain purpose such as controlling a device, placing an order, or finding information. The process of finding out, analyzing, documenting and checking these services and constraints is called requirements engineering (RE).

If a company wishes to let a contract for a large software development project, it must define its needs in a sufficiently abstract way that a solution is not predefined. The requirements must be written so that several contractors can bid for the contract, offering, perhaps, different ways of meeting the client organization's needs. Once a contract has been awarded, the contractor must write a system definition for the client in more detail so that the client understands and can validate what the software will do. Both of these documents may be called the requirements document for the system.

Some of the problems that arise during the requirements engineering process are a result of failing to make a clear separation between these different levels of description. I distinguish between them by using the term 'user requirements' to mean the high-level abstract requirements and 'system requirements' to mean the detailed description of what the system should do. User requirements and system requirements may be defined as follows:

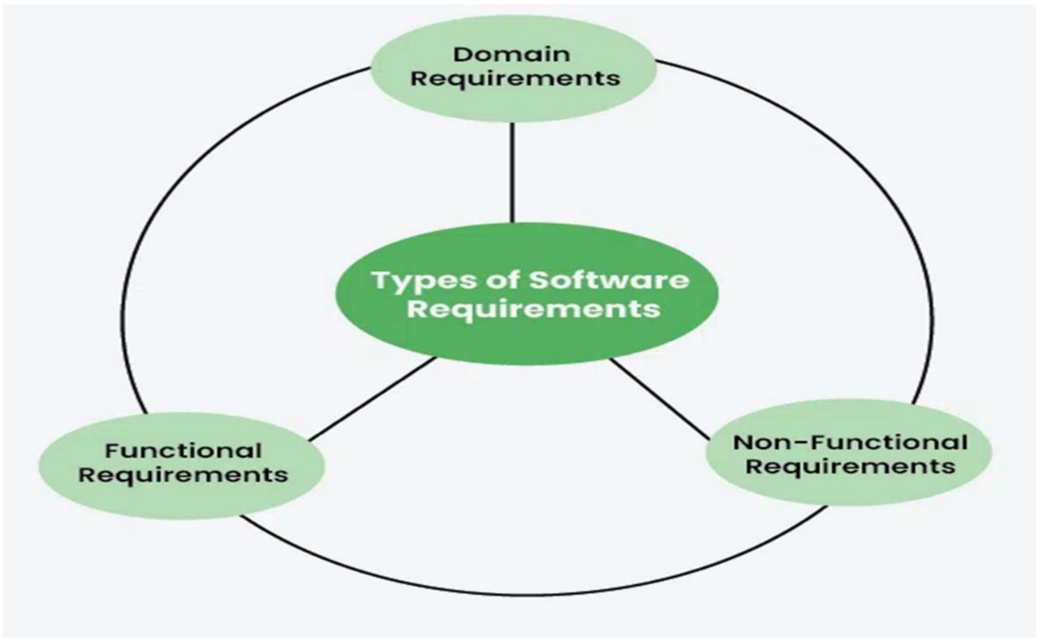
1. **User requirements** are statements, in a natural language plus diagrams, of what services the system is expected to provide to system users and the constraints under which it must operate.
2. **System requirements** are more detailed descriptions of the software system's functions, services, and operational constraints. The system requirements document (sometimes called a functional specification) should define exactly what is to be implemented. It may be part of the contract between the system buyer and the software developers.

What are Software Requirements?

According to IEEE standard 729, a requirement is defined as follows:

1. A condition or capability needed by a user to solve a problem or achieve an objective
2. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed documents
3. A documented representation of a condition or capability, as in 1 and 2.

Classification of Software Requirements is important in the software development process. It organizes our requirements into different categories that make them easier to manage, prioritize, and track. The main types of Software Requirements are functional, non-functional, and domain requirements.



1 .Functional Requirements

Definition: Functional requirements describe what the software should do. They define the functions or features that the system must have.

Examples:

User Authentication: The system must allow users to log in using a username and password.

Search Functionality: The software should enable users to search for products by name or category.

Report Generation: The system should be able to generate sales reports for a specified date range.

Explanation: Functional requirements specify the actions that the software needs to perform. These are the basic features and functionalities that users expect from the software.

2 .Non-functional Requirements

Definition: Non-functional requirements describe how the software performs a task rather than what it should do. They define the quality attributes, performance criteria, and constraints.

Examples:

Performance: The system should process 1,000 transactions per second.

Usability: The software should be easy to use and have a user-friendly interface.

Reliability: The system must have 99.9% uptime.

Security: Data must be encrypted during transmission and storage.

Explanation: Non-functional requirements are about the system's behavior, quality, and constraints. They ensure that the software meets certain standards of performance, usability, reliability, and security.

4. Domain Requirements

Definition: Domain requirements are specific to the domain or industry in which the software operates. They include terminology, rules, and standards relevant to that particular domain.

Examples:

Healthcare: The software must comply with HIPAA regulations for handling patient data.

Finance: The system should adhere to GAAP standards for financial reporting.

E-commerce: The software should support various payment gateways like PayPal, Stripe, and credit cards.

Explanation: Domain requirements reflect the unique needs and constraints of a particular industry. They ensure that the software is relevant and compliant with industry-specific regulations and standards.

What are Functional Requirements?

Functional Requirements are the requirements that the end user specifically demands as basic facilities that the system should offer. It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality that defines what function a system is likely to perform. All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected.

- They are the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements. For example, in a hospital management system, a doctor should be able to retrieve the information of his patients.
- Each high-level functional requirement may involve several interactions or dialogues between the system and the outside world.
- To accurately describe the functional requirements, all scenarios must be enumerated.
- There are many ways of expressing functional requirements e.g., natural language, a structured or formatted language with no rigorous syntax, and formal specification language with proper syntax.
- Functional Requirements in Software Engineering are also called Functional Specification.

What are Non-functional Requirements?

These are basically the quality constraints that the system must satisfy according to the project contract. Nonfunctional requirements, not related to the system functionality, rather define how the system should perform the priority or extent to which these factors are implemented varies from one project to other. They are also called non-behavioral requirements. They basically deal with issues like:

- ✓ Portability
- ✓ Security
- ✓ Maintainability
- ✓ Reliability
- ✓ Scalability
- ✓ Performance
- ✓ Reusability
- ✓ Flexibility

Non-functional requirements are classified into the following types:

- ✓ Interface constraints
- ✓ Performance constraints: response time, security, storage space, etc.
- ✓ Operating constraints
- ✓ Life cycle constraints: maintainability, portability, etc.
- ✓ Economic constraints

The process of specifying non-functional requirements requires the knowledge of the functionality of the system, as well as the knowledge of the context within which the system will operate .

They are divided into two main categories

Execution qualities: These consist of thing like security and usability, which are observable at run time.

Evolution qualities: These consist of things like testability, maintainability, extensibility, and scalability that are embodied in the static structure of the software system.

What are Domain requirements?

Domain requirements are the requirements that are characteristic of a particular category or domain of projects. Domain requirements can be functional or nonfunctional. Domain requirements engineering is a continuous process of proactively defining the requirements for all foreseeable applications to be developed in the software product line. The basic functions that a system of a specific domain must necessarily exhibit come under this category. For instance, in academic software that maintains records of a school or college, the functionality of being able to access the list of faculty and list of students of each grade is a domain requirement. These requirements are therefore identified from that domain model and are not user-specific.

Other Classifications of Software Requirements

Other common classifications of software requirements can be:

1. **User requirements:** These requirements describe what the end-user wants from the software system. User requirements are usually expressed in natural language and are typically gathered through interviews, surveys, or user feedback.
2. **System requirements:** These requirements specify the technical characteristics of the software system, such as its

architecture, hardware requirements, software components, and interfaces. System requirements are typically expressed in technical terms and are often used as a basis for system design.

3. **Business requirements**: These requirements describe the business goals and objectives that the software system is expected to achieve. Business requirements are usually expressed in terms of revenue, market share, customer satisfaction, or other business metrics.
4. **Regulatory requirements:** These requirements specify the legal or regulatory standards that the software system must meet. Regulatory requirements may include data privacy, security, accessibility, or other legal compliance requirements.
5. **Interface requirements**: These requirements specify the interactions between the software system and external systems or components, such as databases, web services, or other software applications.
6. **Design requirements:** These requirements describe the technical design of the software system. They include information about the software architecture, data structures, algorithms, and other technical aspects of the software.

By classifying software requirements, it becomes easier to manage, prioritize, and document them effectively. It also helps ensure that all important aspects of the system are considered during the development process.

Advantages of Classifying Software Requirements

1. **Better organization:** Classifying software requirements helps organize them into groups that are easier to manage, prioritize, and track throughout the development process.
2. **Improved communication:** Clear classification of requirements makes it easier to communicate them to stakeholders, developers, and other team members. It also ensures that everyone is on the same page about what is required.
3. **Increased quality:** By classifying requirements, potential conflicts or gaps can be identified early in the development process. This reduces the risk of errors, omissions, or misunderstandings, leading to higher-quality software.
4. **Improved traceability:** Classifying requirements helps establish traceability, which is essential for demonstrating compliance with regulatory or quality standards.

Disadvantages of classifying software requirements

1. **Complexity**: Classifying software requirements can be complex, especially if there are many stakeholders with different needs or requirements. It can also be time-consuming to identify and classify all the requirements.

2. **Rigid structure**: A rigid classification structure may limit the ability to accommodate changes or evolving needs during the development process. It can also lead to a siloed approach that prevents the integration of new ideas or insights.
3. **Misclassification**: Misclassifying requirements can lead to errors or misunderstandings that can be costly to correct later in the development process.

Overall, the advantages of classifying software requirements outweigh the disadvantages, as it helps ensure that the software system meets the needs of all stakeholders and is delivered on time, within budget, and with the required quality.

Conclusion

Classifying software requirements provides numerous benefits, such as better organization, improved communication, increased quality, and enhanced traceability. By systematically categorizing the requirements development teams can be sure that the software meets the requirements of stakeholders while observing standards and delivered with efficiency and effectiveness.

Questions

What are the 3 types of Software Requirements?

Who defines software requirements?

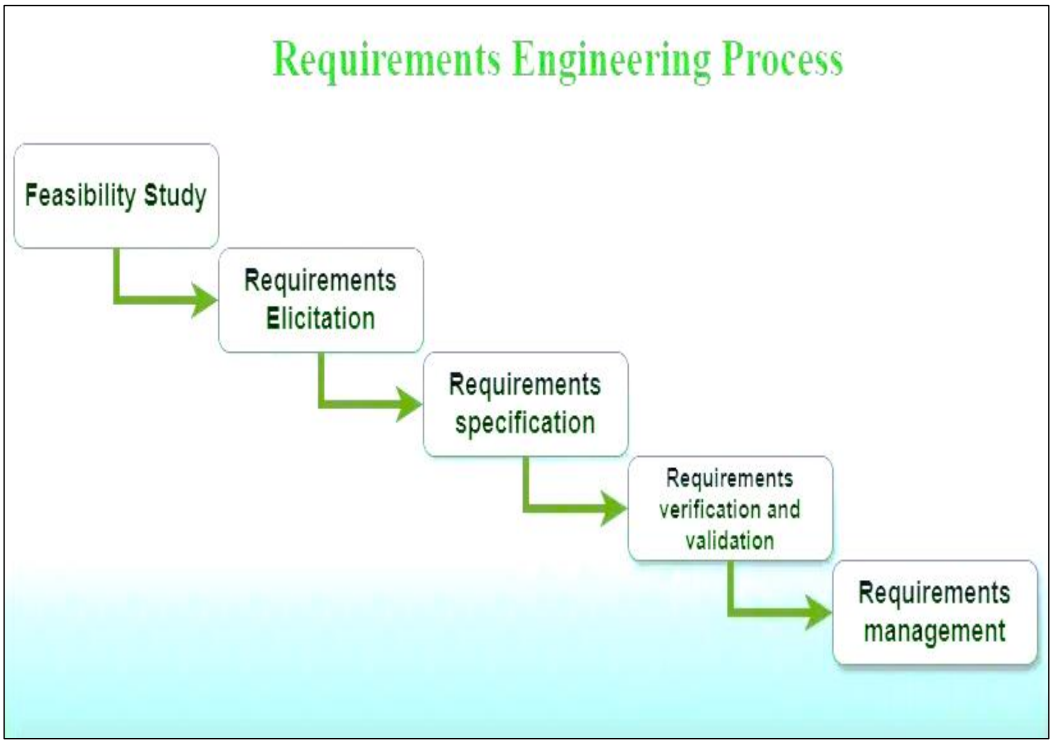
Software requirements are defined by stakeholders, including users, clients, developers, and business analysts.

Requirements Engineering Process in Software Engineering

Requirements Engineering is the process of identifying, eliciting, analyzing, specifying, validating, and managing the needs and expectations of stakeholders for a software system.

Requirements Engineering Process

1. **Feasibility Study**
2. **Requirements elicitation**
3. **Requirements specification**
4. **Requirements for verification and validation**
5. **Requirements management**


Requirements Engineering Process
Feasibility Study
Requirements Elicitation
Requirements specification
Requirements verification and validation
Requirements management

1. Feasibility Study

The feasibility study mainly concentrates on below five mentioned areas below. Among these Economic Feasibility Study is the most important part of the feasibility analysis and the Legal Feasibility Study is less considered feasibility analysis.

1. ***Technical Feasibility***: In Technical Feasibility current resources both hardware software along required technology are analyzed/assessed to develop the project. This technical feasibility study reports whether there are correct required resources and technologies that will be used for project development. Along with this, the feasibility study also analyzes the technical skills and capabilities of the technical team, whether existing technology can be used or not, whether maintenance and up-gradation are easy or not for the chosen technology, etc.

2. ***Operational Feasibility:*** In Operational Feasibility degree of providing service to requirements is analyzed along with how easy the product will be to operate and maintain after deployment. Along with these other operational scopes are determining the usability of the product, Determining suggested solution by the software development team is acceptable or not, etc.

3. ***Economic Feasibility:*** In the Economic Feasibility study cost and benefit of the project are analyzed. This means under this feasibility study a detailed analysis is carried out will be cost of the project for development which includes all required costs for final development hardware and software resources required, design and development costs operational costs, and so on. After that, it is analyzed whether the project will be beneficial in terms of finance for the organization or not.
4. ***Legal Feasibility:*** In legal feasibility, the project is ensured to comply with all relevant laws, regulations, and standards. It identifies any legal constraints that could impact the project and reviews existing contracts and agreements to assess their effect on the project's execution. Additionally, legal feasibility considers issues related to intellectual property, such as patents and copyrights, to safeguard the project's innovation and originality.
5. ***Schedule Feasibility:*** In schedule feasibility, the project timeline is evaluated to determine if it is realistic and achievable. Significant milestones are identified, and deadlines are established to track progress effectively. Resource availability is assessed to ensure that the necessary resources are accessible to meet the project schedule. Furthermore, any time constraints that might affect project delivery are considered to

ensure timely completion. This focus on schedule feasibility is crucial for the successful planning and execution of a project.

2. Requirements Elicitation

It is related to the various ways used to gain knowledge about the project domain and requirements. The various sources of domain knowledge include customers, business manuals, the existing software of the same type, standards, and other stakeholders of the project. The techniques used for requirements elicitation include interviews, brainstorming, task analysis, Delphi technique, prototyping, etc. Elicitation does not produce formal models of the requirements understood. Instead, it widens the domain knowledge of the analyst and thus helps in providing input to the next stage.

Requirements elicitation is the process of gathering information about the needs and expectations of stakeholders for a software system. This is the first step in the requirements engineering process and it is critical to the success of the software development project. The goal of this step is to understand the problem that the software system is intended to solve and the needs and expectations of the stakeholders who will use the system.

Several techniques can be used to elicit requirements, including:

- ***Interviews***: These are one-on-one conversations with stakeholders to gather information about their needs and expectations.

- ***Surveys:*** These are questionnaires that are distributed to stakeholders to gather information about their needs and expectations.
- ***Focus Groups:*** These are small groups of stakeholders who are brought together to discuss their needs and expectations for the software system.
- ***Observation:*** This technique involves observing the stakeholders in their work environment to gather information about their needs and expectations.
- ***Prototyping:*** This technique involves creating a working model of the software system, which can be used to gather feedback from stakeholders and to validate requirements.

It's important to document, organize, and prioritize the requirements obtained from all these techniques to ensure that they are complete, consistent, and accurate.

3. Requirements Specification

This activity is used to produce formal software requirement models. All the requirements including the functional as well as the non-functional requirements and the constraints are specified by these models in totality. During specification, more knowledge about the problem may be required which can again trigger the elicitation process. The models used at this stage include ER diagrams, data flow diagrams (DFDs), function decomposition diagrams (FDDs), data dictionaries, etc.

Requirements specification is the process of documenting the requirements identified in the analysis step in a clear, consistent, and unambiguous manner. This step also involves prioritizing and grouping the requirements into manageable chunks.

The goal of this step is to create a clear and comprehensive document that describes the requirements for the software system. This document should be understandable by both the development team and the stakeholders.

Several types of requirements are commonly specified in this step, including

1. **Functional Requirements:** These describe what the software system should do. They specify the functionality that the system must provide, such as input validation, data storage, and user interface.
2. **Non-Functional Requirements**: These describe how well the software system should do it. They specify the quality attributes of the system, such as performance, reliability, usability, and security.
3. **Constraints:** These describe any limitations or restrictions that must be considered when developing the software system.

4. **Acceptance Criteria**: These describe the conditions that must be met for the software system to be considered complete and ready for release.

To make the requirements specification clear, the requirements should be written in a natural language and use simple terms, avoiding technical jargon, and using a consistent format throughout the document. It is also important to use diagrams, models, and other visual aids to help communicate the requirements effectively.

Once the requirements are specified, they must be reviewed and validated by the stakeholders and development team to ensure that they are complete, consistent, and accurate.

4. Requirements Verification and Validation

Verification**:** It refers to the set of tasks that ensures that the software correctly implements a specific function.

Validation: It refers to a different set of tasks that ensures that the software that has been built is traceable to customer requirements. If requirements are not validated, errors in the requirement definitions would propagate to the successive stages resulting in a lot of modification and rework. The main steps for this process include:

1. The requirements should be consistent with all the other requirements i.e. no two requirements should conflict with each other.
2. The requirements should be complete in every sense.

3. The requirements should be practically achievable.

Reviews, buddy checks, making test cases, etc. are some of the methods used for this.

Requirements verification and validation (V&V) is the process of checking that the requirements for a software system are complete, consistent, and accurate and that they meet the needs and expectations of the stakeholders. The goal of V&V is to ensure that the software system being developed meets the requirements and that it is developed on time, within budget, and to the required quality.

1. *Verification* is checking that the requirements are complete, consistent, and accurate. It involves reviewing the requirements to ensure that they are clear, testable, and free of errors and inconsistencies. This can include reviewing the requirements document, models, and diagrams, and holding meetings and walkthroughs with stakeholders.
2. *Validation* is the process of checking that the requirements meet the needs and expectations of the stakeholders. It involves testing the requirements to ensure that they are valid and that the software system being developed will meet the needs of the stakeholders. This can include testing the software system through simulation, testing with prototypes, and testing with the final version of the software.

3. *Verification and Validation* is an iterative process that occurs throughout the software development life cycle. It is important to involve stakeholders and the development team in the V&V process to ensure that the requirements are thoroughly reviewed and tested.

It’s important to note that V&V is not a one-time process, but it should be integrated and continue throughout the software development process and even in the maintenance stage.

5. Requirements Management

Requirement management is the process of analyzing, documenting, tracking, prioritizing, and agreeing on the requirement and controlling the communication with relevant stakeholders. This stage takes care of the changing nature of requirements. It should be ensured that the SRS is as modifiable as possible to incorporate changes in requirements specified by the end users at later stages too. Modifying the software as per requirements in a systematic and controlled manner is an extremely important part of the requirements engineering process.

Requirements management is the process of managing the requirements throughout the software development life cycle, including tracking and controlling changes, and ensuring that the requirements are still valid and relevant. The goal of requirements management is to ensure that the software system being developed meets the needs and expectations of the stakeholders and that it is developed on time, within budget, and to the required quality.

Several key activities are involved in requirements management, including:

1. **Tracking and controlling changes:** This involves monitoring and controlling changes to the requirements throughout the development process, including identifying the source of the change, assessing the impact of the change, and approving or rejecting the change.
2. **Version control**: This involves keeping track of different versions of the requirements document and other related artifacts.
3. **Traceability**: This involves linking the requirements to other elements of the development process, such as design, testing, and validation.
4. **Communication:** This involves ensuring that the requirements are communicated effectively to all stakeholders and that any changes or issues are addressed promptly.
5. **Monitoring and reporting**: This involves monitoring the progress of the development process and reporting on the status of the requirements.

Requirements management is a critical step in the software development life cycle as it helps to ensure that the software system being developed meets the needs and expectations of stakeholders and that it is developed on time, within budget, and to the required quality. It also helps to prevent scope creep and to ensure that the requirements are aligned with the project goals.

Tools Involved in Requirement Engineering

- Observation report
- Questionnaire (survey, poll)
- Use cases
- User stories
- Requirement workshop
- Mind mapping
- Roleplaying
- Prototyping

Conclusion

As the project develops and new information becomes available, the iterative requirements engineering process may involve going back and reviewing earlier phases. Throughout the process, stakeholders in the project must effectively communicate and collaborate to guarantee that the software system satisfies user needs and is in line with the company’s overall goals.

Questions

1. What is requirements engineering?

Requirements engineering is the process of identifying, analyzing, documenting, and managing the needs and expectations of stakeholders for a software system.

2. Why is requirements engineering important?

It ensures that the software meets the needs of its users, is delivered on time, within budget, and to the required quality standards.

3. What are the main steps in the requirements engineering process?

The main steps are feasibility study, requirements elicitation, requirements specification, requirements verification and validation, and requirements management.

4. How does requirements validation differ from requirements verification?

Verification checks that the requirements are correctly specified and error-free, while validation ensures that the requirements meet the needs and expectations of the stakeholders.

5. What techniques are used to gather requirements?

Techniques include interviews, surveys, focus groups, observation, and prototyping, which help collect detailed information about stakeholder needs and expectations.

4 System modeling

Objectives.

Introduce some types of system model that may be developed as part of the requirements engineering and system design processes.

By the end of this lecture, you will...

- ☒ Understand how graphical models can be used to represent software systems.
- ☒ introduced to some of the diagram types in the Unified Modeling Language (UML) and how these diagrams may be used in system modeling.
- ☒ understand why different types of models are required and the fundamental system modeling perspectives of context, interaction, structure, and behavior.

What is UML?

Unified Modeling Language (**UML**) is a standardized visual modeling language that is a versatile, flexible, and user-friendly method for visualizing a system's design. Software system artifacts can be specified, visualized, built, and documented with the use of UML.

- We use UML diagrams to show the behavior and structure of a system.
- UML helps software engineers, businessmen, and system architects with modeling, design, and analysis.

Why do we need UML?

We need UML (Unified Modeling Language) to visually represent and communicate complex system designs, facilitating better understanding and collaboration among stakeholders. Below is why we need UML:

- Complex applications need collaboration and planning from multiple teams and hence require a clear and concise way to communicate amongst them.
- Businessmen do not understand code. So, UML becomes essential to communicate with non-programmers about essential requirements, functionalities, and processes of the system.

- A lot of time is saved down the line when teams can visualize processes, user interactions, and the static structure of the system.

Types of UML Diagrams

UML is linked with object-oriented design and analysis. UML makes use of elements and forms associations between them to form diagrams. Diagrams in UML can be broadly classified as (*It is up to the student to write a brief research paper*).

When to Use UML Diagrams

- When a system's general structure needs to be represented, UML diagrams can help make it clearer how various parts work together, which facilitates idea sharing between stakeholders.
- When collecting and recording system requirements, UML diagrams, such as use case diagrams, can help you clearly grasp user demands by showing how users will interact with the system.
- If you're involved in database design, class diagrams are great for illustrating the relationships among various data entities, ensuring your data model is well-organized.
- When working with team members or clients, UML diagrams act as a shared language that connects technical and non-technical stakeholders, improving overall understanding and alignment.

Common Challenges in UML Modeling

Below are the common challenges in UML Modeling:

- Accurately representing complex system requirements can be difficult, leading to either oversimplification or overwhelming detail.
- Team members may interpret the model differently, resulting in inconsistencies and misunderstandings about its purpose.
- Keeping UML diagrams current as the system evolves can be time-consuming, risking outdated representations if not managed effectively.
- Agile promotes teamwork, but sometimes UML diagrams are complicated and only a few people understand them. It can be hard to make sure everyone can contribute to and use the diagrams effectively.

Benefits of Using UML Diagrams

Below are the benefits of using UML Diagrams:

- Developers and stakeholders may communicate using a single visual language thanks to UML's standardized approach to system model representation.
- Developers, designers, testers, and business users are just a few of the stakeholders with whom UML diagrams may effectively communicate.

- UML diagrams make it easier to see the linkages, processes, and parts of a system.
- One useful tool for documentation is a UML diagram. They offer an ordered and systematic method for recording a system's behavior, architecture, and design, among other elements.

Finally, *Unified Modeling Language (UML)* is like a blueprint for software developers. It helps them plan and design complex systems by creating diagrams that show how different parts of the system will work together. We'll look at the advantages and disadvantages of UML. Understanding these can help developers use UML effectively and avoid its pitfalls, making their projects more successful.

Advantages of Unified Modeling Language (UML)

Below are the advantages of UML:

- **Standardization:**
    - UML provides a standardized way of representing system models, ensuring that developers and stakeholders can communicate using a common visual language.
- **Communication:**
    - UML diagrams serve as a powerful communication tool between stakeholders, including developers, designers, testers, and business users. They help in conveying complex ideas more understandably.

- **Visualization:**
  - UML diagrams facilitate the visualization of system components, relationships, and processes. This visual representation aids in understanding and designing complex systems.
- **Documentation:**
  - UML diagrams can be used as effective documentation tools. They provide a structured and organized way to document various aspects of a system, such as architecture, design, and behavior.
- **Analysis and Design:**
  - UML supports both the analysis and design phases of software development. It helps in modeling the requirements of a system and then transforming them into a design that can be implemented.

Disadvantages of Unified Modeling Language (UML)

Below are the disadvantages of UML:

- **Complexity:**
  - UML can be complex, especially for beginners. Learning all the aspects of UML and becoming proficient in using it may require a significant investment of time and effort.

- **Overhead:**
  - In some cases, creating and maintaining detailed UML diagrams can be time-consuming. For small and simple projects, the overhead of creating extensive UML documentation may not be justified.
- **Ambiguity:**
  - Interpretation of UML diagrams can be subjective, leading to potential ambiguity. Different individuals may interpret the same diagram in slightly different ways, causing confusion.
- **Learning Curve:**
  - Due to its extensive features and diagrams, there is a steep learning curve associated with UML. Teams may need training and experience to use it effectively.
- **Over-Modeling or Under-Modeling:**
  - There is a risk of over-modeling (creating too many unnecessary details) or under-modeling (omitting important details) in UML diagrams. Striking the right balance is crucial for their effectiveness.

System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system. It is about representing a system using some kind of graphical notation, which is now almost always based on notations in the Unified Modeling Language (UML). Models help the analyst to understand the functionality of the system; they are used to communicate with customers.

Models can explain the system from **different perspectives**:

- An **external** perspective, where you model the context or environment of the system.
- An **interaction** perspective, where you model the interactions between a system and its environment, or between the components of a system.
- A **structural** perspective, where you model the organization of a system or the structure of the data that is processed by the system.
- A **behavioral** perspective, where you model the dynamic behavior of the system and how it responds to events.

Five types of UML diagrams that are the most useful for system modeling:

- **Activity** diagrams, which show the activities involved in a process or in data processing.
- **Use case** diagrams, which show the interactions between a system and its environment.
- **Sequence** diagrams, which show interactions between actors and the system and between system components.

- **Class** diagrams, which show the object classes in the system and the associations between these classes.
- **State** diagrams, which show how the system reacts to internal and external events.

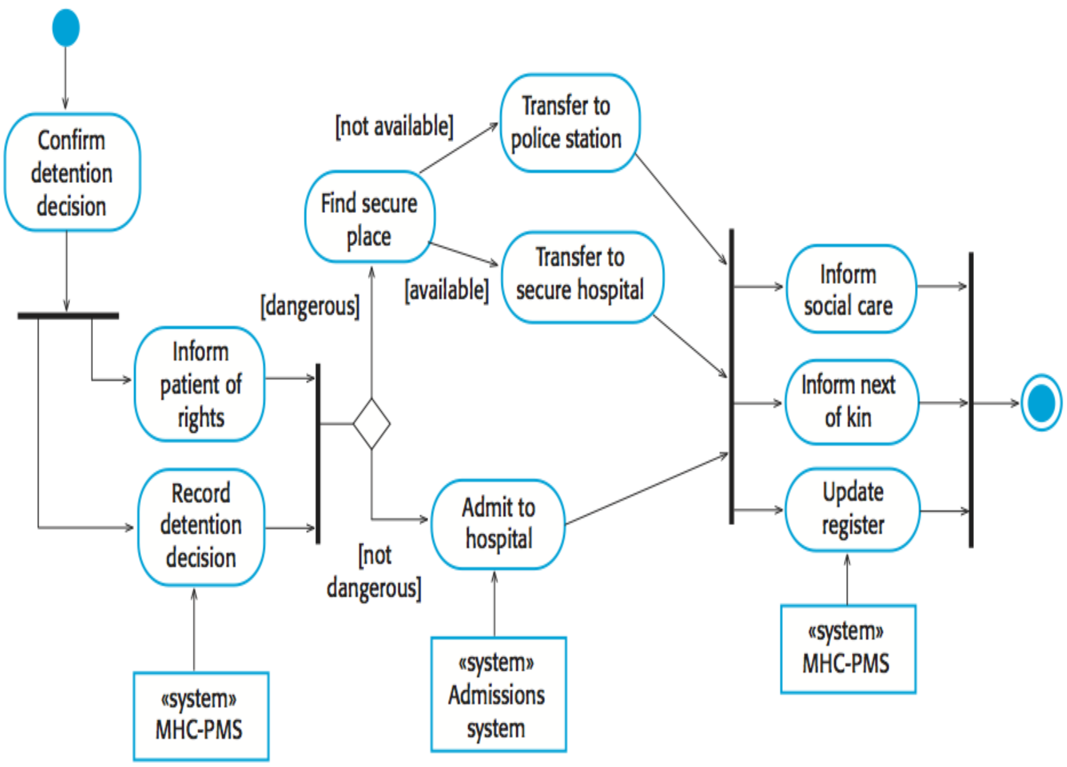
Models of both new and existing system are used during **requirements engineering**. Models of the **existing systems** help clarify what the existing system does and can be used as a basis for discussing its strengths and weaknesses. These then lead to requirements for the new system. Models of the **new system** are used during requirements engineering to help explain the proposed requirements to other system stakeholders. Engineers use these models to discuss design proposals and to document the system for implementation.

Context and process models

- **Context models** are used to illustrate the operational context of a system - they show what lies outside the system boundaries. Social and organizational concerns may affect the decision on where to position system boundaries. Architectural models show the system and its relationship with other systems.

- **System boundaries** are established to define what is inside and what is outside the system. They show other systems that are used or depend on the system being developed. The position of the system boundary has a profound effect on the system requirements. Defining a system boundary is a political judgment since there may be pressures to develop system boundaries that increase/decrease the influence or workload of different parts of an organization.

- Context models simply show the other systems in the environment, not how the system being developed is used in that environment. **Process models** reveal how the system being developed is used in broader business processes. UML activity diagrams may be used to define business process models.
- The example below shows a UML **activity diagram** describing the process of involuntary detention and the role of MHC-PMS (mental healthcare patient management system) in it.



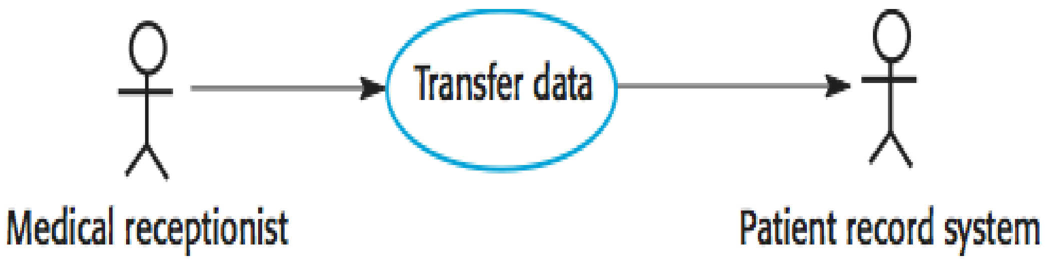
▪ Interaction models

Types of interactions that can be represented in a model:

- Modeling **user interaction** is important as it helps to identify user requirements.
- Modeling **system-to-system interaction** highlights the communication problems that may arise.
- Modeling **component interaction** helps us understand if a proposed system structure is likely to deliver the required system performance and dependability.

Use cases were developed originally to support requirements elicitation and now incorporated into the UML. Each use case represents a discrete task that involves external interaction with a system. Actors in a use case may be people or other systems. Use cases can be represented using a UML use case diagram and in a more detailed textual/tabular format.

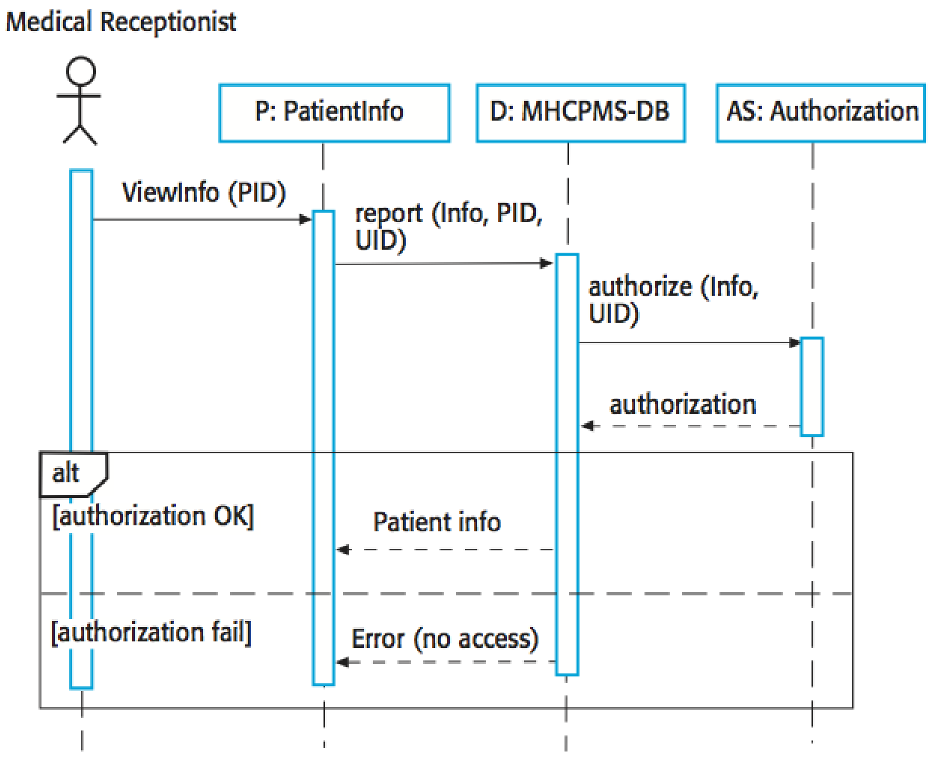
Simple use case diagram:



Use case description in a tabular format:

Use case title	**Transfer data**
Description	A receptionist may transfer data from the MHC-PMS to a general patient record database that is maintained by a health authority. The information transferred may either be updated personal information (address, phone number, etc.) or a summary of the patient's diagnosis and treatment.
Actor(s)	Medical receptionist, patient records system (PRS)
Preconditions	Patient data has been collected (personal information, treatment summary); The receptionist must have appropriate security permissions to access the patient information and the PRS.
Postconditions	PRS has been updated
Main success scenario	1. Receptionist selects the "Transfer data" option from the menu. 2. PRS verifies the security credentials of the receptionist. 3. Data is transferred. 4. PRS has been updated.
Extensions	2a. The receptionist does not have the necessary security credentials. 2a.1. An error message is displayed. 2a.2. The receptionist backs out of the use case.

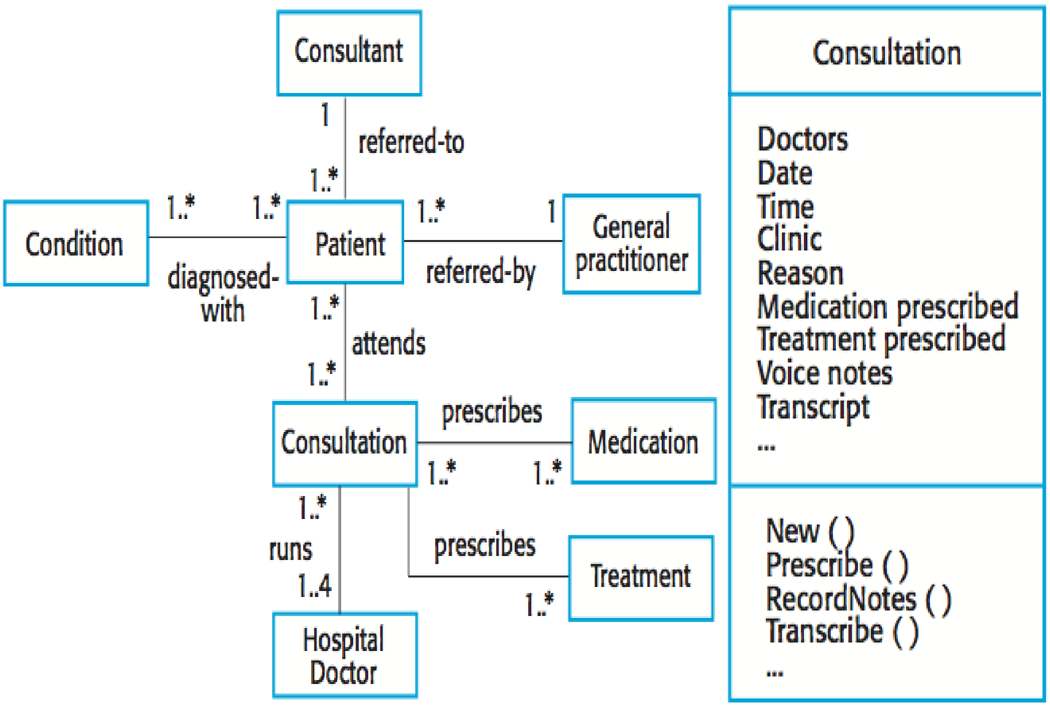
UML **sequence diagrams** are used to model the interactions between the actors and the objects within a system. A sequence diagram shows the sequence of interactions that take place during a particular use case or use case instance. The objects and actors involved are listed along the top of the diagram, with a dotted line drawn vertically from these. Interactions between objects are indicated by annotated arrows.



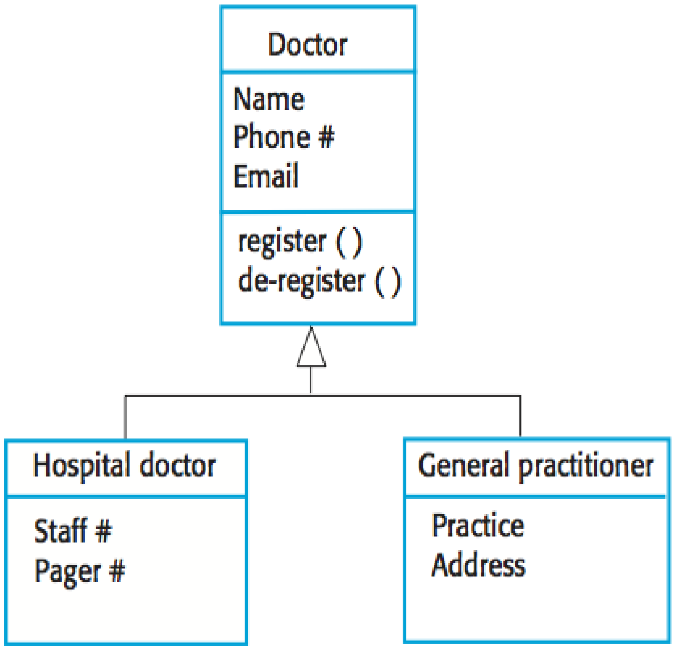
▪ Structural models

- **Structural models** of software display the organization of a system in terms of the components that make up that system and their relationships. Structural models may be **static** models, which show the structure of the system design, or **dynamic** models, which show the organization of the system when it is executing. You create structural models of a system when you are discussing and designing the system architecture.

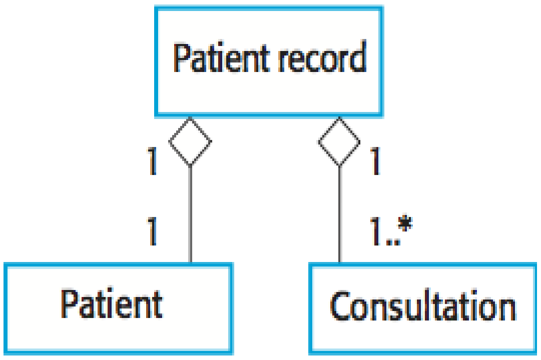
- UML **class diagrams** are used when developing an object-oriented system model to show the classes in a system and the associations between these classes. An object class can be thought of as a general definition of one kind of system object. An association is a link between classes that indicates that there is some relationship between these classes. When you are developing models during the early stages of the software engineering process, objects represent something in the real world, such as a patient, a prescription, doctor, etc.



Generalization is an everyday technique that we use to manage complexity. In modeling systems, it is often useful to examine the classes in a system to see if there is scope for generalization. In object-oriented languages, such as Java, generalization is implemented using the class **inheritance** mechanisms built into the language. In a generalization, the attributes and operations associated with higherlevel classes are also associated with the lower-level classes. The lower-level classes are subclasses inherit the attributes and operations from their super classes. These lower-level classes then add more specific attributes and operations.



An **aggregation** model shows how classes that are collections are composed of other classes. Aggregation models are similar to the part-of relationship in semantic data models.

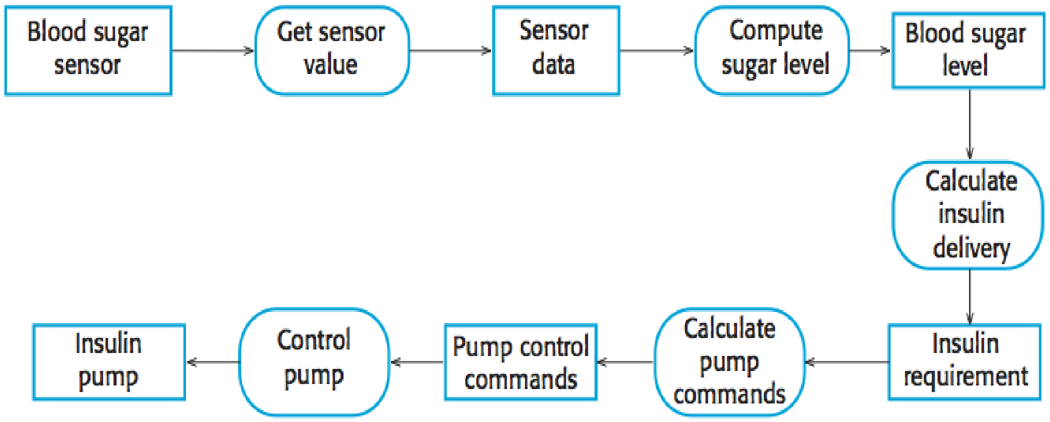


▪ Behavioral models

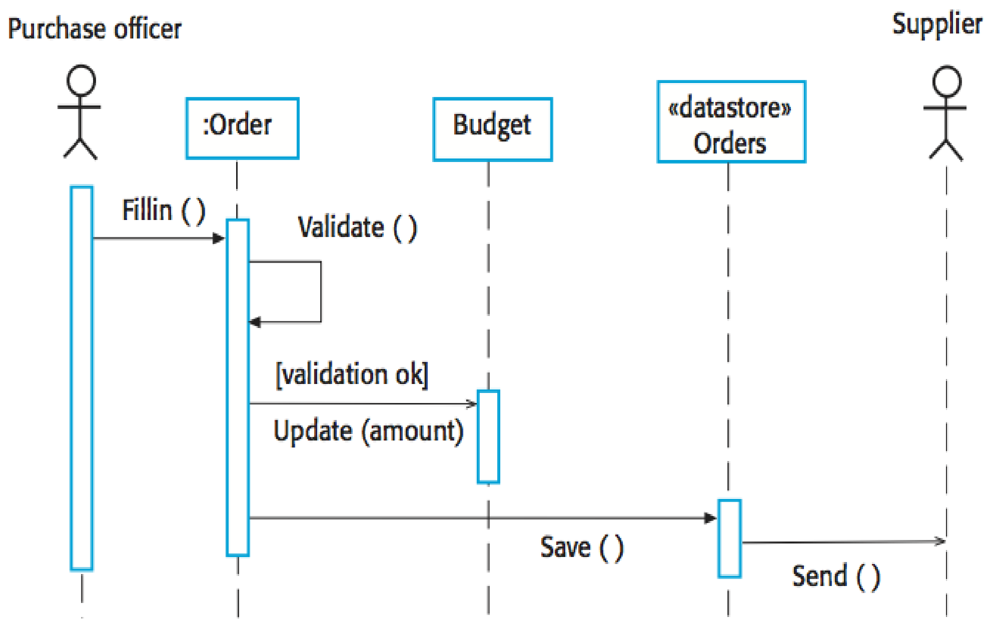
Behavioral models are models of the dynamic behavior of a system as it is executing. They show what happens or what is supposed to happen when a system responds to a stimulus from its environment. Two types of stimuli:

- Some **data** arrives that has to be processed by the system.
- Some **event** happens that triggers system processing. Events may have associated data, although this is not always the case.

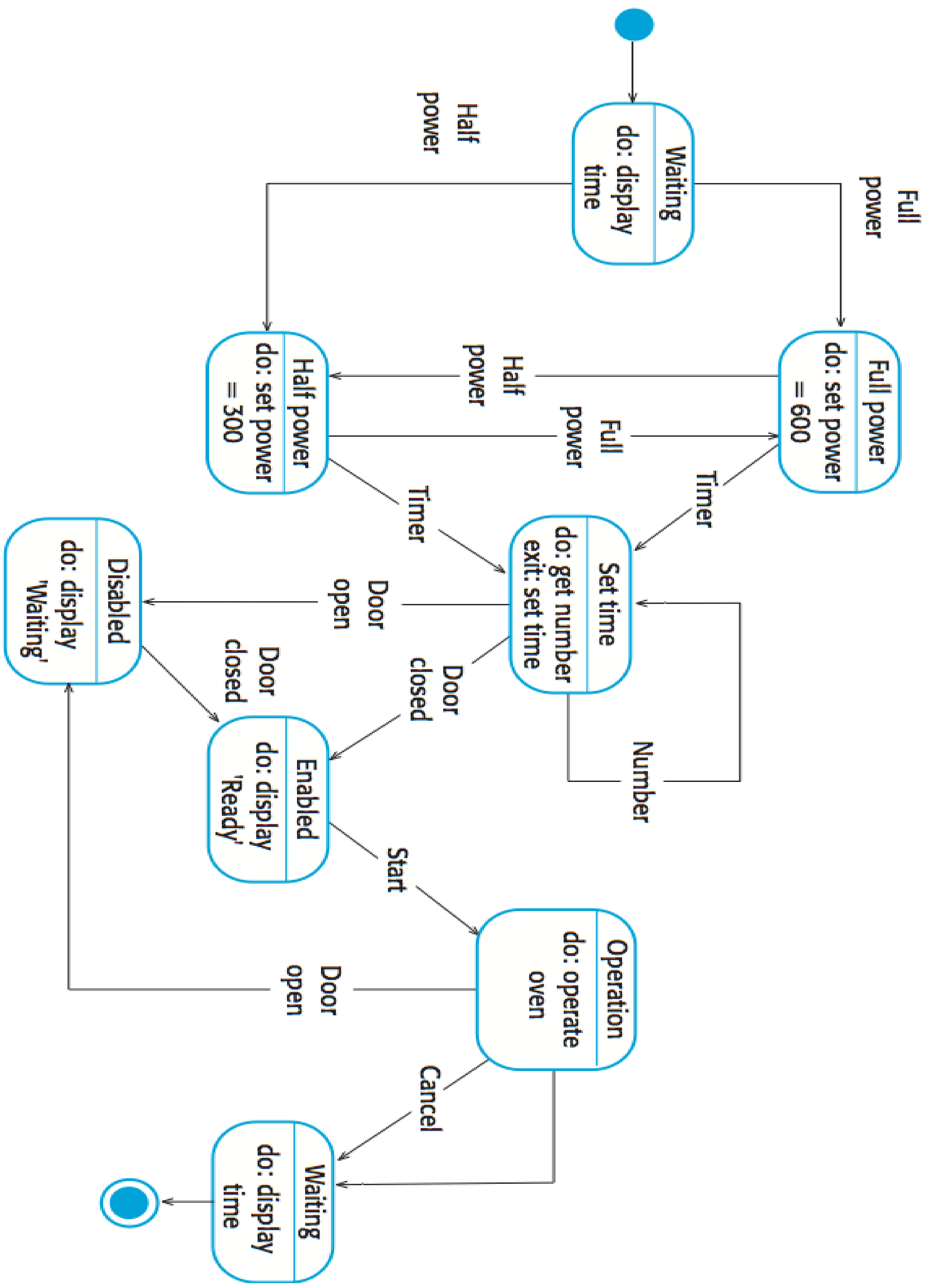
Many business systems are data-processing systems that are primarily driven by data. They are controlled by the data input to the system, with relatively little external event processing. **Data-driven models** show the sequence of actions involved in processing input data and generating an associated output. They are particularly useful during the analysis of requirements as they can be used to show end-to-end processing in a system. Data-driven models can be created using UML **activity diagrams**:



Data-driven models can also be created using UML **sequence diagrams**:



Real-time systems are often event-driven, with minimal data processing. For example, a landline phone switching system responds to events such as 'receiver off hook' by generating a dial tone. **Event-driven models** show how a system responds to external and internal events. It is based on the assumption that a system has a finite number of states and that events (stimuli) may cause a transition from one state to another. Event-driven models can be created using UML **state diagrams**:


Waiting
do: display time
Full power
Half power
Full power
do: set power = 600
Half power
do: set power = 300
Half power
Full power
Timer
Timer
Set time
do: get number
exit: set time
Number
Door open
Door closed
Disabled
do: display 'Waiting'
Door closed
Enabled
do: display 'Ready'
Start
Operation
do: operate oven
Door open
Cancel
Waiting
do: display time

Conclusion

- A model is an abstract view of a system that ignores some system details. Complementary system models can be developed to show the system's context, interactions, structure, and behavior.
- Context models show how a system that is being modeled is positioned in an environment with other systems and processes. They help define the boundaries of the system to be developed.
- Use case diagrams and sequence diagrams are used to describe the interactions between user the system being designed and users/other systems. Use cases describe interactions between a system and external actors; sequence diagrams add more information to these by showing interactions between system objects.
- Structural models show the organization and architecture of a system. Class diagrams are used to define the static structure of classes in a system and their associations.
- Behavioral models are used to describe the dynamic behavior of an executing system. This can be modeled from the perspective of the data processed by the system or by the events that stimulate responses from a system.
- Activity diagrams may be used to model the processing of data, where each activity represents one process step.
- State diagrams are used to model a system's behavior in response to internal or external events.

Questions & Answers

1. The Unified Modeling Language (UML) has become an effective standard for software modelling. How many different notations does it have?

a) Three

b) Four

c) Six

d) Nine

Explanation: The different notations of UML include the nine UML diagrams namely class, object, sequence, collaboration, activity, state-chart, component, deployment and use case diagrams.

2. Which model in system modelling depicts the dynamic behavior of the system?

a) Context Model

b) Behavioral Model

c) Data Model

d) Object Model

Explanation: Behavioral models are used to describe the dynamic behavior of an executing system. This can be modeled from the perspective of the data processed by the system or by the events that stimulate responses from a system.

3. Which model in system modelling depicts the static nature of the system?

a) Behavioral Model

b) Context Model

c) Data Model

d) Structural Model

Explanation: Structural models show the organization and architecture of a system. These are used to define the static structure of classes in a system and their associations

4. Which perspective in system modelling shows the system or data architecture.

a) Structural perspective

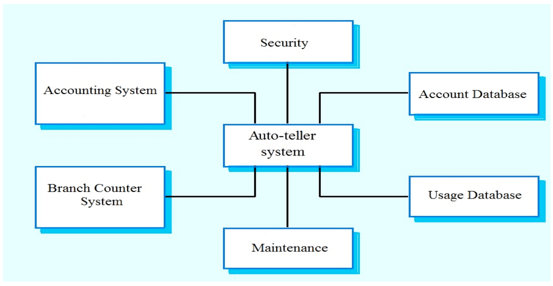
b) Behavioral perspective

c) External perspective

d) All of the mentioned

Explanation: Structural perspective is used to define the static structure of classes in a system and their associations.

5. Which system model is being depicted by the ATM operations shown below:



a) Structural model

b) Context model

c) Behavioral model

d) Interaction model

Explanation: Context models are used to illustrate the operational context of a system. They show what lies outside the system boundaries.

6. The UML supports event-based modeling using _____________ diagrams.

a) Deployment

b) Collaboration

c) State chart

d) All of the mentioned

Explanation: State diagrams show system states and events that cause transitions from one state to another.

7. Activity diagrams are used to model the processing of data.

a) True

b) False

Explanation: The statement mentioned is true and each activity represents one process step.

5 Architectural Design

Objectives.

Introduce the concepts of software architecture and architectural design.

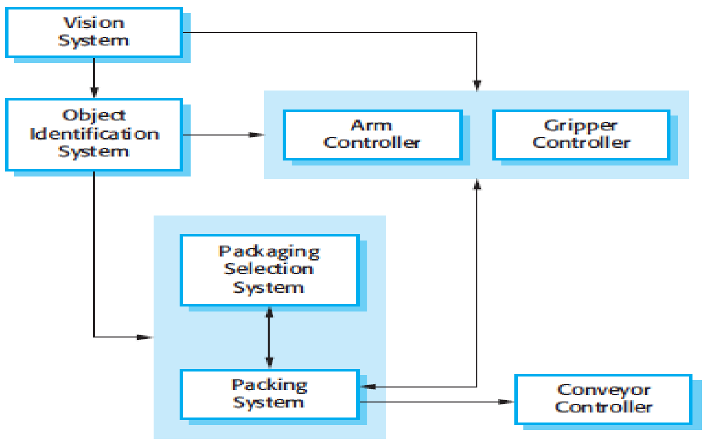
By the end of this lecture, you will...

- ☒ Understand why the architectural design of software is important.
- ☒ Introduced to the idea of architectural patterns, well-tried ways of organizing system architectures, which can be reused in system designs.

The software needs an architectural design to represent the design of the software. IEEE defines architectural design as "the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system." The software that is built for computer-based systems can exhibit one of these many architectural styles.

Architectural design is concerned with understanding how a system should be organized and designing the overall structure of that system. Architectural design is the first stage in the software design process. It is the critical link between design and requirements engineering, as it identifies the main structural components in a system and the relationships between them. The output of the architectural design process is an architectural model that describes how the system is organized as a set of communicating components.

To help you *understand what I mean by system architecture*, consider the following Figure.



This shows an abstract model of the architecture for a packing robot system that shows the components that have to be developed. This robotic system can pack different kinds of object. It uses a vision component to pick out objects on a conveyor, identify the type of object, and select the right kind of packaging. The system then moves objects from the delivery conveyor to be packaged. It places packaged objects on another conveyor. The architectural model shows these components and the links between them.

In practice, there is a significant overlap between the processes of requirements engineering and architectural design. Ideally, a system specification should not include any design information. This is unrealistic except for very small systems. *Architectural decomposition* is usually necessary to structure and organize the specification. Therefore, as part of the requirements engineering process, you might propose an abstract system architecture where you associate groups of system functions or features with large-scale components or sub-systems. *You can then use this decomposition to discuss the requirements and features of the system with stakeholders*.

Software architecture is important because it affects the performance, robustness, distributability and maintainability of a system (Bosch, 2000). As Bosch discusses, individual components implement the functional system requirements. The nonfunctional requirements depend on the system architecture—the way in which these components are organized and communicate. In many systems, non-functional requirements are also influenced by individual components, but there is no doubt that the architecture of the system is the dominant influence.

Bass et al. (2003) *discuss three advantages of explicitly designing and documenting software architecture*:

1. *Stakeholder communication* the architecture is a high-level presentation of the system that may be used as a focus for discussion by a range of different stakeholders.
2. *System analysis* Making the system architecture explicit at an early stage in the system development requires some analysis. Architectural design decisions have a profound effect on whether or not the system can meet critical requirements such as performance, reliability, and maintainability.
3. *Large-scale reuse* A model of a system architecture is a compact, manageable description of how a system is organized and how the components interoperate. The system architecture is often the same for systems with similar requirements and so can support large-scale software reuse.

System architectures are often modeled using simple block diagrams, as in the above figure. Each box in the diagram represents a component. Boxes within boxes indicate that the component has been decomposed to sub-components. Arrows mean that data and or control signals are passed from component to component in the direction of the arrows.

System Category Consists of

- A set of components (eg: a database, computational modules) that will perform a function required by the system.
- The set of connectors will help in coordination, communication, and cooperation between the components.
- Conditions that define how components can be integrated to form the system.
- Semantic models that help the designer to understand the overall properties of the system.

The use of architectural styles is to establish a structure for all the components of the system.

Taxonomy of Architectural Styles

1] Data centered architectures:

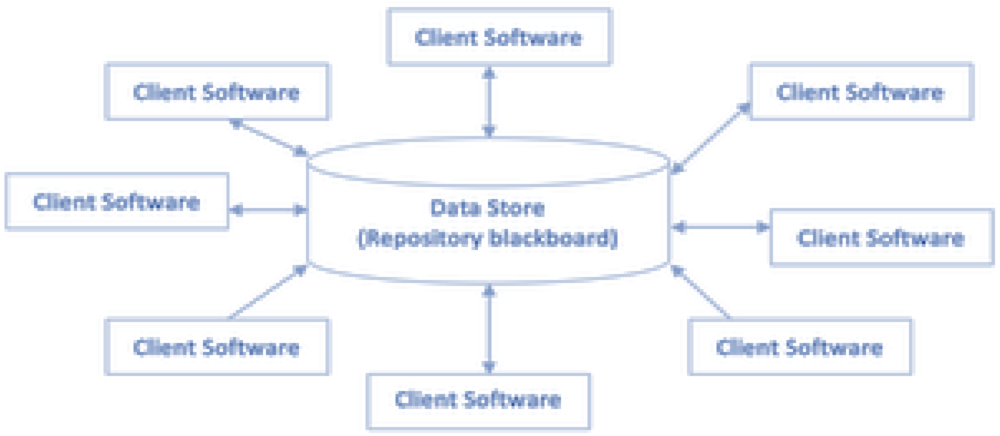
- A data store will reside at the center of this architecture and is accessed frequently by the other components that update, add, delete, or modify the data present within the store.
- The figure illustrates a typical data-centered style. The client software accesses a central repository. Variations of this approach are used to transform the repository into a blackboard when data related to the client or data of interest for the client change the notifications to client software.
- This data-centered architecture will promote integrability. This means that the existing components can be changed and new

client components can be added to the architecture without the permission or concern of other clients.

- Data can be passed among clients using the blackboard mechanism.

Advantages of Data centered architecture:

- Repository of data is independent of clients
- Client works independent of each other
- It may be simple to add additional clients.
- Modification can be very easy



2] Data flow architectures:

- This kind of architecture is used when input data is transformed into output data through a series of computational manipulative components.
- The figure represents pipe-and-filter architecture since it uses both pipe and filter and it has a set of components called filters connected by lines.

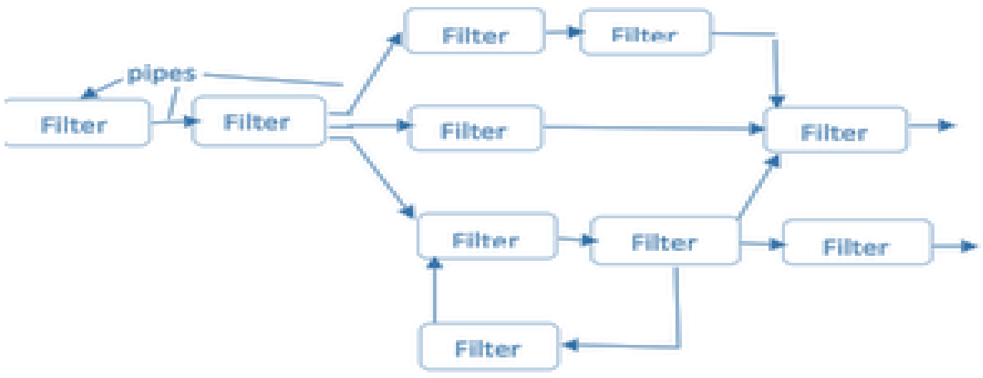
- Pipes are used to transmitting data from one component to the next.
- Each filter will work independently and is designed to take data input of a certain form and produces data output to the next filter of a specified form. The filters don't require any knowledge of the working of neighboring filters.
- If the data flow degenerates into a single line of transforms, then it is termed as batch sequential. This structure accepts the batch of data and then applies a series of sequential components to transform it.

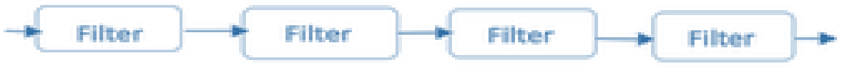
Advantages of Data Flow architecture:

- It encourages upkeep, repurposing, and modification.
- With this design, concurrent execution is supported.

Disadvantage of Data Flow architecture:

- It frequently degenerates to batch sequential system
- Data flow architecture does not allow applications that require greater user engagement.
- It is not easy to coordinate two different but related streams

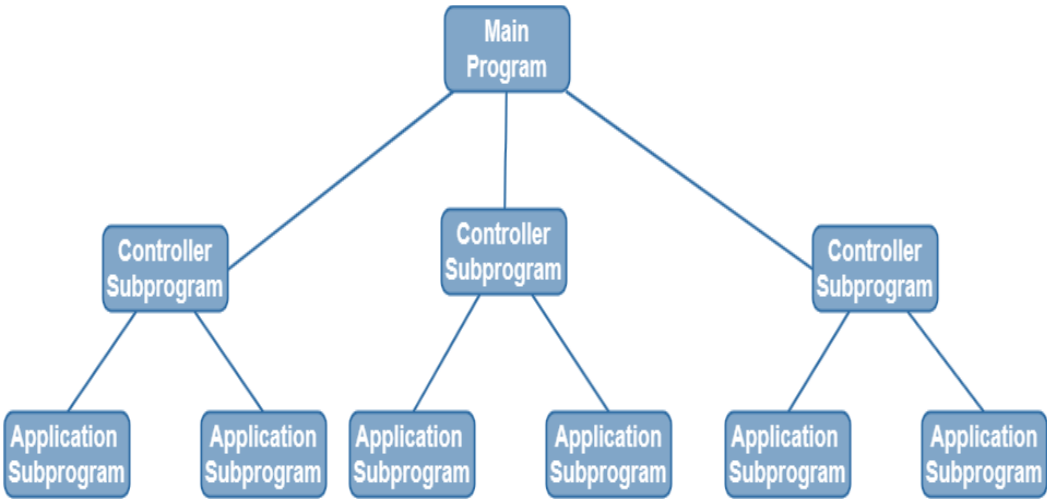




3] Call and Return architectures

It is used to create a program that is easy to scale and modify. Many sub-styles exist within this category. Two of them are explained below.

- **Remote procedure call architecture:** This component is used to present in a main program or sub program architecture distributed among multiple computers on a network.
- **Main program or Subprogram architectures:** The main program structure decomposes into number of subprograms or function into a control hierarchy. Main program contains number of subprograms that can invoke other components.



4] Object Oriented architecture

The components of a system encapsulate data and the operations that must be applied to manipulate the data. The coordination and communication between the components are established via the message passing.

Characteristics of Object-Oriented architecture:

- Object protect the system's integrity.
- An object is unaware of the depiction of other items.

Advantage of Object-Oriented architecture:

- It enables the designer to separate a challenge into a collection of autonomous objects.
- Other objects are aware of the implementation details of the object, allowing changes to be made without having an impact on other objects.

5] Layered architecture

- A number of different layers are defined with each layer performing a well-defined set of operations. Each layer will do some operations that becomes closer to machine instruction set progressively.
- At the outer layer, components will receive the user interface operations and at the inner layers, components will perform the operating system interfacing (communication and coordination with OS)
- Intermediate layers to utility services and application software functions.

Conclusion

- A software architecture is a description of how a software system is organized. Properties of a system such as performance, security, and availability are influenced by the architecture used.
- Architectural design decisions include decisions on the type of application, the distribution of the system, the architectural styles to be used, and the ways in which the architecture should be documented and evaluated.
- Architectures may be documented from several different perspectives or views. Possible views include a conceptual view, a logical view, a process view, a development view, and a physical view.
- Architectural patterns are a means of reusing knowledge about generic system architectures. They describe the architecture, explain when it may be used, and discuss its advantages and disadvantages.

6 Design and implementation

Objectives.

introduce object-oriented software design using the UML and highlight important implementation concerns.

By the end of this lecture, you will...

- ☒ understand the most important activities in a general, object-oriented design process.
- ☒ understand some of the different models that may be used to document an object-oriented design.
- ☒ know about the idea of design patterns and how these are a way of reusing design knowledge and experience.

Software design and implementation is the stage in the software engineering process at which an executable software system is developed. *For some simple systems*, software design and implementation is software engineering, and all other activities are merged with this process. However, *for large systems*, software design and implementation is only one of a set of processes (requirements engineering, verification and validation, etc.) involved in software engineering.

Software design and implementation activities are invariably interleaved. Software design is a creative activity in which you identify software components and their relationships, based on a customer's requirements. Implementation is the process of realizing the design as a program. *Sometimes*, there is a separate design stage and this design is modeled and documented. *At other times*, a design is in the programmer's head or roughly sketched on a whiteboard or sheets of paper. *Design is about how to solve a problem*, so there is always a design process. However, it isn't always necessary or appropriate to describe the design in detail using the UML or other design description language.

Design and implementation are closely linked and you should normally take implementation issues into account when developing a design. For example, using the UML to document a design may be the right thing to do if you are programming in an object-oriented language such as Java or C#. It is less useful, I think, if you are developing in a dynamically typed language like Python and makes no sense at all if you are implementing your system by configuring an off-the-shelf package. Agile methods usually work from informal sketches of the design and leave many design decisions to programmers.

Therefore, I don’t cover programming topics here. Instead, this lecture has two aims:

1. To show how system modeling and architectural design are put into practice in developing an object-oriented software design.
2. To introduce important implementation issues that are not usually covered in programming books. These include software reuse, configuration management, and open-source development.

Object-oriented design using the UML.

An object-oriented system is made up of interacting objects that maintain their own local state and provide operations on that state. The representation of the state is private and cannot be accessed directly from outside the object. Object-oriented design processes involve designing object classes and the relationships between these classes. These classes define the objects in the system and their interactions. When the design is realized as an executing program, the objects are created dynamically from these class definitions.

Object-oriented systems are easier to change than systems developed using functional approaches. Objects include both data and operations to manipulate that data. They may therefore be understood and modified as stand-alone entities. Changing the implementation of an object or adding services should not affect other system objects. Because objects are associated with things, there is often a clear mapping between real world entities (such as hardware components) and their controlling objects in the system. *This improves* the understandability, and hence the maintainability, of the design.

To develop a system design from concept to detailed, object-oriented design, *there are several things that you need to do*:

1. Understand and define the context and the external interactions with the system.
2. Design the system architecture.
3. Identify the principal objects in the system.
4. Develop design models.
5. Specify interfaces.

Like all creative activities, design is not a clear-cut, sequential process. You develop a design by getting ideas, proposing solutions, and refining these solutions as information becomes available. You inevitably have to backtrack and retry when problems arise. Sometimes you explore options in detail to see if they work; at other times you ignore details until late in the process. Consequently, I have deliberately not illustrated this process as a simple diagram because that would imply design can be thought of as a neat sequence of activities. In fact, all of the above activities are interleaved and so influence each other.

System context and interactions.

The first stage in any software design process is to develop an understanding of the relationships between the software that is being designed and its external environment. This is essential for deciding how to provide the required system functionality and how to structure the system to communicate with its environment. Understanding of the context also lets you establish the boundaries of the system.

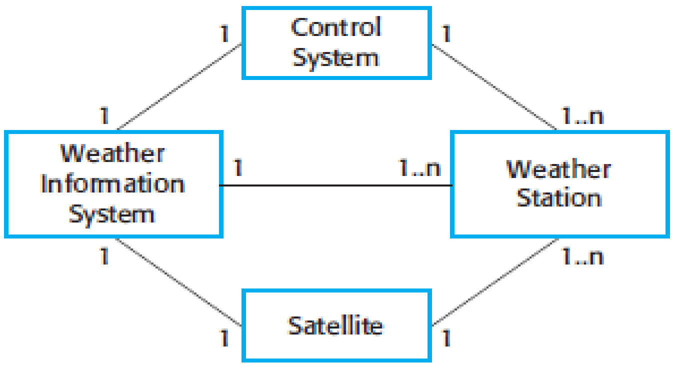
Setting the system boundaries helps you decide what features are implemented in the system being designed and what features are in other associated systems. In this case, you need to decide how functionality is distributed between the control system for all of the weather stations, and the embedded software in the weather station itself.

System context models and interaction models present complementary views of the relationships between a system and its environment:

1. A system context model is a structural model that demonstrates the other systems in the environment of the system being developed.

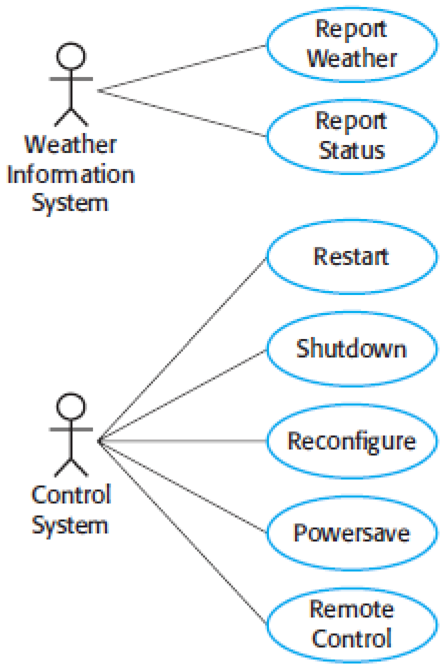
2. An interaction model is a dynamic model that shows how the system interacts with its environment as it is used.

The context model of a system may be represented using associations. Associations simply show that there are some relationships between the entities involved in the association. The nature of the relationships is now specified. You may therefore document the environment of the system using a simple block diagram, showing the entities in the system and their associations. This is illustrated in the following Figure (System context for the weather station).



When you model the interactions of a system with its environment you should use an abstract approach that does not include too much detail. *One way to do this is to use a use case model*. Each use case represents an interaction with the system. Each possible interaction is named in an ellipse and the external entity involved in the interaction is represented by a stick figure.

The use case model for the weather station is shown in the following Figure. This shows that the weather station interacts with the weather information system to report weather data and the status of the weather station hardware. Other interactions are with a control system that can issue specific weather station control commands. A stick figure is used in the UML to represent other systems as well as human users.



Design models:

Design or system models, show the objects or object classes in a system. They also show the associations and relationships between these entities. These models are the bridge between the system requirements and the implementation of a system. They have to be abstract so that unnecessary detail doesn't hide the relationships between them and the system requirements. However, they also have to include enough detail for programmers to make implementation decisions.

Generally, you get around this type of conflict by developing models at different levels of detail. Where there are close links between requirements engineers, designers, and programmers, then abstract models may be all that are required. Specific design decisions may be made as the system is implemented, with problems resolved through informal discussions. When the links between system specifiers, designers, and programmers are indirect (e.g., where a system is being designed in one part of an organization but implemented elsewhere), then more detailed models are likely to be needed.

An important step in the design process, therefore, is to decide on the design models that you need and the level of detail required in these models. This depends on the type of system that is being developed. You design a sequential data-processing system in a different way from an embedded real-time system, so you will need different design models. The UML supports 13 different types of models but, you rarely use all of these. Minimizing the number of models that are produced reduces the costs of the design and the time required to complete the design process.

When you use the UML to develop a design, you will normally develop two kinds of design model:

1. *Structural models,* which describe the static structure of the system using object classes and their relationships. Important relationships that may be documented at this stage are generalization (inheritance) relationships, uses/used-by relationships, and composition relationships.
2. *Dynamic models*, which describe the dynamic structure of the system and show the interactions between the system objects. Interactions that may be documented include the sequence of service requests made by objects and the state changes that are triggered by these object interactions.

In the early stages of the design process, I think there are three models that are particularly useful for adding detail to use case and architectural models:

1. *Subsystem models*, which that show logical groupings of objects into coherent subsystems. These are represented using a form of class diagram with each subsystem shown as a package with enclosed objects. Subsystem models are static (structural) models.
2. *Sequence models*, which show the sequence of object interactions. These are represented using a UML sequence or a collaboration diagram. Sequence models are dynamic models.

State machine model, which show how individual objects change their state in response to events. These are represented in the UML using state diagrams. State machine models are dynamic models.

A subsystem model is a useful static model as it shows how a design is organized into logically related groups of objects. As well as subsystem models, you may also design detailed object models, showing all of the objects in the systems and their associations (inheritance, generalization, aggregation, etc.). However, there is a danger in doing too much modeling. You should not make detailed decisions about the implementation that really should be left to the system programmers.

Sequence models are dynamic models that describe, for each mode of interaction, the sequence of object interactions that take place. When documenting a design, you should produce a sequence model for each significant interaction. If you have developed a use case model then there should be a sequence model for each use case that you have identified.

State diagrams are useful high-level models of a system or an object's operation. You don't usually need a state diagram for all of the objects in the system. Many of the objects in a system are relatively simple and a state model adds unnecessary detail to the design.

For further reading, refer to SOFTWARE ENGINEERING-Ninth Edition-Ian Sommerville chapter 7.

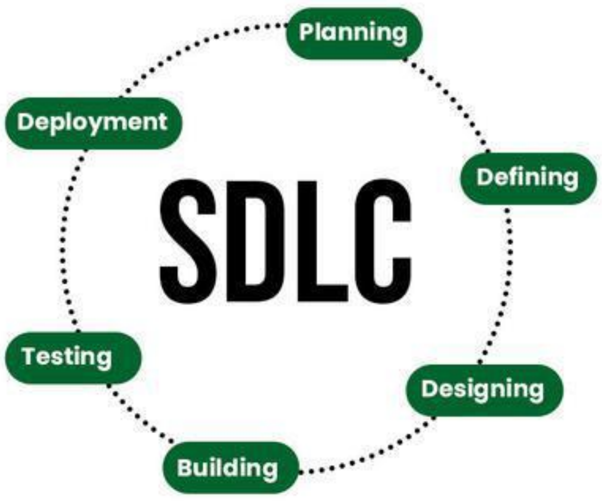
Conclusion

- Software design and implementation are interleaved activities. The level of detail in the design depends on the type of system being developed and whether you are using a plan-driven or agile approach.
- The process of object-oriented design includes activities to design the system architecture, identify objects in the system, describe the design using different object models, and document the component interfaces.
- A range of different models may be produced during an object-oriented design process. These include static models (class models, generalization models, association models) and dynamic models (sequence models, state machine models).
- Component interfaces must be defined precisely so that other objects can use them. A UML interface stereotype may be used to define interfaces.
- When developing software, you should always consider the possibility of reusing existing software, either as components, services, or complete systems.

7 Software Development Life Cycle (SDLC)

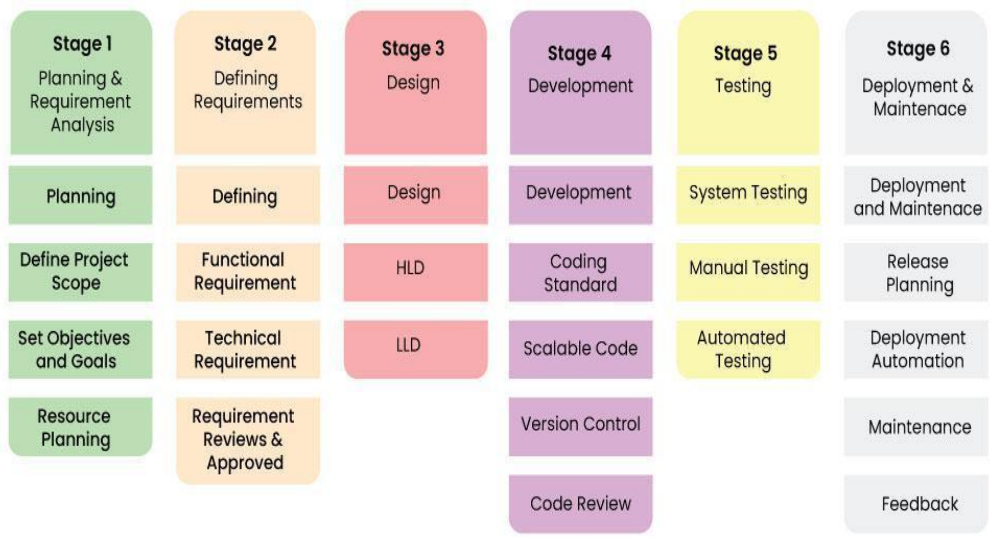
Software development life cycle (SDLC) is a structured process that is used to design, develop, and test good-quality software. SDLC, or software development life cycle, is a methodology that defines the entire procedure of software development step-by-step.

SDLC is a process followed for software building within a software organization. SDLC consists of a precise plan that describes how to develop, maintain, replace, and enhance specific software. The life cycle defines a method for improving the quality of software and the all-around development process.



Stages of the Software Development Life Cycle

SDLC specifies the task(s) to be performed at various stages by a software engineer or developer. It ensures that the end product is able to meet the customer's expectations and fits within the overall budget. Hence, it's vital for a software developer to have prior knowledge of this software development process. SDLC is a collection of these six stages, and the stages of SDLC are as follows:



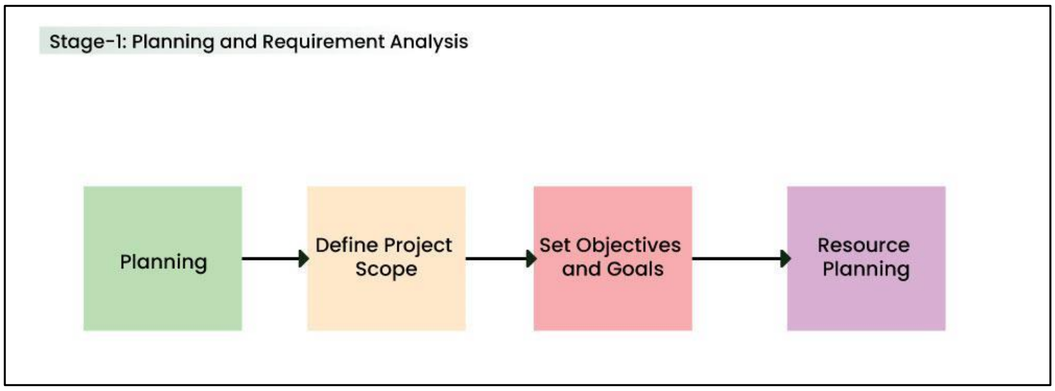
6 Stages of Software Development Life Cycle

The **SDLC Model involves six phases or stages** while developing any software.

Stage-1: Planning and Requirement Analysis

Planning is a crucial step in everything, just as in software development. In this same stage, requirement analysis is also performed by the developers of the organization. This is attained from customer inputs, and sales department/market surveys.

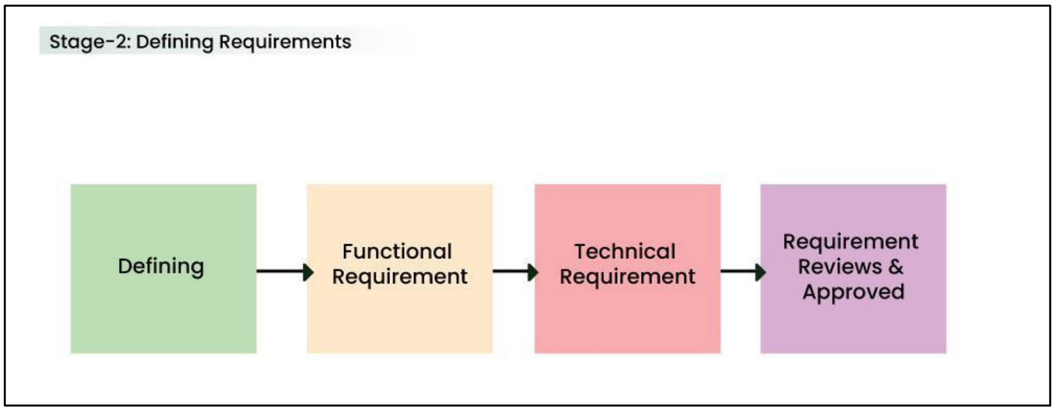
The information from this analysis forms the building blocks of a basic project. The quality of the project is a result of planning. Thus, in this stage, the basic project is designed with all the available information.



Stage-2: Defining Requirements

In this stage, all the requirements for the target software are specified. These requirements get approval from customers, market analysts, and stakeholders.

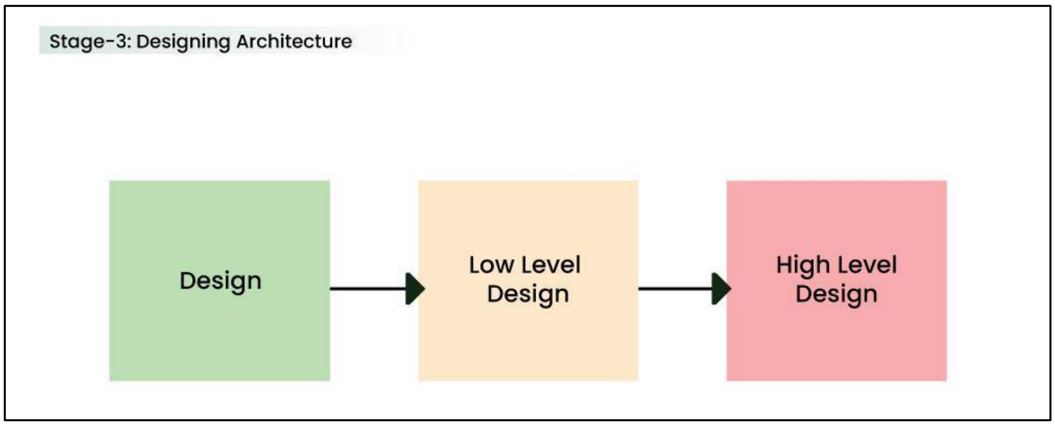
This is fulfilled by utilizing SRS (Software Requirement Specification). This is a sort of document that specifies all those things that need to be defined and created during the entire project cycle.


Stage-2: Defining Requirements
Defining
Functional Requirement
Technical Requirement
Requirement Reviews & Approved

Stage-3: Designing Architecture

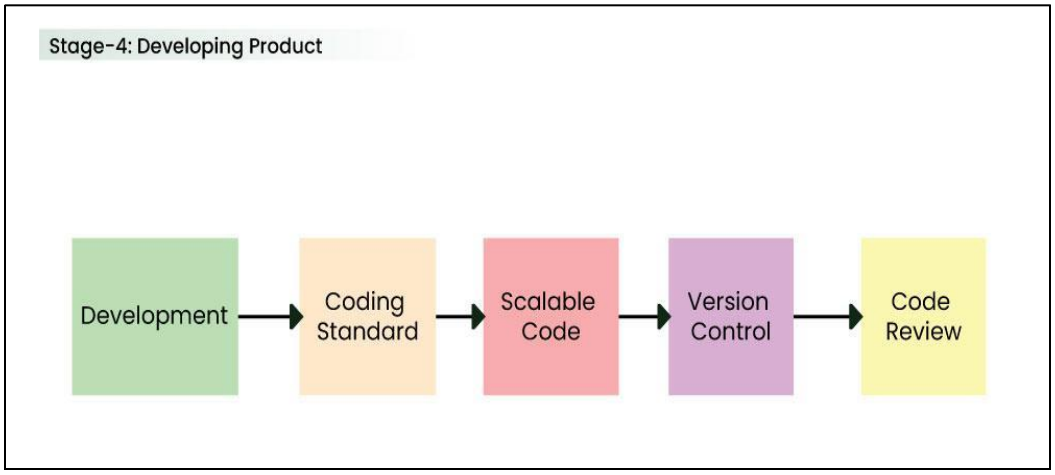
SRS is a reference for software designers to come up with the best architecture for the software. Hence, with the requirements defined in SRS, multiple designs for the product architecture are present in the Design Document Specification (DDS).

This DDS is assessed by market analysts and stakeholders. After evaluating all the possible factors, the most practical and logical design is chosen for development.



Stage-4: Developing Product

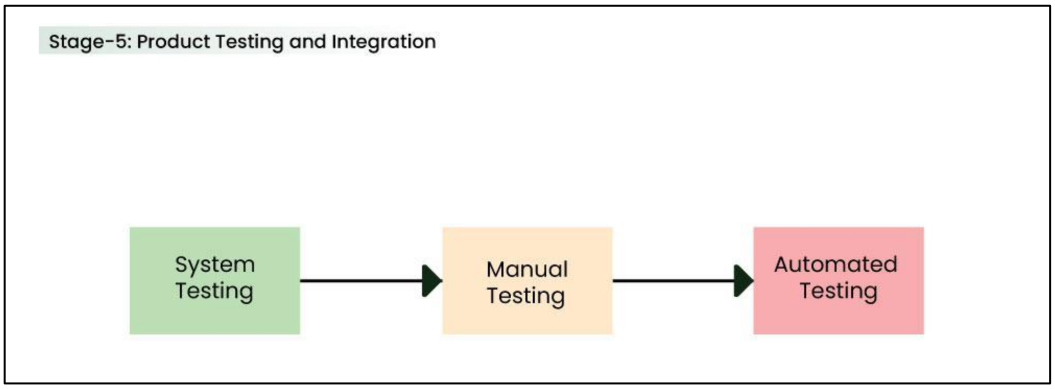
At this stage, the fundamental development of the product starts. For this, developers use a specific programming code as per the design in the DDS. Hence, it is important for the coders to follow the protocols set by the association. Conventional programming tools like compilers, interpreters, debuggers, etc. are also put into use at this stage. Some popular languages like C/C++, Python, Java, etc. are put into use as per the software regulations.



Stage-5: Product Testing and Integration

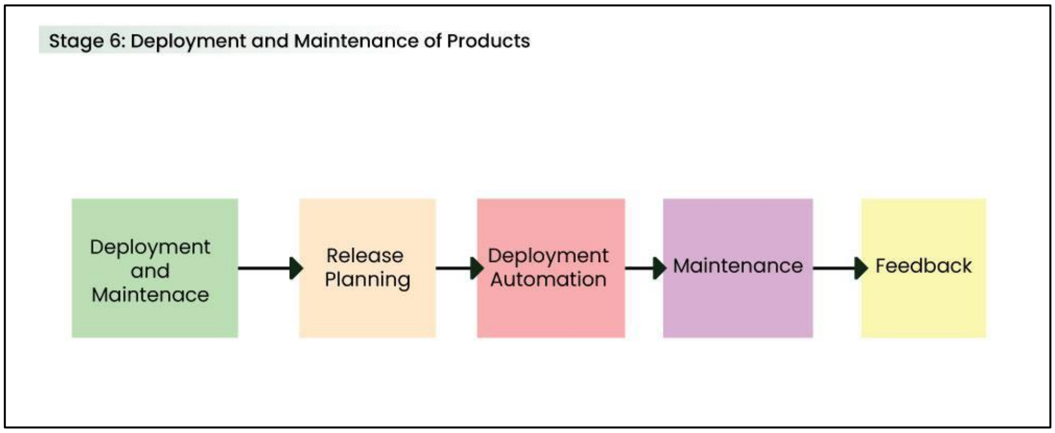
After the development of the product, testing of the software is necessary to ensure its smooth execution. Although, minimal testing is conducted at every stage of SDLC. Therefore, at this stage, all the probable flaws are tracked, fixed, and retested. This ensures that the product confronts the quality requirements of SRS.

Documentation, Training, and Support: Software documentation is an essential part of the software development life cycle. A well-written document acts as a tool and means to information repository necessary to know about software processes, functions, and maintenance. Documentation also provides information about how to use the product. Training in an attempt to improve the current or future employee performance by increasing an employee's ability to work through learning, usually by changing his attitude and developing his skills and understanding.



Stage-6: Deployment and Maintenance of Products

After detailed testing, the conclusive product is released in phases as per the organization's strategy. Then it is tested in a real industrial environment. It is important to ensure its smooth performance. If it performs well, the organization sends out the product as a whole. After retrieving beneficial feedback, the company releases it as it is or with auxiliary improvements to make it further helpful for the customers. However, this alone is not enough. Therefore, along with the deployment, the product's supervision.


Stage 6: Deployment and Maintenance of Products
Deployment and Maintenace
Release Planning
Deployment Automation
Maintenance
Feedback

What is the need for SDLC?

SDLC is a method, approach, or process that is followed by a software development organization while developing any software. SDLC models were introduced to follow a disciplined and systematic method while designing software. With the software development life cycle, the process of software design is divided into small parts, which makes the problem more understandable and easier to solve. SDLC comprises a detailed description or step-by-step plan for designing, developing, testing, and maintaining the software.

Real Life Example of SDLC

Developing a banking application using SDLC:

- ***Planning and Analysis:*** During this stage, business stakeholders' requirements about the functionality and features of banking application will be gathered by program managers and business analysts. Detailed SRS (Software Requirement Specification) documentation will be produced by them. Together with business stakeholders, business analysts will analyse and approve the SRS document.
- ***Design:*** Developers will receive SRS documentation. Developers will read over the documentation and comprehend the specifications. Web pages will be designed by designers. High level system architecture will be prepared by developers.

- ***Development:*** During this stage, development will code. They will create the web pages and APIs needed to put the feature into practice.
- ***Testing:*** Comprehensive functional testing will be carried out. They will guarantee that the banking platform is glitch-free and operating properly.
- ***Deployment and Maintenance:*** The code will be made available to customers and deployed. Following this deployment, the customer can access the online banking. The same methodology will be used to create any additional features.

Conclusion

In conclusion, we now know that the **Software Development Life Cycle (SDLC) in software engineering is an important framework for the better and more structured development of optimized software programs.** In a world full of rapid evolution in technology, SDLC phases plays a crucial role in enabling some good and innovative solutions for helping users and organizations. Also, it's better to adapt SDLC principles to achieve software development goals effectively.

Questions

How does SDLC work?

The SDLC involves planning the project, gathering requirements, designing the system, coding the software, testing it for issues, deploying it to users, and maintaining it post-release. Each phase ensures the software meets user needs and functions correctly, from start to finish.

What are the main phases of SDLC?

The main phases of SDLC include Requirements, Design, Implementation (Coding), Testing, Deployment, and Maintenance. These phases represent the stages a software project goes through from initiation to completion.

Why is SDLC important?

SDLC ensures a structured and organized approach to software development, leading to the creation of reliable and high-quality software. It helps manage resources efficiently, reduces development time, and minimizes the risk of project failure.

What are the key objectives of SDLC?

The key objectives of SDLC include delivering a high-quality product, meeting customer requirements, managing project resources effectively, minimizing risks, and providing a clear and transparent development process.

How does SDLC differ from Agile methodology?

SDLC is a more traditional, sequential approach to software development, while Agile is an iterative and flexible methodology. SDLC follows a structured path, while Agile allows for incremental development with frequent reassessment and adaptation.

Which of the following is not a life cycle model?

(A) Spiral model

(B) Prototyping model

(C) Waterfall model

(D) Capability maturity model

8 Software Requirements Specification (SRS)

What Is a Software Requirement Specification?

A software requirement specification describes what the product does and how we expect it to perform. It is the main point of reference for the entire team.

Software requirement specification (SRS) is a complete document or description of the requirements of a system or software application. In other words, "**SRS is a document that describes what the features of the software will be and what its behaviour will be, i.e. how it will perform.**" The advantage of SRS is that it takes less time and effort for developers to develop software. What SRS is is like the layout of the software which the user can review and see whether it (SRS) is made according to his needs or not. We learned about software requirement specification, what it is, now let us read its features.

A software requirement specification (SRS) includes information about all the functional and non-functional requirements for a given piece of software. The SRS serves as the main point of reference for the software development team who'll build the software product, as well as for all other involved stakeholders.

A software requirements specification (SRS) is a description of a software system to be developed. In order to write a good SRS, some common practices have to be followed which are as follows:

Communication Practices or Principles:

1. Listen to the speaker and concentrate on what is being said.
2. Prepare before you meet, by searching, researching and understanding the problem.
3. Someone should facilitate the meeting and should have an agenda.
4. Face to Face communication is best but also have a document or presentation to focus on discussion.
5. Strive for collaborations and decisions.
6. Document all the decisions.

Planning Practices or Principles:

1. Understand the scope of the project.
2. Involve the customer in the planning activity.
3. Recognize that planning is iterative and things will change.
4. Estimation should be done based on only what you know.
5. Consider the risk and it should be defined in the plan.
6. Be realistic on how much can be done on each day.
7. Define how you ensure quality can be achieved.

Construction Deployment Practices or Principles:

1. Understand the problem you are trying to solve.
2. Understand basic design principles and concepts.

3. Pick a programming language that meets the need of the software to be built.
4. Create a set of unit test cases that will be applied once the component you code is completed.
5. Constraint your algorithm by structured programming practices.
6. Understand the software architecture and create interfaces that are compatible with it.
7. Keep conditional logic as simple as possible.
8. Write a code that is self-documenting.

What Are the Components of a Software Requirement Specification (SRS)?

- Introduction
- Product Description
- Software Requirements
- External Interface Requirements
- Non-Functional Requirements

Do We Need Software Requirement Specification?

Yes, because an SRS acts as the single source of truth for the lifecycle of the software. The SRS will contain information about all the software components that make up the product or deliverable. The SRS describes those components in detail so the reader can understand what the software does functionally as well as how, and for what purpose, it's been developed. Finally, the SRS will contain sections that describe the target non-functional behavior of the software product, which include performance and security.

Why Is a Software Requirement Specification Important?

Having a solid SRS is of massive importance to software projects. This documentation brings everyone involved to the same shared understanding about the project’s purpose and scope. This documentation helps avoid misalignment between development teams so everyone understands the software’s function, how it should behave and for what users it is intended.

Through the SRS, teams gain a common understanding of the project’s deliverable early on, which creates time for clarification and discussion that otherwise only happens later (during the actual development phase). This means teams are more likely to deliver a software product that fits the original scope and functionality as set forth in the SRS, and that are in line with user, customer and stakeholder expectations.

How to Write Software Requirement Specification

Writing an SRS is just as important as making sure all relevant participants in the project actually review the document and approve it before kicking off the build phase of the project. Here’s how to structure your own SRS.

Introduction

This section presents the purpose of the document, any specific conventions around language used and definitions of specific terms (such as acronyms or references to other supporting documents), the document's intended audience and finally, the specific scope of the software project.

Product Description

This section outlines the high-level context that motivates the software product's development, including a summary of its main features and functionality. A very important component of the product description is an explanation of the product's intended user, what processes developers will use to accomplish their goal and for which type of environment this product is most well suited (business, customer, industry and so forth). The product descriptions will also contain any external dependency by which the product's development will be affected.

Software Requirements

The meat of the document, the software requirements section, describes in detail how the software will behave and the functionality it offers the user.

For example, a functional requirement may state a user will be able to upload videos using the user interface.

Detailed requirement information is usually laid out in the document as a written list of requirements broken down by key topic areas that are specific to the product. For example, gaming software may have functional requirements specific to players and the surrounding environment. Otherwise, you might have an external attachment to a requirements template wherein this template is a simple file that contains a granular list, or table, of requirements with key information (description of the requirement, who it's for, which version of the product it refers to and more).

External Interface Requirements

This section contains a description of how the user interacts with the software product through its interface, as well as a description of the hardware necessary to support that interface.

In addition, this section usually features a description of how the software will communicate with other software using the various available communication standards. For example, you might have descriptions of compatible message formats (such as audio or visual) as well as standards for the data size the product can send or receive by way of a specific user action.

Non-Functional Requirements

This section speaks to the software's target behavior considering performance, security, safety and quality. Questions this section may answer include:

- ***Performance***: How fast do users receive results upon executing a certain action?
- ***Security:*** How will the software manage data privacy?
- ***Safety:*** Is there any potential harm the product may create and what guardrails exist to protect the user, the company and (potentially) the public at large?
- ***Quality:*** How reliable is the product?

Conclusion:

So, the Software Requirements Specification is an important part of successful software development projects. This all-encompassing guide brings together stakeholders, helps in project planning and directs development activities. Whatever the changes, in the end it makes client satisfaction its own. Ignoring the importance of a correct SRS can there- fore mean project failure, increased costs in development, and dissatisfied stakeholders. Therefore, developing a powerful SRS is the first step on the road to success, for any software developer worthy of his salt.