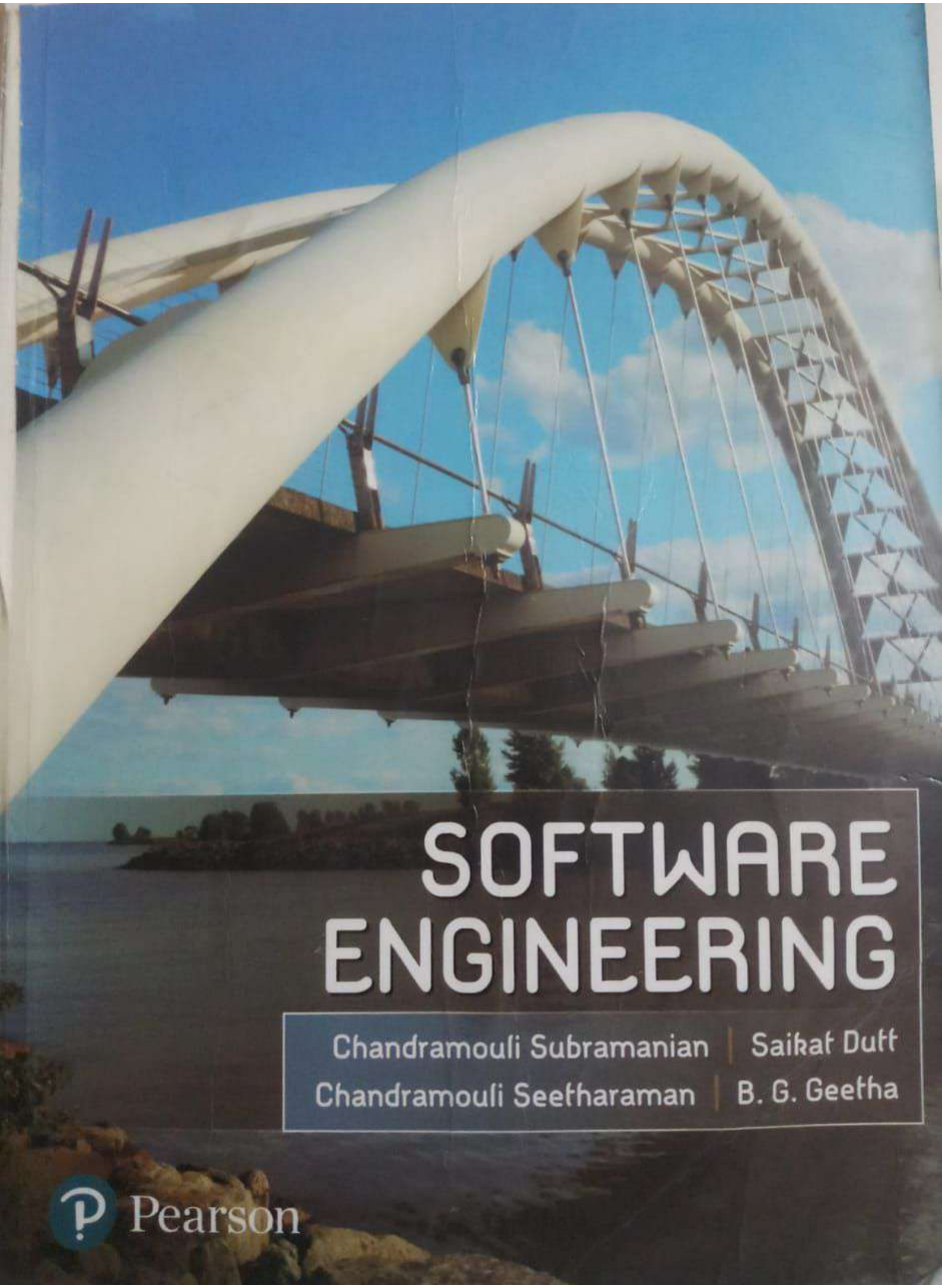


SOFTWARE ENGINEERING

Chandramouli Subramanian | Saikat Dutt
Chandramouli Seetharaman | B. G. Geetha

Pearson

Contents

Section 1:
Introduction to Software Engineering

1

Software Engineering – Introduction

CHAPTER COVERAGE

1. *Introduction to Software Engineering*
2. *Software Process*
3. *Software Process Models*
4. *Software Product*

1.1 INTRODUCTION TO SOFTWARE ENGINEERING

The success of mankind in performing a process with perfection and accuracy commenced with the origin of computers. The digitalized computing machines, initialized in 1940s, merely carried out arithmetic operations marking the first step toward today's innovative computers. The development attained today in the field of computing was never predicted. Speaking of computers, we cannot miss the components that make it up as a whole device. Man developed many individual components to aid him in day-to-day life. Every device invented simplified the efforts to a larger extent. Starting from the pulley, the wheel and eventually the computer, relieved the stress and strain and proved to be a masterpiece.

Initially, the computer performed simple tasks and produced results. However, the device has so rapidly progressed today to perform analysis, comparison of details and make decisions that its functions require only minimum human intervention. Applications of this incredible device can be seen everywhere. Extended functionalities of the computer have, to a great extent, reduced the need for investing human effort in nearly all fields. The primary components of the computer included a memory space to store the variables, a procedure to perform the operations and a display to show the results to the user. Instructions to the first-generation computer were provided by wired components, which were too messy. Gradually, the idea of storing the programmed instructions internally into the device without further need to input every single instruction was developed. This required a programming language to create a procedure for every task. A programming language has certain formats to guide the user for easier and faster designing of the program. Programs of individual functions

grouped together form software. Software is a collection of similar tasks that provides a single interface to the user to obtain the results for a variety of requests. Software is the controlling unit of a hardware component. Both the components complete the computer structure; software or hardware alone is useless. A memory device, a power unit, capacitors, resistors, processor, monitor, and cables to connect are the hardware components used in a computer. The software defines the functions of data transfer, path of control messages and transfer of input and output between the hardware.

This chapter explores the concepts of software, its development stages and explains its importance in the evolution of computers. The application of software involves many tiring tasks that have evolved to the present stage. Different methods have been proposed for the development of software and its deployment in a suitable environment. The deployment of software does not end the life cycle of a software, but it continues until the next version is developed. The development team continues to have its relationship with the software as they release periodic updates of the software to meet the ever demanding needs of the clients and the end users. Software engineering deals with the terminologies of designing and delivering software to the client's satisfaction. The origin of software, its phases, the level of applicability and the role it plays in the functioning of the computer are discussed in the following sections.

1.1.1 What Is Software?

Software is defined as the tool with designated order of instructions, performing tasks to provide the desired results after obtaining the requirements of the users. The methodology of how to perform the operations in an optimal path to obtain accurate results quickly states the functions of computer software.

Software also consists of procedures, rules, software-related documents and the associated basic data required to run the software (Refer Figure 1.1).

Software = Computer Programs (Instructions) + Procedures + Rules + Associated documents + Data

Procedures and rules include step-by-step instructions of how to install (deploy) and use the software. Documents include requirement documents, design documents and testing-related documents. To run the software basic data needs to be fed in.

Computer software resides in the memory space of the computer, commanding it in every situation. The process of the software begins with the booting of the computer until it is shut down.

POINTS TO PONDER

Software = Computer Programs (Instructions) + Procedures + Rules + Associated documents + Data

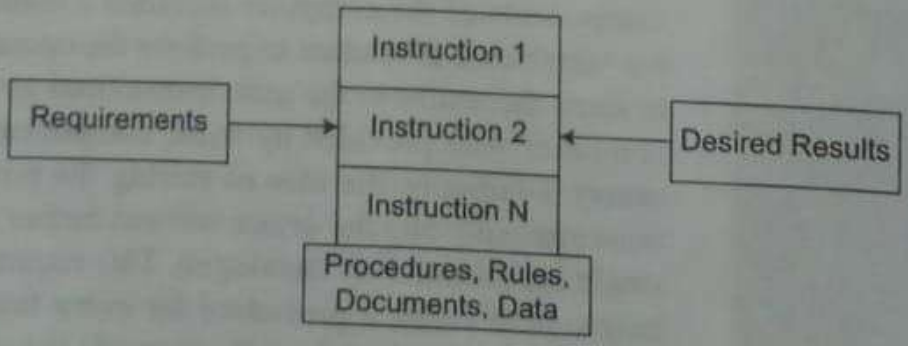


Figure 1.1 Components of Software

Examples of Software:

1. Microsoft Word
2. Microsoft Excel
3. Microsoft Power Point
4. Linux
5. MySQL
6. Adobe Photoshop

1.1.2 Evolving Role of Software

Software has evolved from a single instruction provided to the machine to the present stage of decision-making software (refer Figure 1.2). Initially, the devices were operated by inputting single real-time instructions; later a programming language framed a structure to combine programs into software. The software was then divided into categories based on their implementation areas.

Individual programs perform an operation; collectively they form software and perform a function for the user (refer Figure 1.3). Software is classified into two types based on the areas they command, namely application software and system software.

System Software comprises programs that control the operations of the hardware components (refer Figure 1.3). The technical functioning of the computer is dependent on the operating system, built in with a number of adequate system software. System software controls the transfer of messages handling the internal processes, which are mostly abstracted from the external users of the computer.

Examples:

1. Operating systems like Dos, Windows and Linux.
2. Device drivers (hardware controllers) such as video device driver software.

Application Software consists of programs purely for the tasks of the users, obtaining the inputs and processing them in the instructions already defined (refer Figure 1.3). They have no access to the hardware and the prerecorded system software of the device. As they are intended for interfacing and processing with the user, they have been adorned with the unifying taste of the users.

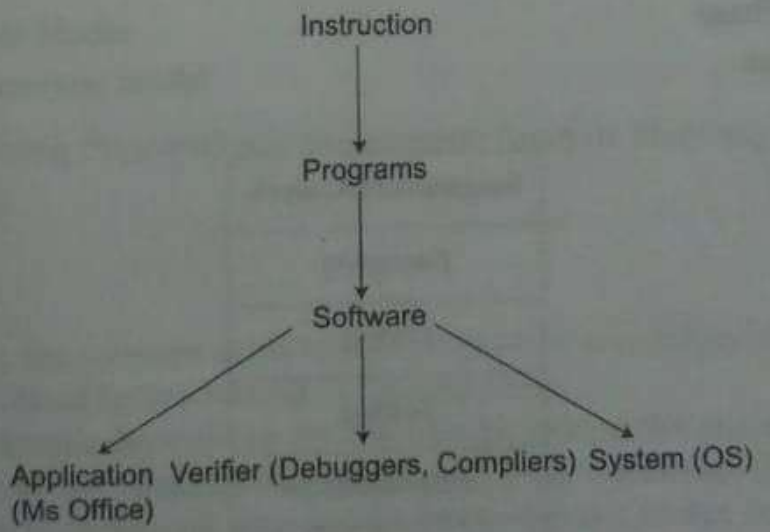


Figure 1.2 Evolution of Software

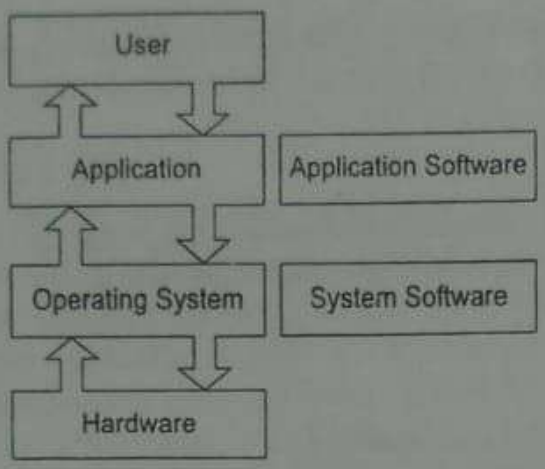


Figure 1.3 Interrelationship of Software

Examples: MS Office suite, Microsoft Calculator, Notepad, etc.

There are other types of software which are dedicated to the verification of the programs in terms of coding, formats, flow of control and order of execution. These verifying software are not involved in any other part of the operation. They are used to detect and report the syntax errors in the designed program. Without the report of verifiers and stated rectifications the program is not allowed to execute.

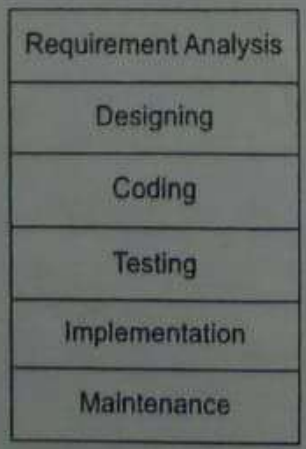
Examples: Compilers and Debuggers.

1.1.3 Phases in Software Development

The phases in software development define the steps of organizing the development of the software. Each step is called a phase. Each phase will have at least one deliverable (output).

The general software development phases are described as follows:

1. Requirements Analysis Phase
2. Designing Phase
3. Coding and Testing Phase
4. Implementation Phase
5. Maintenance Phase



1.1.3.1 Requirements Analysis

The requirements analysis is the first phase of software development wherein the customer and the business analysts (usually developer) interact with each other and document the requirements of the outcome of the software. The customers would state their requirements how and what the end product should perform. Requirements form the stepping stone of any project or product. The requirements include the appearances (Interface requirements), the functionalities (Functional requirements) and the software output expectations (Nonfunctional requirements). The requirement gathering team discusses the noted requirements with the experts for their implementation into the product, which is called as analysis part of the phase. The expert team analyzes the possibilities and impact of implementing the stated requirements. The analysis of the requirements includes the budget and the time span required for completing the product. The complete analysis of the requirements is documented for future reference.

It is easy to fix a problem in the requirements at the requirements analysis stage. However if that faulty requirement percolates through later stages of design and coding, then it would cost 10–100 times more to fix the same. Capturing the requirements fully and correctly at the early stage of the project is essential to make the project profitable.

Types of requirements include:

- Functional Requirements
- Nonfunctional Requirements
- Interface Requirements

The steps in Requirement Engineering phase include:

- Feasibility Study
- Eliciting Requirements
- Requirements Analysis
- Specify and Validate Requirements
- Requirements Management

Requirement Analysis Models (Approach) include:

- Structured Analysis Model
- Object Oriented Analysis Model

Requirement Engineering Principles and Requirement Analysis Modeling are discussed in Chapters 2 and 3, respectively.

1.1.3.2 Designing

During the design phase, the software requirement is converted into design models. Each design outcome is verified and validated before moving to the next phase.

The design of the software depends on the complexity level of the product, the graphical interfaces required and ease of use requirements of the end users. The design should not leave out any of the agreed requirements of the user. The Design Phase acts as a bridge between the Requirement Analysis phase and the Coding Phase. The Design Phase converts the business requirements into technical requirements. The design has no functionality but only the structure of the product.

The design of the software product should

1. Be precise and specific
2. Conform to budget and requirements
3. Provide a direct solution to the problem

Modularity, cohesion, coupling and layering are the factors to be considered while designing the software. UML is a modeling language used to model software and nonsoftware systems. UML and the above factors are discussed in detail in the later chapters.

Software design includes Data Design, Architectural Design, User Interface Design, Procedural Design and Deployment Level Design.

Software Design Concepts discussed in Chapter 4 includes design models, architectural engineering, etc.; Object Oriented Concepts in Chapter 5 include Class, Objects, Inheritance, Polymorphism, Design Patterns, etc.; Object Oriented Analysis and Design in Chapter 6 includes Use Case Modeling, UML notations, etc. and User Interface Design is discussed in Chapter 7.

1.1.3.3 Coding and Testing

Coding is defined as the phase where the documented design is transformed into lines of codes in programming language. In a typical waterfall model, coding is done after the requirement analysis (requirement clarification) and design. Writing standardized codes for software helps not only to maintain it properly once it is built, but also to enhance the code easily at a later point of time; it also helps to identify and fix the bugs quickly and easily. Therefore just writing a proper working code is not enough, but the focus should be on writing a standard code. So codes are instructions that execute the program in the environment where it is supposed to be. The designs in high level language are converted into machine level language for implementation in the real-world environment. Coding conventions and coding standards are discussed in detail in the later chapters.

After coding phase is completed, the code have to be tested for ensuring its right functionality in the platform. Testing is carried out by a separate team of experts (software testers) specialized in generating sample cases and subjecting the product to various tests. Only after the product has been proved to be reliable by the tests, it enters into the implementation phase. The process of testing and its procedures are discussed in detail in the later chapters.

Chapters that are related to coding and testing aspects are:

Chapter 8 Software Coding

Chapter 9 Introduction to Software Measurement and Metrics

Chapter 10 LOC Function Point and Object Oriented Metrics

Chapter 19 Software Testing Plan and Test Case Preparation

Chapter 20 Software Test Automation

1.1.3.4 Implementation

This phase involves the release of the developed software into the market and to the end users. This phase is also called as Deployment Phase. The tested software product is then implemented in the customer's domain. The software developed is installed in the platform for its prescribed function. The success of the implemented product depends on how much the end users like the product and prefer to use it regularly. There will be similar products satisfying the same functionalities from competing organizations in

the market, racing for people's attraction. Hence, the implemented product should be simple to use (user interface intensive organizations) and accurate (in the case of financial and trading organizations).

1.1.3.5 Maintenance

Software maintenance is the core aspect of software engineering. After the system development, the system gets deployed into the production environment. The process of modifying the production system after the delivery to correct the faults, improve the performance and adapt to the changing environment is called as software maintenance. The responsibility of the team of developers does not end after the product is implemented, but continues for a considerable long time. Training has to be provided to the clients and users on handling the product. Without proper training and knowledge on how the product works, there are chances of errors and failures. Every user will not be skilled in the technical aspects of the software other than the developers themselves. The software is maintained and upgraded with ever increasing demands and requirements of the clients and end users. All the changes during the maintenance stage are clearly documented for further reference. When a new product of similar characteristics needs to be developed, these documents are referred for analyzing the components which could be reused. The reusable component shortens the time and effort of the development team by facilitating them to skip the design and coding process for that particular portion of the code. The maintenance and its corresponding process are discussed in Chapter 9 Introduction to Software Measurement and Metrics.

1.1.4 Software Characteristics

A software has the following characteristics (refer Figure 1.4):

1. **It should be complete:** The software product being developed should meet all the requirements of the customers no matter how complex the product is, and the requirements agreed upon should be present in the product. The final product should not miss out any of the required functions.
2. **It should be exclusive and reliable:** Unique software delivers its ensured performance to a particular problem, and should concentrate on the given problem or requirements. The solution obtained after using the software should be distinct. The software needs to be stable for a considerable time and usage, providing an unfailing service.
3. **It should be precise:** Unwanted steps and operations should be eliminated and shorter operations with the right solution should be preferred. Unwanted steps lead to wastage of resources, manpower and hence the cost. These extra operations would have no effect on the final solution of the software. Selective and adequate processes should be enough and thus the structure needs to be clear-cut.
4. **It should be efficient:** A well organized software produces an optimal result at every moment of usage. The structure of the software needs to obey a strict and predefined order

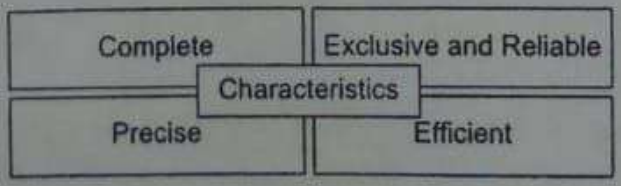
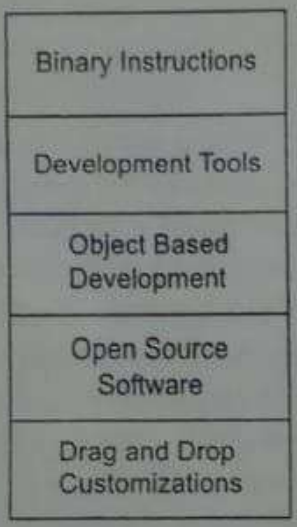


Figure 1.4 Characteristics of Software

to be efficient. It should also be flexible in nature so that it can be extended at a later point of time.

1.1.5 Changing Nature of Software

In the beginning, binary instructions were fed into the computer machine. Today the software has evolved to a matured level and has sophisticated tools to develop the software and no longer requires to type the binary instructions. Now-a-days Object-based software development is in use. The urge of increasing features and demands of the business users lead to necessary changes in the software development. The source code was earlier visible only to the development team but is now available to the customers as well. Open source software is a big hit in the market and anybody can change the code while using and customizing it as per the need. Tools are now developed to facilitate the users to design a software product themselves by dragging and dropping the components (contents) at their intended positions. Proper knowledge of coding and testing is not required to design and code the software. Software development tools simplified the design and implementation phases drastically. The process of developing software becomes simple.



1.1.5.1 In-built Software

Various mechanisms of automobiles, planes and robots employed for various operations in the fields of military and research involve dedicated software with complex operations and decision-making skills. These devices need embedded software as they involve serious analysis and take appropriate actions at the right time. The in-built software are deeply sensitive and efficient, prescribing the intelligence functions of the devices. The intelligence includes decision-making skill sets because no human can be practically involved to command the robot, which reports on the atmospheric conditions of the planet Mars. Engineering and scientific research use this type of software mostly, reducing the participation of the mankind in in-human conditions.

1.1.5.2 System-based Software

The system-based software has also some prescribed function, but they are limited to simpler devices. Home computers or computers of any kind necessitate an operating system and additional driver softwares for the functioning of the entire system. The system-based software is related to programs,

which define the functionality of the hardware components of a particular computer. The operation of a computer as a whole is facilitated by the system software containing the basic and advanced modes of operations at different instances of the end user. Without this software, the hardware is merely a machine. Compliers, linkers and debuggers are also some of the important system software for testing the programs.

1.1.5.3 Application-based software

These kinds of software delivers functionality or service to the requesting users. They obtain an input condition or a query from the end user, analyze it and display the results on the monitor. They are not as complex as the above-mentioned type. The structure and the language used for the applications vary by a large extent, such as for a web-based application the HTML and XML languages are used while for a database application, an ORACLE program is used. Different applications possess distinguishable styles and frameworks. Every field has its very own requirements, area of interests, functional aspects and thus different application softwares belong to different category and need not be the same.

1.1.6 Software Myths

The software development brackets many myths. The myths have created an incomplete and false image of the software development processes. Some of the myths are listed below.

1.1.6.1 Software Is Better than Implementing Devices

Devices have been limited to the wear and tear prone to usage. They include the definitions of being used. Software programs are theoretically known to change indefinitely with alterations in the code and reusability factor. Software does not come with an expiry date, but practically there is a limit to its usage. Although the changes are easier to be updated in the software than devices, they need to be verified.

1.1.6.2 Reusability Ensures Better Solution

The conceptual model of reusing a component is an incredible strategy to improve the performance reduce the time in designing and implementing the software. But there is a major factor of adaptability of the existing component into the newer component. Certain modifications may be needed in the component to be reused. Testing identifies errors and reports these kind of compatibility issues. Further remedial actions are taken for a component to be the best fit into the product.

1.1.6.3 Additional Features Add Efficiency

As already mentioned, the product being developed needs to have a direct and dedicated solution to the present problem. More straightforward software is preferred over the software with additional and unwanted features. One has to keep in mind that the product may not run accurately at the very first time, and testing alone cannot identify all the problems.

1.1.6.4 More Designers–more Delay

This is absolutely not true because the development team comprises of engineers with great knowledge on the strategies of rapid development. Expertise and experience of supplementary engineers will surely help in deploying the product at a faster rate. But there should be balance between how much work is pending and how many resources we deploy to get that done.

1.1.7 What is Software Engineering?

Software engineering comprises strict disciplines that need to be followed to develop a programmable solution to solve customer's problems.

Strategies are defined to obtain a viable, efficient and maintainable software solution, which takes care of the costs, quality, resources and other expenditures. These strategies have been proposed by various engineers after implementing and analyzing the merits and demerits of each strategy.

1.1.8 Why Study Software Engineering?

Software engineering encompasses every concepts and standards of the disciplines in relation to design, coding and developing maintainable software complying with the available resources and budget constraints. Without a minimal intellect on software engineering and its disciplines, the developer would find it difficult to cope and control a vast area of designing, converting the design into a code, and finally implementing and upgrading the product with the changing market and conditions. Every developer cannot just jump into the ocean of creativity and in-depth programming skills without an assessment of software engineering terminologies.

1. It is the basic of any software development
2. It helps to produce reliable, reusable and quality software
3. Software engineering understands the customer needs and develop the software
4. It helps the developers to understand the disciplines of software life cycle
5. It helps to develop software in efficient way
6. Software engineering teaches the best practices of software development
7. Software engineering helps to write software, which can be integrated easily with other software

1.1.9 Generic View of Software Engineering

The sequential steps that describe the stages of a product being developed are the generic view of software engineering. Every action required for the development is categorized into three generic stages (phases) as follows (Refer Figure 1.5):

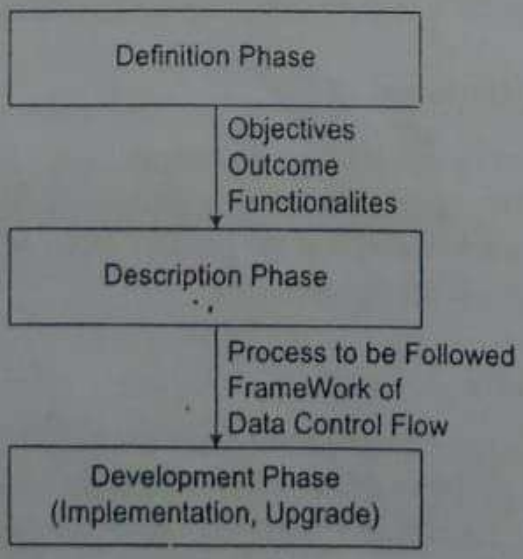


Figure 1.5 Generic view of Software Engineering

1. **Definition Phase:** The desired outcomes, functionalities and objectives are obtained from the customers and documented. These objectives are the driving force for a product to be developed. The team of developers and customers agree upon the final and feasible list of goals to be implemented into the product.
2. **Description Phase:** The next generic phase takes the necessary steps for further development; the desired outcomes are framed with optimal preceding and succeeding processes. The framework of data and control flow in a product, how the processes are ordered and how the functionalities are defined are described in this phase.
3. **Deployment Phase:** After the processes are described, they have to be transformed into the corresponding codes and implemented in the real-world environments. Implementation of the software product also includes the training provided to the customers and end users, maintenance and upgrade at regular intervals.

1.1.10 Role of Management in Software Engineering

Management of the software products depends on the following factors (refer Figure 1.6).

1.1.10.1 Participants

Members with various skill sets, from the team member to the team leader and manager, are responsible for every process of software development. The customers are the initial participants of a product. They put forward their problem and request a solution. A team of skilled personnel will discuss different solutions obeying the technical and administrative constraints. This participating team of developers is assigned by the top-level management based on the acquired and proficiency level of each member. Staffing requires the analysis of how proficient a member is in a particular field.

Managers play a crucial role in the development of the software. They are not only responsible for the final outcome of the project, but also responsible for the execution of the project from the beginning to the end. The manager prepares a plan and tries to execute the project as per the plan to create the deliverables. Manger is also responsible for managing scope, time, cost and quality of the overall project.

1.1.10.2 Procedure

Procedure describes the way in which the product is being developed. The order and types of procedures are devised by the team members and the final plan to be implemented is approved by the team

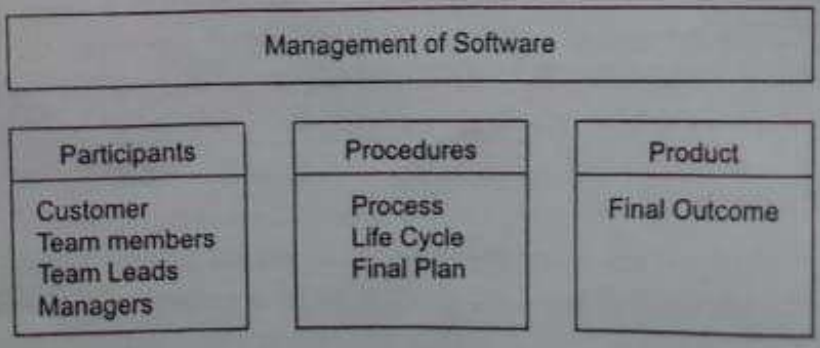


Figure 1.6 Participants, Procedures and Product

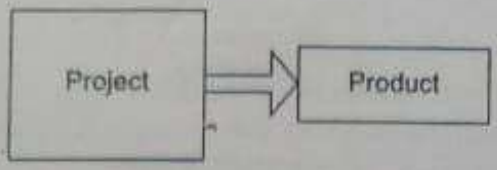


Figure 1.7 Project vs Product

manager. Managers also determine the life cycle that has to be followed among the various models. The best option would help to conserve the resources and money invested by the management. A slight deviation from the proposed procedure would disturb the efficiency of the entire process.

1.1.10.3 Product

The selection of best plans and procedures, and assigning the right and efficient staff to the project ensure the success of the project thereby ensuring the final outcome of the project called as product (refer Figure 1.7). Product is the outcome of the project.

There are many models of developing a product, evolved from the very beginning era of software development. Different models proposed increased the efficiency of the team of developers by altering and speeding up the actions of normal methods.

1.2 SOFTWARE PROCESS

A set of interrelated actions and activities to achieve a predefined result is called as process. Process has its own set of inputs and produces the output(s). Tools and techniques acts on the process to achieve the desired outcome. The processes have to be planned with great care and the knowledge about the effects of each implementation and activity on the product should be understood clearly by all the stakeholders. Groups of individual processes constitute whole software.

For example: Preparing a dosa (an Indian dish) as a process. If we call five different people (chefs) and give them the same ingredients and tools required to make the dosa, will the output be the same? The size and taste of the dosa is bound to vary. Why is it so? This is not only because of the difference in skills possessed by the chefs, but also because of the techniques they have learnt over a period. Thus, we need to understand and learn the techniques of a process to produce a better output.

In the following few sections an overview of different software development processes are given.

1.2.1 The Linear Sequential Model

A sequence of actions describes the order in which the execution of activities are planned and followed. Linear sequential model was the initial step in developing a software product, otherwise known to be the lifecycle model (refer Figure 1.8).

The process of development is divided into sequence of actions (steps) from requirements analysis, designing, coding and testing, implementation and finally maintenance. Each activity is carried out in a linear order (in sequential fashion) and no other order of execution is followed other than the flow mentioned. The result of the first activity is forwarded to the next stage and thus to the final stage. Without the completion of the previous step and forwarding the result, the next activity cannot be executed. It follows a strict sequence and it is strongly believed that no step can be overlapped.

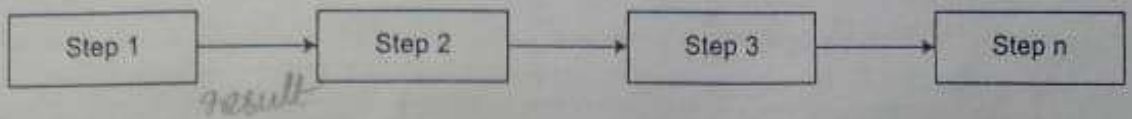


Figure 1.8 Linear Sequential Model

This model is a straightforward approach. Linear Sequential model is also called Predictive Life Cycle Model and Classical Life Cycle Model. Prototyping Model and Rapid Application Development Models can be the best examples of this model.

Examples for Linear Sequential Model include developing a portal that follows the steps of Requirement Analysis, Design, Coding, Testing and Implementation.

Some of the advantages of Linear Sequential Model are:

1. Easy to understand
2. Easy for implementation
3. Widely used and known; hence all stakeholders understand this
4. Identifies deliverables and milestones upfront.

Some of the disadvantages of Linear Sequential Model are:

1. This model assumes that requirements can be frozen before starting the Design Phase, which is not true in most of the situations.
2. Requirements are elaborated over a period of time.
3. Strictly following this model takes longer time to finish the project as each step takes its own time.
4. Specialists are required to run each step, which is difficult in long-term projects.

1.2.2 A Layered Technology

Similar to the linear sequential model, the layered technology (refer Figure 1.9) introduced the concepts to use the software technologies and terminologies even better than the previous strategies. A layered approach ensures a much stronger product. The layered model of software engineering divides the activities into four broad categories:

1. Quality Process: For the successful engineering process there should be a strong base of quality processes.

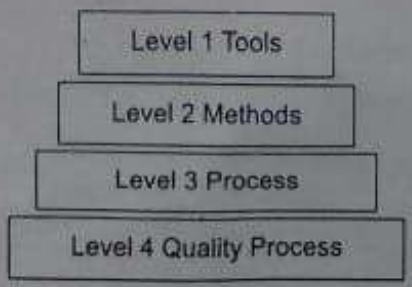


Figure 1.9 Layered Technology

2. Process: The software engineering processes ensure that all the technology layers work together and enable the timely development of the software system.
3. Methods: The methods in software engineering provide the technical capabilities to the system. The requirement analysis, design, modeling, development, testing, etc. are the tasks related to the methods.
4. Tools: The tools provide the support to the process and methods by making the tasks automated or semi-automated.

For example: the roads are layered with four or five layers (refer Figure 1.10). The bottom layer is a combination of clay and large stones, followed by another layer of smaller stones and tar. The next layer combines tar and even smaller stones for providing a neat finish. The quality of the roads depends on the tar used, the number of times run over by a roller machine and the time given for the settlement of those combinations. The layered technology is compared with many construction techniques for ensuring a well-built structure.

Layers are responsible for defining the activities in the order of execution. The layers begin with the requisition of input conditions and storing them at a location, and the next layers are dedicated to describing the most advantageous processes and methodologies to produce the preferred outcome. The last layer concentrates on submission of the designed product with the best quality beyond every difficulty. Developing a quality product includes important approaches, such as selecting and engaging valued resources and staff.

1.2.3 Prototyping Model

The product once developed using layered approach cannot be reversed easily. But in the real-life situations, the requirements are subject to numerous changes until they are implemented into the product. Thus with the ever-changing requirements, the product cannot proceed with mere predictions and prove to be right.

Hence the prototyping model (refer Figure 1.11) was proposed with a completely different strategy. Every product to be developed initially receives the requirements from the customers. With the given requirements, a sample or test model called the prototype is developed and displayed to the customer. The customer would respond by giving their opinion and rectifications needed over the current prototype. The prototype is altered until the customer is satisfied on the style, design and execution of the processes. Otherwise the development team continues to consider different options for creating the prototype. The final list of requirements reported will lead the development team to the next phase of design.

There are two types of prototypes (refer Figure 1.12) developed in this model, namely evolutionary and throwaway prototypes.

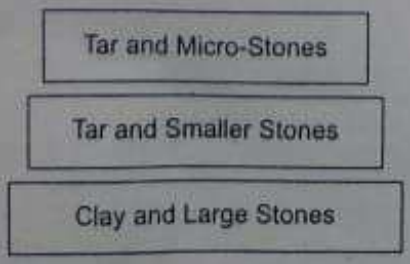


Figure 1.10 Example of Layered Technology

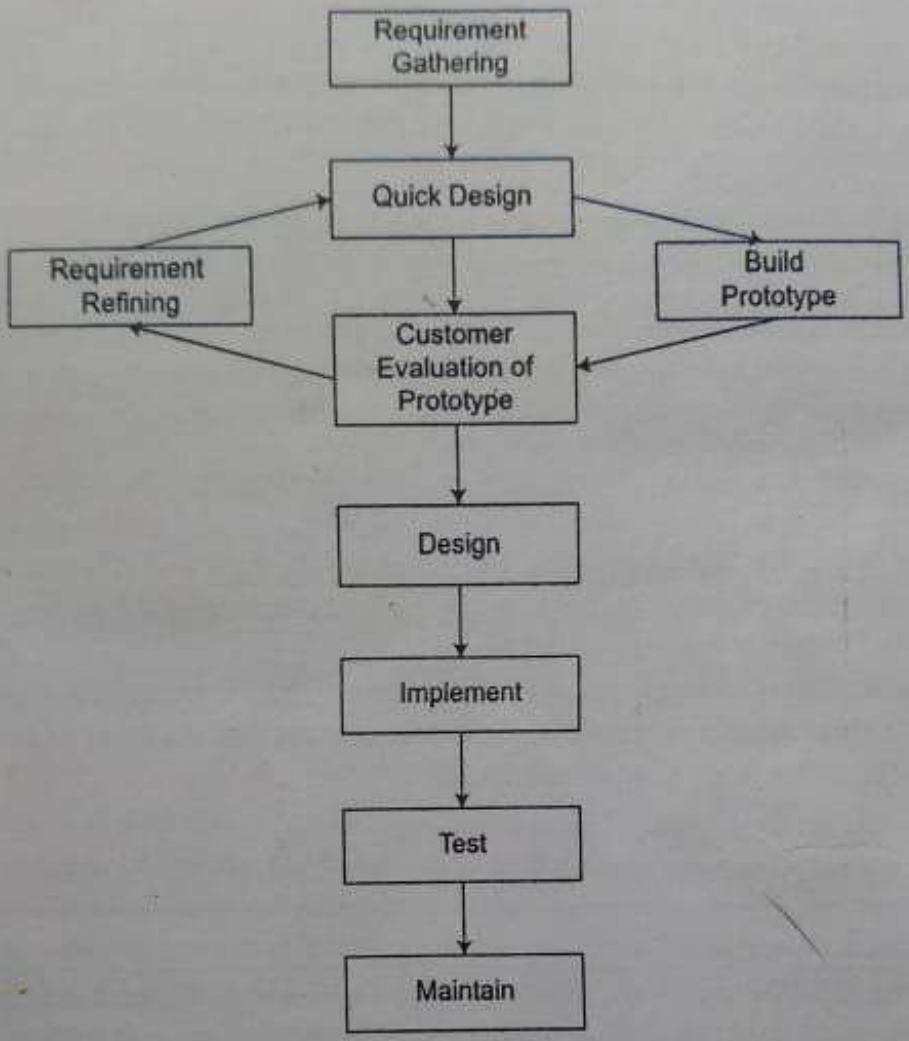


Figure 1.11 Prototyping Model

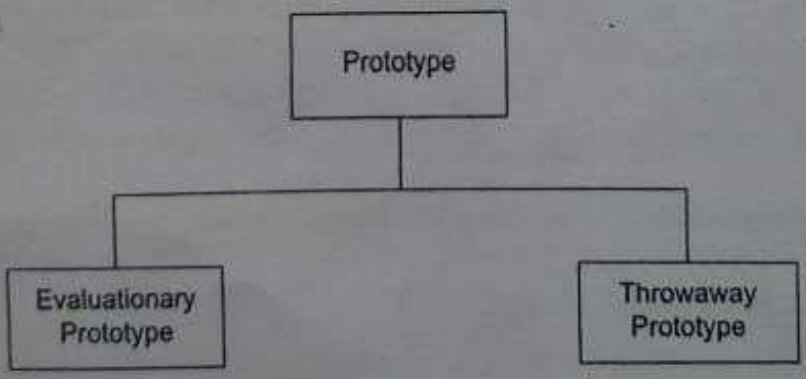


Figure 1.12 Prototyping Model

In the former type wrong prototype will be reused and the changes are implemented in that particular prototype. The new changes and corrections are updated in the existing prototype until the final list of requirements is obtained, without wasting the used resources. The other type, as the name suggests, destroys the wrong prototypes. Throwaway prototypes are proved to be wrong and thus the new prototypes are developed in the next stages. The resources used in the previous prototypes are either freed or rejected.

POINTS TO PONDER

Two types of prototypes are Evolutionary Prototype and Throwaway Prototype.

1.2.4 RAD Model

Rapid Application Development (RAD) model is the methodology proposed for quicker design and deployment of a product (refer Figure 1.13). This model includes the technologies and concepts of prototyping and iterative designs. RAD model requires a very short time span of 60–90 days for completing the software and delivering it to the customer. Unlike the traditional approaches, many innovative and simpler strategies of less time consuming activities are executed by the team of developers. RAD model proves to be the most preferable solution to quick needs and implementation of web-based applications. An organization needs to promote itself through World Wide Web by deploying a web application. This application should be simple to use and produce the desired results. Already present with numerous applications in web-based concepts, the customer may have to just refer the available applications and select from the same. Reuse of existing prototypes in the referred application facilitates faster deployment of the new application.

RAD model includes the following five phases for development:

1. Business modeling
2. Data modeling
3. Process modeling
4. Application generation
5. Testing and turnover

Prototyping + Iterative

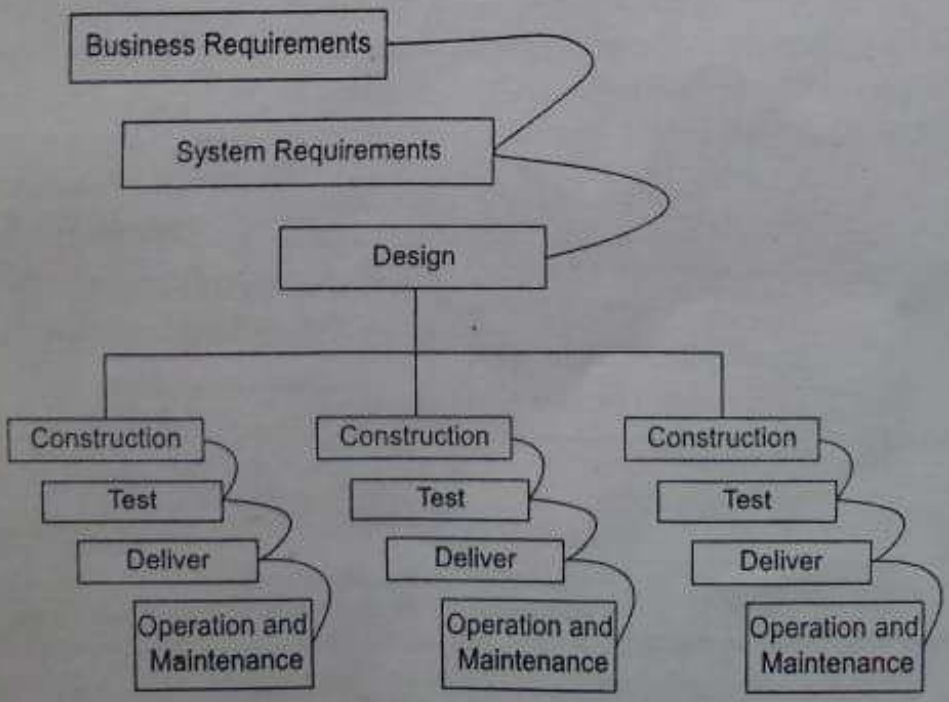


Figure 1.13 RAD Model

1.2.4.1 Business Modeling

Business modeling is the stage in which the types of information that are prime factors of the application are defined. The category of information involved in the processes, the procedures that use the information for various computations, initiators of the information and transfer of data between the processes are defined in the business modeling phase. With the details on the information being processed, the area of functionality of the application can be determined.

1.2.4.2 Data Modeling

The obtained information from the previous phase is filtered into useful and meaningful sets of data. They are refined into entities of high importance. The characteristics of the entities and their relationships are also defined in this data modeling phase. Attributes and entities enable a better understanding on the chief factors around which the entire processes are framed.

1.2.4.3 Process Modeling

Process modeling defines the activities needed to process the entities. Every instruction is framed in this phase. The instructions concern the processing of the data from the initial to the final stages of the application. Processes are defined for the storage and retrieval of the objects, their attributes and their associated metrics. Without proper design of these processes, the desired outcome cannot be achieved.

1.2.4.4 Application Generation

Application generation is a phase that uses automated tools to generate the working model of the designed application. The automated tools are used for faster analysis of similar components for reuse and save the time for new designs. Software tools have provided a wide range of amenities for faster deployment of an application with the same accuracy.

1.2.4.5 Testing and Turnover

The final phase is to test the correctness and consistency of the developed application. With such a short time and implementation of many reusable components, the accuracy of the application has to be tested with greater care. Reusing a component involves increased risk factors because the interfacing modules need to relate to the different modules of varying applications.

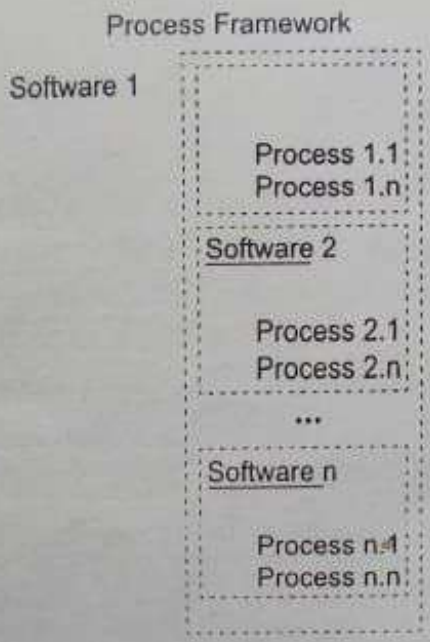
POINTS TO PONDER

The business modeling is the stage in which the types of information that are prime factors of the application are defined. Application generation is a phase that uses automated tools to generate the working model of the designed application.

1.2.5 Process Framework

Framework activities are processes of basic functionalities and are common to almost all software products. These actions can be applied to every product without any further modifications. For example, a user login form always commences further session of every user. The login form consists of

only two requirements: username and password. The new application with the same requirement need not create a new design but can reuse the whole module. The login form also requires a database for storing the usernames and the corresponding passwords.



1.2.5.1 Analysis on the Process Framework Activities

1. **Gathering the requirements:** The customary process framework activities begin with the analysis and observations made on the requirements. This process called as communication phase involves intended customers and the team of developers. Concluding the communication process produces a list of gathered and approved requirements.
2. **Scheduling and staff allocation:** Based on the gathered information and the plans the model is selected for development, processes are framed and the analysis on the feasible risks are made. In addition, the schedule for the development is also agreed. Appropriate members are assigned depending on the knowledge possessed for every task.
3. **Design and Deployment:** The team members start designing and developing the coding for their assigned task. The design process may be eliminated by reusing a module. Efficient modeling and processes ease the risks at the later stages of a product.
4. **Supplementary Framework Activities:** The product being developed has to be checked at regular intervals to ensure that it is proceeding in the right path. Slight deviations from the intended path would result in starting over the whole process again. Risk analysis is also another major course of action to prevent disastrous outcomes at the end. Detection and mitigation of risks at the earlier stages would shorten the final stages.

POINTS TO PONDER

Framework activities are processes of basic functionalities and are common to almost all software products. These actions can be applied to every product without any further modifications

1.2.6 Capability Maturity Model Integration

Capability Maturity Model Integration (CMMI) is an approach for improving the process level operations (refer Figure 1.14). The CMMI is a compilation of many best approaches to enhance a product's efficiency, quality and endurance. It collects the strategies for improving the processes at many levels of functionality, on teams, tasks, subtasks, and the whole project. As the name denotes, many actively superior methodologies are combined and incorporated into a single process for betterment in many ways. Individual products have their own processes perfectly designed and implemented. Identifying such best processes and integrating them into the new products of other organizations is the ultimate goal of CMMI. The levels of CMMI are depicted as follows.

CMMI heeds about three areas:

1. Product and Service Development
2. Service Establishment and Management
3. Product and Service Acquisition

The products are the functional units to be designed and the services are functionalities of such products. Integrating numerous individual services into a product is the prime accomplishment of CMMI. The services are analyzed for resemblance, the level of matching and how well they suit themselves into the new product.

CMMI Levels

The CMMI model was evolved from the software CMM for integrating many different and well-organized models into new models. There are five levels of model that are evolutionary.

1. Initial
2. Repeatable
3. Defined
4. Managed
5. Optimizing

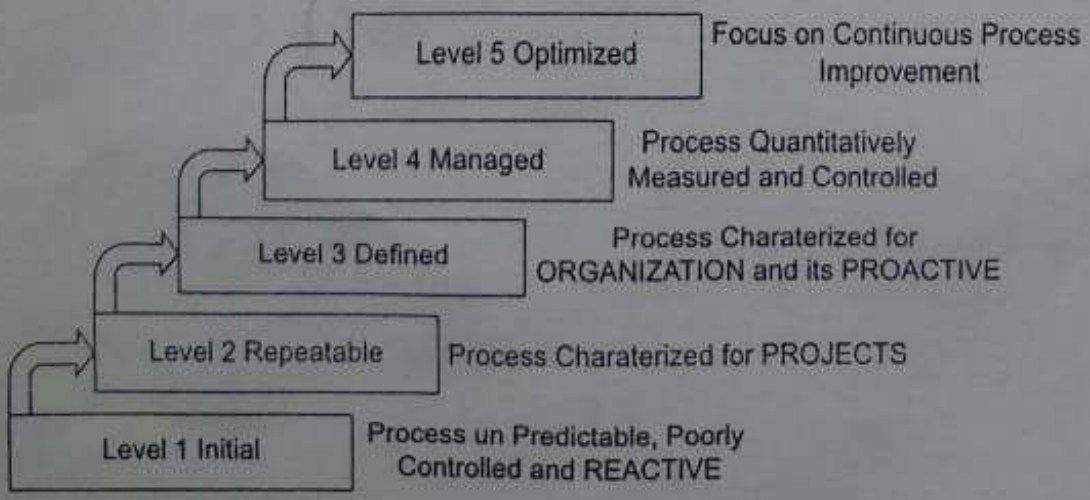


Figure 1.14 CMMI Levels

The 'initial' level is a poorly controlled phase with no proper designation of the schedule and the resources needed for the development. The primary requirements are analyzed, selected and advanced to the next level of the model. In the succeeding level, certain processes are chosen to produce the desired outcomes after organizing the schedule and resources needed. Yet those processes are in urge of additional consistency in the 'repeatable' level. The control processes are ordered and in position.

The third level 'defined' requires stringent measures to be followed. The processes are regulated, well understood and thus properly ordered. The consistency of all the processes is achieved in this stage. 'Manage' is the fourth level in which adequate steps are taken to determine the actions to update and upgrade the processes in the product. The level defines measures to widen the functionality of every process. This level forwards its outcome to the 'optimizing' level which concentrates on smoothening of approaches for better, faster and risk-free execution of the processes.

All these levels are proposed to achieve a process incremental model helping the development team to nominate an efficient product with minimal effort.

People Capability Maturity Model (PCMM) is another model of capability maturity model released in 1995 (book form in 2001) and it deals with the improvement of human resources (assets) of the organization. The people management process areas addressed in PCMM contain the common features and key practices.

1.2.7 Process Pattern

There has been a standard order of activities for completing a task. The order may not be altered, as the pattern has been proven as the most optimal solution. There will not be another alternative with an equal strength as an existing pattern. Patterns compile a set of activities or tasks or actions, following a prescribed sequence of execution. These patterns are considered in software lifecycles, customer communication, coding, reviews and test patterns.

1.2.8 Process Assessments

Every process has its own strength, weakness and associated risks. These factors have to be discovered to determine the level of applicability. With high risks and weaknesses a process cannot be applied to a product. The strength of all possible affirmative solutions has to be resolved to decide on the applicable solution and on the first alternative. Evaluation of the strengths, weaknesses and risks of every process against a process model is called as Process Assessment.

Process assessment describes the results of evaluation as

- Incomplete-Still needs revision and modifications
- Performed-Complete definition and satisfies the customer
- Managed-Controlled and maintained
- Established-Standard procedure applied
- Predictable-Defined with limits of execution
- Optimizing-Slight alterations required

1.3 SOFTWARE PROCESS MODELS

Software process models are systematic methods for controlling and coordinating the development of a software product achieving all the stated objectives. Methodical processes are followed from the initial requirements analysis process to maintenance phase. The systematic processes provide guidelines and prevent the development team deviating from the intended path leading to a failure.

Software process models help the developer team to recognize the following:

1. Set of tasks, subtasks and interfaces
2. Resources needed for every process
3. Adequate time span

1.3.1 Traditional Software Process Models

Traditional approaches of software process do not include the improved and simplified procedures attained today. Linear sequential model is already discussed in Section 1.2.1. Some of the traditional approaches are explained below.

1.3.1.1 Waterfall Model

Waterfall model (refer Figure 1.15) is the first and foremost methodology of Software Development Life Cycle (SDLC). The original methodology consisted of the following activities: Requirements Specification (phase 1), Design (phase 2), Construction (phase 3), Integration (phase 4), Testing (phase 5), Debugging (phase 6), Installation (phase 7) and Maintenance (phase 8). These activities proceed one after the other (like a waterfall). This conceptual model is based on the construction and infrastructure domain. The process model of construction activity is to design a blueprint, collect all the raw materials for construction, organize the workers and determine the time needed for the project to be completed.

The waterfall model was devised by Royce in 1970, and many organizations implemented this process model in earlier days. The requirements and conditions are obtained from the customer in the first step. This specification and analysis is obviously the first process that leads the development team. The development team will start designing a framework for achieving the objectives. The design of a software product is like a blueprint of a construction site. Following the efficient design, the coding begins. This resembles a construction with walls. Testing is the phase that ensures correctness of the product being developed. Without testing, no product will be allowed to be implemented in the real-world environments. The maintenance process cannot be predicted and assumed. The software needs additional features after deployment based on the demand from users and customers.

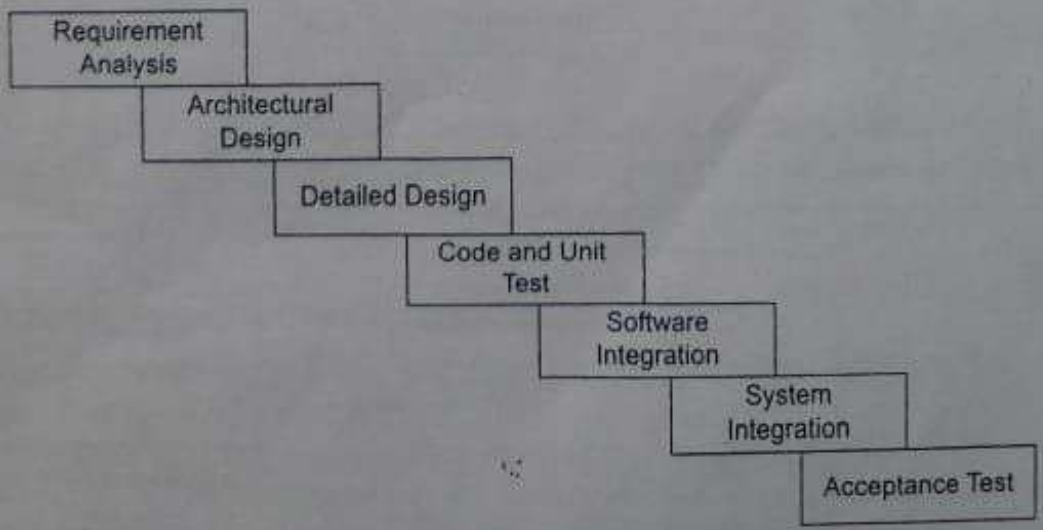


Figure 1.15 Waterfall Model

All the processes of the waterfall model ensue in sequential order. These processes are documented, and the confirmed information are printed and stored for future references. This model is also called as document-driven approach.

Advantages of Waterfall Model

1. Easy to understand
2. Easy for implementation
3. Widely used and known; hence, all stakeholders understand this
4. Identifies deliverables and milestones upfront

Disadvantages of Waterfall Model

1. Does not match well with reality
2. It is unrealistic to freeze the requirement upfront
3. Software is delivered only at the last phase
4. Difficult and expensive to make changes (CRs always)

1.3.1.2 Incremental Process Model

The incremental model is one of the evolutionary models, which overcame the drawback of the linear execution style of the waterfall model (refer Figure 1.16). Incremental model has the capability of multiple streams being progressed independently and simultaneously. Once the requirements are gathered, separate teams can work individually to produce a product assigned. The whole product is achieved by integrating all the individual components developed.

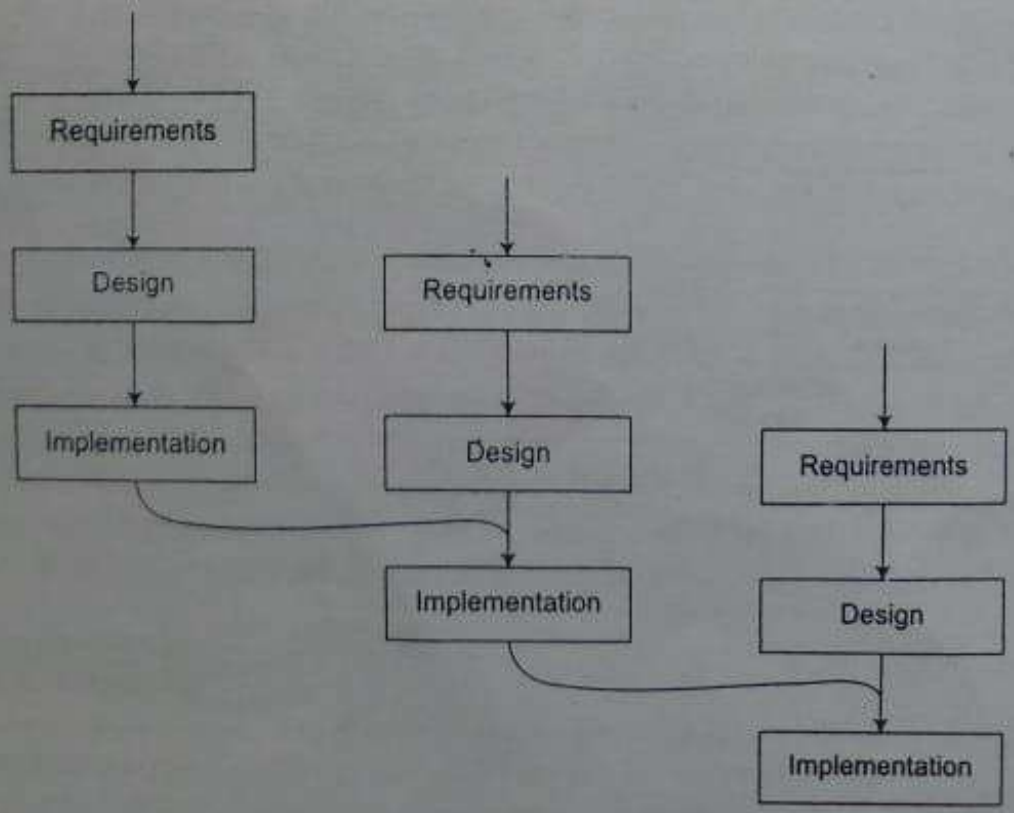


Figure 1.16 Incremental Model

The major advantage of the incremental model is that the consecutive steps do not have to wait until the preceding function completes its activity. This assists the team of developers to complete the design process much faster. The large and complex problem can be divided into a number of smaller and unique tasks, providing an easier chance to solve. Incremental model is suitable for applications that need additions and alterations in its contents after certain time period. For example, the shopping mall application needs to introduce new offers and concessions on the price list of all items for festive moments. They cannot be stable at all times and the waterfall model is not the right method to develop the product. The constraints on time and expenses justified the need of the incremental model at once. The already existing product can be combined with a new requirement by means of the integrator. Incremental model also facilitates the isolation of errors and faulty components at a faster rate, the modules being independent and capable of running without the support of other modules. Faulty modules remain intact until rectified.

Versions and higher configurations of an existing product can be developed with incremental process model. Links would have been established to permit the linkage of additional features in a next version. The links are the functional spots of the integrator component, which builds up a communication link between the versions.

Advantages of the Model

1. Quick feedback loop from business stakeholders
2. Focus on customer feedback
3. Customer feels the product early

Disadvantages of the Model

1. Scheduling of development is difficult with incremental model
2. Although the final product is given in pieces, tracking and monitoring is difficult
3. Testing needs to be exhaustive for each part

1.3.2 Evolutionary Process Models

The Evolutionary process models smoothen the progress of a software development in a number of ways. These strategies are like - creating prototypes and making steps overlapping and autonomous. The demerits of linear sequential models are eliminated and these models stand among the best process models in software development. Some of the evolutionary process models are described as follows:

- Spiral Model
- Components-based Development Model
- Formal Methods Model

1.3.2.1 Spiral Model

The spiral model was proposed by Boehm as an evolutionary model; it introduced the concept of risk analysis in the earlier stages of development process. Spiral model is also called Risk Spiral Model. This model combines the software development life cycle with Risk Management principles to control the risks at early stage of the project.

The previously mentioned methodologies did care about the detection of risks, but only at the near end of the project that increased their burden of rework. If a risk was found to be too disastrous, then

the product had to be designed anew. Taking the importance of a risk management process and testing process into account, the spiral model defined new strategies.

A spiral model encompasses the iterative steps in a spiral representation as shown in Figure 1.17. Each circle in the model denotes a succeeding level of the previous state. These are divided task regions, which represent the selective process. The task regions may vary from the complexity level of different products.

The planning part includes the tasks for designing an efficient framework as a solution to the objectives. Planning also happens in steps: Life Cycle Plan is prepared first; Development Plan is prepared in the next level; and Integration Plan is in the last level.

In the Risk Analysis Phase the prototype is being developed. The risk analysis would report the identification and effects of the present risks in the prototype. Assessments of risks at every rotation of the spiral evolution reduce the problems and the time required for solving at the final stage. The risk analysis strategies have to be stringent enough such that even small errors are not left out. Prototyping, incremental and premature risks detection proves this to be an efficient method over the other process models. Elimination of risks occurs in next iteration. The feedback from the customers is noted for updating the next prototype. The final prototype is developed after stated suggestions are subjected to risk analysis process.

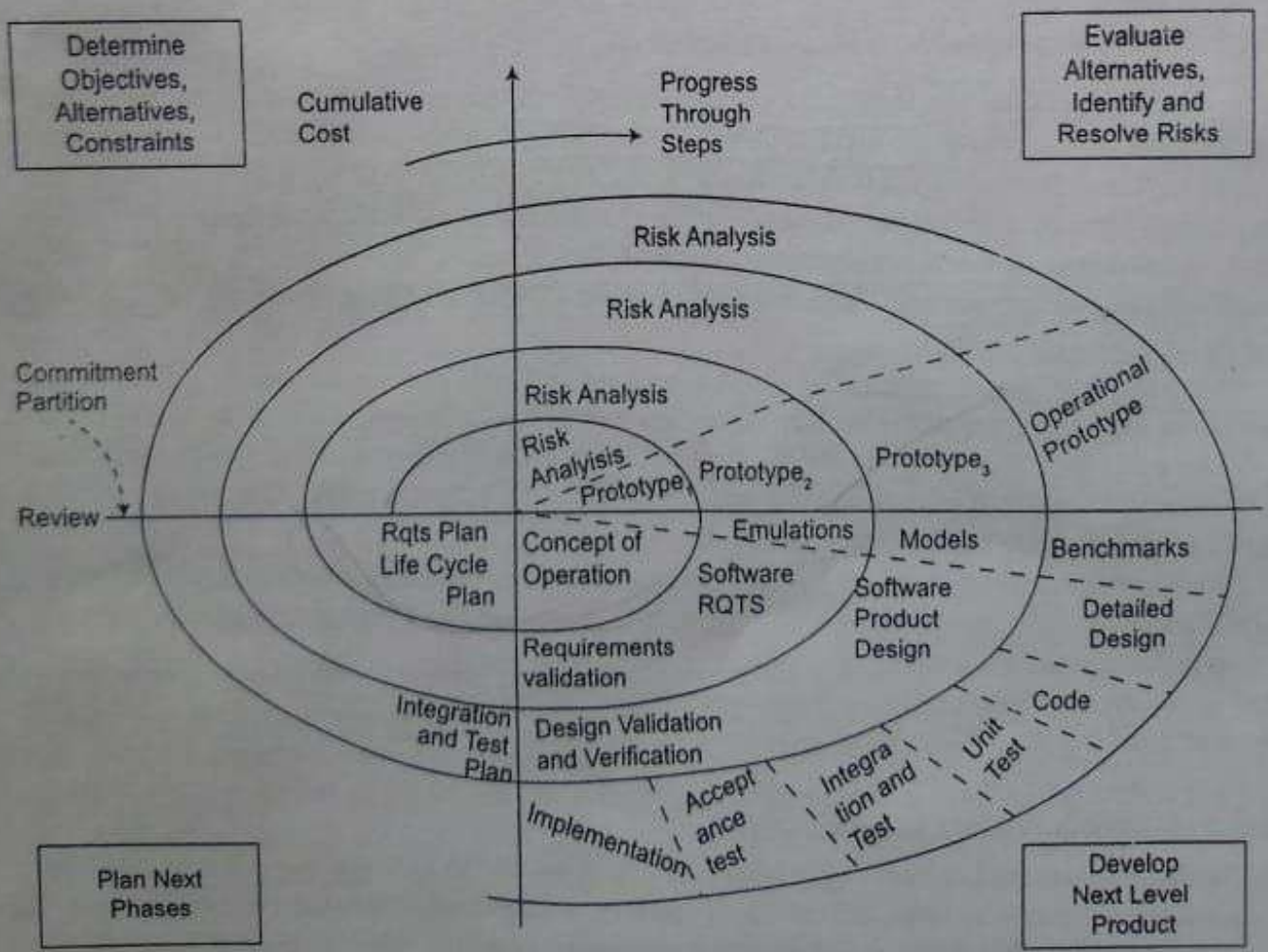


Figure 1.17 Spiral Model

Construction and release also happens in iterative fashion based on the base prototype developed. At every level it is also being strengthened and the final level is the final code to be delivered to the customer.

The number of iterations is determined by the team manager depending upon the rectifications made and completion of the product. The cost and schedule cannot be assumed right at the beginning phase as the prototypes are altered at every rotation. The team manager adjusts cost and time metrics after every circle. The spiral model is selected for software of higher level organizations.

POINTS TO PONDER

A spiral model encompasses the iterative steps in a spiral representation. Each circle in the model denotes a succeeding level of the previous state. There are divided task regions, which represent the selective process.

WIN–WIN Spiral Model

The results are based on the negotiations made by the customers. WIN–WIN result is decided for best negotiations. The customers have their own requirements and needs about the system or product. The customer wins by getting the system or product that satisfies the majority of the customer's needs. The developer has some criteria and budgets given by their organization and the developer wins by working to realistic and achievable budgets and deadlines. Boehm's WIN–WIN spiral model defines a set of negotiation activities at the beginning of each pass around the spiral.

The activities of the system are decided by identifying the key stakeholders and the win condition for the system. Negotiation of the stakeholders' wins conditions are reconciled into a set of winwin conditions for all concerned.

The WIN–WIN spiral model introduces three process milestones, called anchor points. These points help to establish the completion of one cycle around the spiral and provide decisions before the software project initiates its forthcoming cycle.

The anchor points represent three different views of progress

1. Life Cycle Objectives (LCO): It defines a set of objectives for each major software engineering activity.
2. Life Cycle Architecture (LCA): It establishes goal that must be met as the system and software architecture is defined.
3. Initial Operational Capability (IOC): It represents a set of objectives associated with the preparation of the software for installation/distribution, site preparation prior to installation and assistance required by all parties that will use or support the software.

1.3.3 Component-based Development Model

This development model is one of the evolutionary practices for developing a software product. Component-based development (refer Figure 1.18) introduced the concept of reusability to the full extreme. After analysis of the requirements, instead of entering into the design phase, components or modules of the existing similar products are checked for their match. Software reuse is the main motto of this evolutionary model. This model becomes the best in faster and cheaper deployment of the desired product.

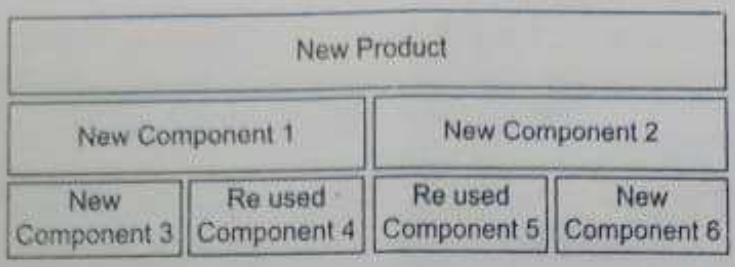


Figure 1.18 Component-based Model

1.3.3.1 Requirements Comparison

The requirements needed for a new product are compared with the recorded list of requirements from the existing products. The percentage of match between the new and existing requirements is evaluated. The matching level should be considerably high enough to be a perfect match.

1.3.3.2 Modification and Implementation

After a suitable match or a near-matching component is perceived, they are implemented in its place. But the question is whether a 100% match can be obtained or not. The result is a NO in most cases. No component can be directly implemented into a new product. The near-matching product needs some modifications to meet the stated requirements. Interfacing needs to be perfect to make the reusable component adapt to a new environment. Modifications and design of interfaces are comparatively easier than designing a whole new component.

1.3.3.3 Risk Analysis

Integration of existing modules has a greater level of risk than the newly developed components. The outcome of the product may be similar but the preceding and succeeding activities may disturb the new component. Hence, a more sincere risk analysis methodology is required for eradication of the risks.

The cost, manpower and time can be reduced to a great extent by using this evolutionary process model. Prototyping and spiral model enhanced the speed of developing a product by its means, and thus component-based software engineering.

1.3.4 The Formal Methods Model

There are some methodologies with strong technical and mathematical background for eliminating most of the errors missed out by other paradigms. Applying mathematical constants improves the rate of detecting errors relating to ambiguity, inconsistency and irregularity. The errors are quantified with notations and standards of mathematical expressions. Although this approach ensures an error-free software product, many other reasons prevent the usage of Formal Methods Model.

This model is

- Time consuming
- Training is needed for proper usage of mathematical expressions
- Customers and end users are mostly unfamiliar with the strategy

Knowledge on the mathematical implications is absolute in this case for detecting an error-free software product, whereas most products do not need such efficient methodologies considering the extra cost and time factors.

1.3.5 Software Engineering Process

Engineering process defines a dedicated and disciplined order of activities with a clear-cut view on how to reach the destination. A software product of high significance needs a pertinent roadmap leading to the objectives. Deriving such an efficient path toward the destination can be achieved only if the order is obeyed completely. The engineering process has gained expertise in many of today's applications and products. The engineering process, unlike other problem-solving strategies and design approaches, orders activities with respect to profound technical information of the product statement.

Software engineering process differs from the ordinary process models in the following means. Normal problem-solving strategies define and solve the problem by verification of the proposed solution, whereas in the engineering processes, the software requirements are analyzed and specified followed by designing and implementating a suitable product and, finally, verifying the designed product with respect to the specified requirements. Efficiency is promoted as the prime factors are narrowed down and the knowledge on the processes is imparted to all employees and participants. This knowledge is acquired by working on the previous successful projects and exposure to the technical background.

1.3.5.1 Primary Steps of Software Engineering Process

As mentioned, the initial step is to 'Analyze' and 'Specify'; the requirements are analyzed for general aspects, such as identifying the main objectives of the entire product and then the associated objectives of next importance and other indirect activities that support the objectives are analyzed, ensuring problem identification and determination of the functional and nonfunctional requirements. These functional and nonfunctional requirements are the resources, which have a direct or indirect impact on the outcome of the product. These analyses produce a list of specifications on the requirements and the identified problem.

'Designing and Implementation' of a well-organized technical plan would deduce the number of designs to meet the goals of the product. Once the design has been proved as an efficient solution, it can be developed as a prototype for examination by the customer for any changes. The prototype eliminates the chances of later delays if properly verified. A prototype is extended with full functionalities and implemented in the final product.

'Verifying' the deployed product involves the cross-checking of how well the whole product works in the real-time environment. Usually, the product is developed and enhanced by introducing additional features when the demands increase. The data design and the flow design are separately verified to ensure maximum efficiency. Functional design verifies the relationships among the existing processes and transfer of data in between the available functions without any conflicts.

1.3.6 Business Process Engineering

Software products are employed in almost every field, mostly in business applications. They are implemented in business areas and are never constant in backdrop as the customers and end users need features of global influence. Distributed environments of a same business process cannot be the same. Operational responsiveness is the ability of a software product to respond to different queries

from the customers. The serviceable business processes are intended to respond, provide solutions regarding the business outsources and purpose. The detailed information on the available services to the customers is promised to every requester with 100% satisfaction.

Operational responsiveness advocates the principles, intentions and functional aspects of a particular organization. The organization develops a software product with its business processes employed. The software developed concentrates on the devoted business processes unlike other software solutions.

1.3.7 Fourth-generation Techniques

The Fourth-Generation Techniques (4GT) combines distinguished directory of software tools with assisting amenities for a software developer. Nonprocedural languages for databases maintenance and data manipulation operations, high end graphics interfacing, spreadsheet operations, drag and drop creations of web-based applications are some of the high-level 4G technologies. Imagine a whole system implementing all of these separate techniques into a single procedure. 4GT is designed to generate the source code by themselves with minimal specification from the designer. The developer does not put any extra effort other than inputting the complete requirements and other conditions. Positioning the preconditions, post conditions and the flow conditions at the correct location would be enough; the rest of the coding is taken care automatically by the procedure itself. After mentioning such conditions a nonoperational prototype is developed as a sample output. The foremost point is that the customer should be able to understand every functionality and error-prone zones in the prototype. This procedure aids in faster development and deployment of a software product with greater interactive sessions and participation of a customer. As all the processes eliminate the abstract terminologies, the entire process becomes completely visible from architecture to coding. The major drawback is that a customer himself may lack the optimal way of describing the requirements, and unwanted processes may be wasting the efficiency of the 4GT.

This technique includes the following processes:

- Requirements Specification
- Prototype Design
- Implementation
- Constructing the Original Framework

1.4 SOFTWARE PRODUCT

1.4.1 Characteristics of A Good Software Product

A good software product should possess the following characteristics:

Completeness–The developed software products should cover all the stated and agreed requirements without missing out any.

Consistency–Operations of the product should be stable and capable of handling the problems in implementation.

Durability–Customers may vary in technology requirement or geographical aspects, and the product should respond to a diversity of environments.

Efficiency–The product should not waste the resources, execution and waiting time of operations and the memory spaces allocated for the processes.

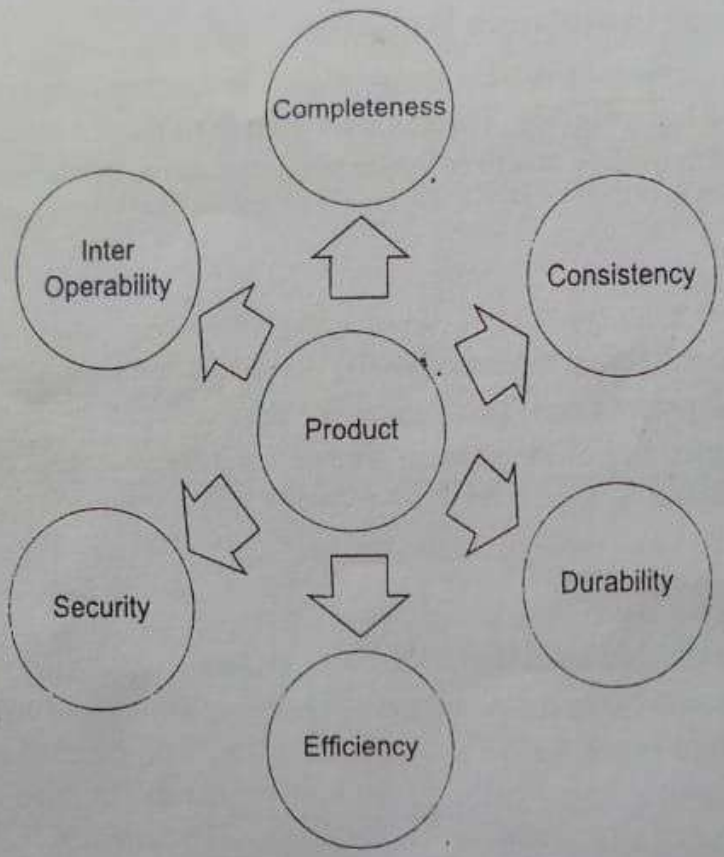


Figure 1.19 Product Characteristics

Security–The stored contents and the input statements of the software should be confidential and conserved if they possess a high significance.

Interoperability–Communication between other processes and environments should be enabled to apply a product universally.

1.4.2 Engineering Aspects of Software Product

A software product has different views of understanding. For technical and engineering aspects of software product, the customers see an abstract image of the product. Engineering aspects contemplate the description of a product in terms of its practical implications and has an impact over the platform. The technical aspects include the following:

1.4.2.1 Purpose

The aim of a software product needs a strong justification and thus represents the urge of being developed. The areas the product is supposed to cover, the necessary conditions for the product to work and the dependability on other processes are the primary aspects considered.

1.4.2.2 Process

Definition of the operations needs engineering skills for the right placement and interrelationships. The experience of the developers in previous projects aids them for better design approaches. Unlike normal methodologies, engineering process needs strong background on the construction ideologies.

1.4.3 Role of Management in Software Products

Management plays an important role in driving the processes of development in the right path benefitting the customer and the organization. The ultimate goal is to limit the budget and achieve a reasonable profit satisfying the quality standards. Managing the development process includes the following steps.

1.4.3.1 Defining the Vision

The goal of the organization is different from that of the objectives of the product. The vision is the force that motivates the development team. This defines the standards and principles of the organization apart from the desired outcomes of the product. The vision keeps the team on track to maintain the standards under any condition.

1.4.3.2 Defining the Plan

The plans are the operations derived with inputs and desires on the outputs. Plan is the blueprint, which has to be followed for achieving the outcomes. The plan encompasses many individual processes, structured in an ordered sequence. This structure defined is an optimal solution for perfect execution of operations. Without a proper plan, the solution cannot be obtained as promised. There are also additional plans designed as a backup in case of failures. The alternative plans are also numbered as the first alternative and so on. Secondary options are initiated only if the optimal plan fails to succeed.

1.4.3.3 Defining the Designation

Every participating member in a development process is assigned with a role and jobs to perform. The leader of the team has supplementary duties to ensure that every activity is performed as scheduled and planned. Every member follows the vision and principles of the organization besides their prescribed role. The members are categorized according to their authority level and decision-making capability. But all members have the liberty to express their views and ideas over the tasks.

1.4.3.4 Testing the Pathway

Testing is an important process in managing the defined plans. Testing includes the verification of the correctness and consistency of the product, checking on whether the resources are properly used, and whether budget and time constraints are met. Thus testing process ensures that the development team proceeds in the destined path as defined. This reports the deviation from the defined pathway, if the results are different from the sample outputs.

1.4.3.5 Supportive Activities

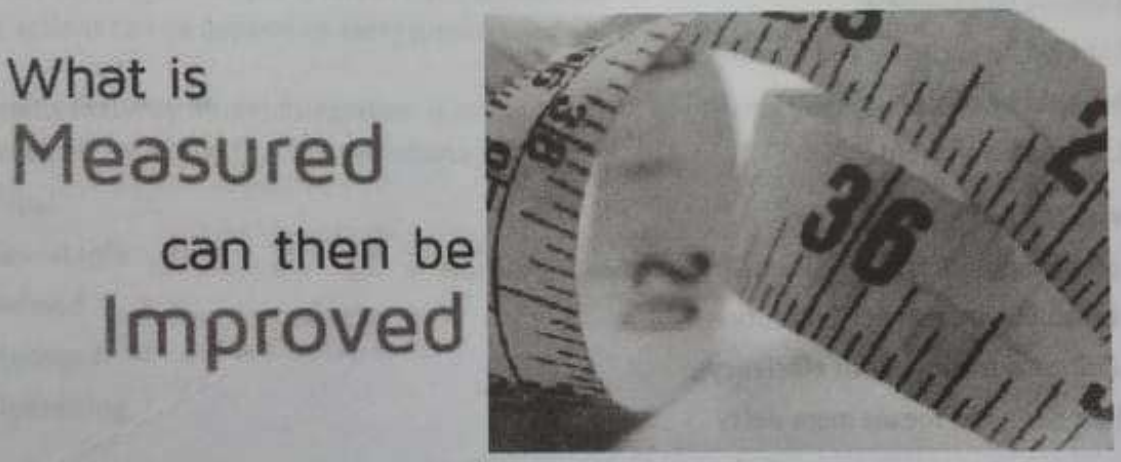
There are many associated processes beyond the normal procedures. Licensing the product as the registered outcome of a particular organization and marketing the product among the competitive organizations are some of the additional and adequate activities, which support the delivery of the software product. End user training has to be provided after delivery to ensure efficient utilization. Not every user can readily use the software product as many may lack the basic knowledge to understand the functionality.

The project management concepts are discussed in the following chapters:

Chapter 13–Project Management Introduction
Chapter 14–Risk Analysis and Management
Chapter 15–Communication and Team Management
Chapter 16–Project Time and Cost Management
Chapter 17–Project Stakeholder Management

1.4.4 Role of Metrics and Measurements

The terms representing the quality and quantity of a particular process are metrics and measures. They are used in determining the level of any means in the form of numbers or units. Understanding these terms needs the definition of data points. These are units that denote the condition of a process, the variables involved and the errors in a particular spot.



A 'Measure' is when the data point values are noted. A 'Measurement' is the representation of data points after review of a whole module or processes collectively. Measurements are of two types, either direct or indirect. Direct measurements denote the values after quantifying the errors in lines of code, resources misuse and memory misuses. Indirect measurements estimate the errors in functionality, complexity, reliability and maintainability.

'Metrics' are quantitative details obtained by monitoring and controlling the budget and time span of the product. Any deviations from the scheduled path would be reported by the metric values observed at regular intervals.

Software measurement and metrics are discussed in Chapter 9 Introduction to Software Measurement and Metrics and LOC Function Point and Object Oriented Metrics in Chapter 10.

1.4.5 Product Engineering Overview

Global implementation of software products has forced the introduction of many innovative techniques for delivering a high-quality product within a short duration and with low expenses. This urge had driven the methodologies toward simpler and faster completion motives. Agile methodology, Pair Programming, Extreme Programming and Reusability are some of the concepts that enabled today's goals. Product engineering methodology groups all these strategies and the resultant method has overcome all the drawbacks of traditional methodologies. This engineering model is purely dedicated to the development of products, differentiating it from software or process engineering concepts.

2

Requirements Engineering Principles

CHAPTER COVERAGE

2.1 INTRODUCTION

As discussed in the previous chapter, in software development, the software project life cycle starts with capturing the requirements of the project. Developing a product or executing a project is nothing but catering and taking the requirements to the next level. Gathering, documenting, disseminating and managing the requirements is the key to success of any project. Although it sounds easy, gathering the requirements effectively and managing them efficiently is a complex task and needs to be handled in a systematic manner. Requirements engineering principles help to do this task in a better way.

According to industry average, more than 90% of the software projects fail due to faulty or incomplete requirements capturing. This certainly gives an idea of how important this step is for the success of any software project. In this chapter, we will cover the different aspects of requirements engineering with steps and guidelines.

2.2 WHAT IS REQUIREMENTS ENGINEERING?

Before getting into the details of requirements engineering, let us understand what "requirement" is.

According to IEEE Standard 610.12-1990, requirement is defined as "a condition or capability needed by a user to solve a problem or achieve an objective" or "a condition or capability that must be met or processed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents."

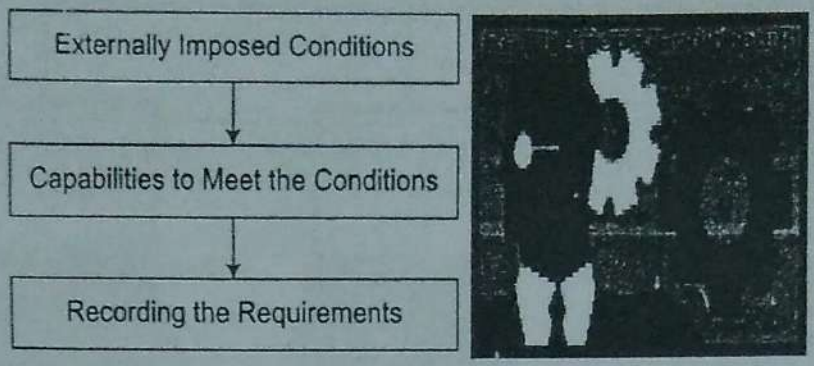


Figure 2.1 Requirements Engineering

This means that the requirements originate from the user's needs and are then captured and tracked to satisfy a contract or a specification.

Now, let us look into the definition of "requirements engineering."

"Requirements engineering is the discipline that explores the externally imposed conditions on a proposed computer system and tries to identify the capabilities that will meet those imposed conditions and recording the same in the form of documentation called the requirement document of the computer system."

The whole gamut of activities related to requirements from inception to understanding, tracking and maintaining the requirements is termed requirements engineering (Figure 2.1).

2.3 IMPORTANCE OF REQUIREMENTS

Requirements are the stepping stones to the success of any project. If software projects get started without properly understanding the user's needs or without exploring the multiple dimensions of the requirements, there will be misalignment at the end between the final result delivered and the user's expectations of the project resulting in a lot of rework.

In software development, primary focus is usually given to the construction phase, which leads to a lot of problems at the end.

A software project has to be completed within a specified time frame and budget. However, in some cases, rework has to be done due to incomplete requirements understanding, which is the primary cause of schedule and budget overruns.

Figure 2.2 shows the relative cost of repairing a defect at each stage of the software project life cycle. It is evident that the earlier the error is detected, the easier and cheaper it is to fix the error.

For example, if a defect in the requirements is found at the requirements review stage, then it will cost less (e.g., $x) to fix the issue, but as this faulty requirement percolates through the later stages it would cost more to fix the same issue: 3 times the cost to fix it during the design stage, 5 to 10 times the cost to fix it during the construction stage, 10 times the cost to fix it during the system testing stage of the project and 10 to 100 times the cost to fix it during the postrelease phase. Therefore, identifying and fixing errors at an early stage will cost less to the organization.

This signifies that capturing the requirements completely and correctly at the early stages of the project life cycle is essential to keep the project within the budget so that it is profitable. The only way to ensure this is to follow a disciplined requirements engineering process.

Cost correct an Error		Stage detected				
		Requirements	Design	Construction	System Test	Post release
Stage introduced	Requirements	1X	3X	5-10X	10X	10-100X
	Design	-	1X	10X	15X	25-100X
	Construction	-		1X	10X	10-25X

Figure 2.2 Cost of Fixing Errors at Different Stages

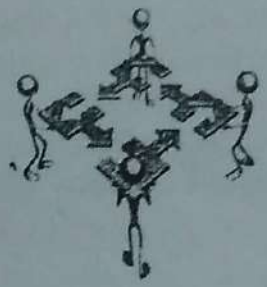
POINTS TO PONDER

The cost of fixing an error in the system increases exponentially with the stage of the project.

2.4 TYPES OF REQUIREMENTS

Requirements can be categorized into three main types: functional requirements, nonfunctional requirements, and interface specification.

Types of Requirements
- Functional Requirements
- Non Functional Requirements
- Interface Specification



2.4.1 Functional (User) Requirements

The set of requirements (Figure 2.3) that defines what the system will do or accomplish is called functional requirements. These requirements determine how the software will behave to meet users' needs.

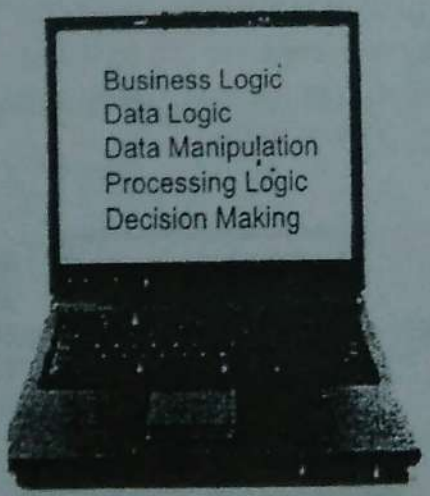


Figure 2.3 Functional Requirements Characteristics

The requirements may be for performing calculations, data manipulation, processing, logical decision-making, etc., which form the basis of business rules. Functional requirements are often captured in use cases. Functional requirements are the main drivers of the application architecture of a system. An example of functional requirement is a retail shop billing software having the functionalities of capturing commodity prices, calculating discounts, generating bills, printing invoices, etc.

2.4.2 Nonfunctional (System) Requirements

Functional requirements are supported by nonfunctional requirements (see Figure 2.4). The quality attributes and the design and architecture constraints that the system must have are called nonfunctional requirements (NFRs).

Some of the quality attributes of the system from an end user's perspective include performance, availability, usability and security and from a developer's perspective include reusability, testability, maintainability and portability.

NFRs as shown in Figure 2.4 are critical in most of the software projects than functional requirements. NFRs cater to the architectural needs of the overall system, whereas functional requirements cater to the design needs of the overall system.

Some examples of NFRs are:

"The system should be available 24 × 7" (Availability)

"The system should save or fetch 1000 records in 1 second" (Performance)

NFRs may be related to the product or the organization or to the external requirements of the system. Product-related NFRs include performance, availability, maintainability, portability, reliability, security, scalability, testability and usability, whereas organization-related NFRs include standards requirements and implementation requirements. External NFRs include legal requirements and ethical requirements.

Measuring NFRs We need to first define measurable criteria for each NFR and then realize the metrics by measuring it. For example, Availability can be first defined in terms of percentage and then define the duration within which we are going to measure it. To measure the availability, we can assume the availability of the system in the past 1 week to be 100% as the system was running continuously without any downtime (another related metrics). A Service Level Agreement is established with the customer for setting an agreed level of availability depending on the stability of the system. If the system is more stable without any downtime, we can fix the agreed limit of availability as 100%. If the system is not stable, then there will be frequent system shut downs and the target availability needs to be reduced accordingly (e.g., 95% or 90%).

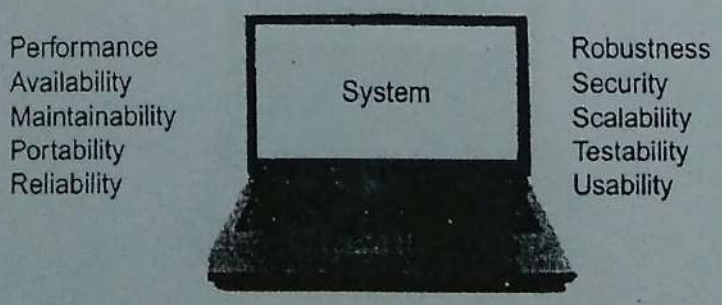


Figure 2.4 Nonfunctional Requirements Characteristics

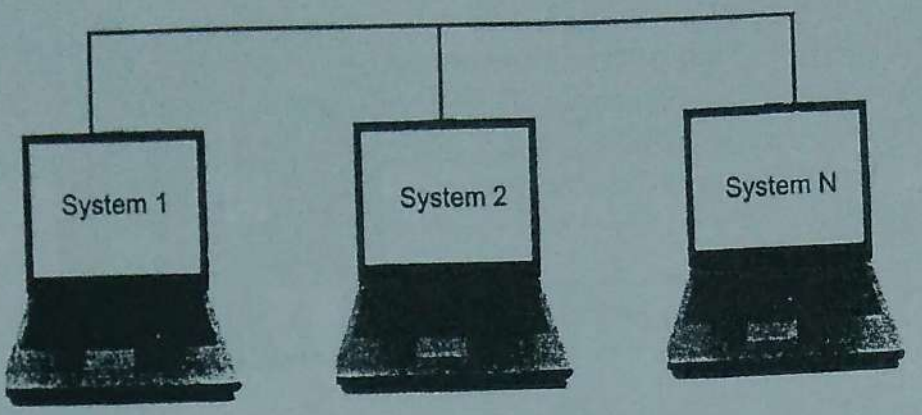


Figure 2.5 Interface Specification Characteristics

Approaches to NFRs NFRs can be approached in two ways- product-oriented and process-oriented approaches. As the names suggest, the product-oriented approach concentrates on the product and its associated NFR aspects. while the process-oriented approach concentrates on the process and its associated NFR aspects. Second, the NFRs can be approached qualitatively and quantitatively. The qualitative approach concentrates on non-numeric-related NFRs, whereas the quantitative approach focuses on the numeric aspects of the NFRs.

2.4.3 Interface Specification

Most of the software systems do not work alone and need to interact with other systems in order to work (Figure 2.5). They interact with many other systems to receive and send data, get help in processing logic, store information in other systems, etc. The interaction requirement of one system with another is defined in the interface specification.

Interface specification helps in building different systems in such a fashion that they can seamlessly interact with each other to produce the desired outcome. Let us consider our previous example of the retail shop billing system. It may interact with another system – an inventory system – to ensure that the warehouse keeps track of the stock of the material at hand. The protocol of how these two systems interact with each other are captured in the interface specification document.

POINTS TO PONDER

Functional requirements define what the system must do to fulfill the user's needs, nonfunctional requirements put forth the constraints on design and architecture, whereas interface requirements define how the system will interact with other external systems.

2.5 STEPS INVOLVED IN REQUIREMENTS ENGINEERING

Although the requirements engineering process may vary based on the application domain, a few generic steps are common across all types of software projects.

Requirements engineering includes requirements development and requirements management (Figures 2.6 and 2.7). The basic aims of the requirements engineering process are to provide a

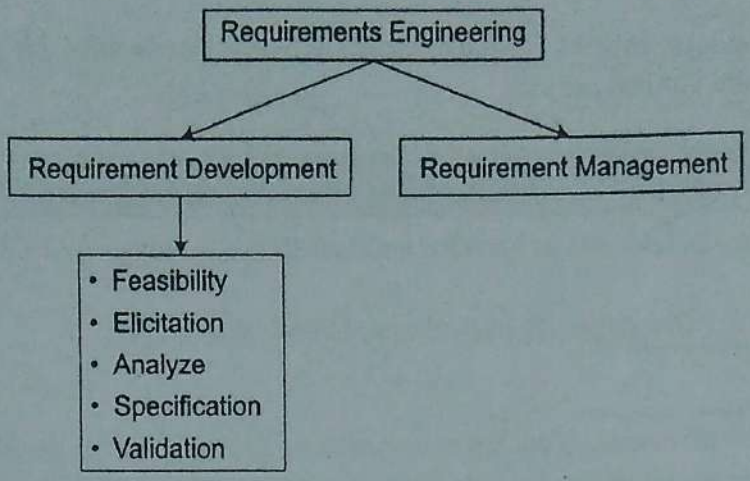


Figure 2.6 Requirements Engineering Steps

mechanism to understand the user's needs, analyze the need, assess the feasibility, specify and validate the requirements and proposed solution, and manage the requirements over the whole life cycle of the project, which are discussed in the following sections:

- Feasibility study
- Requirements elicitation
- Requirements analysis
- Specifying and validating requirements
- Requirements management

Figure 2.7 shows how each step of requirements engineering is related to each other.

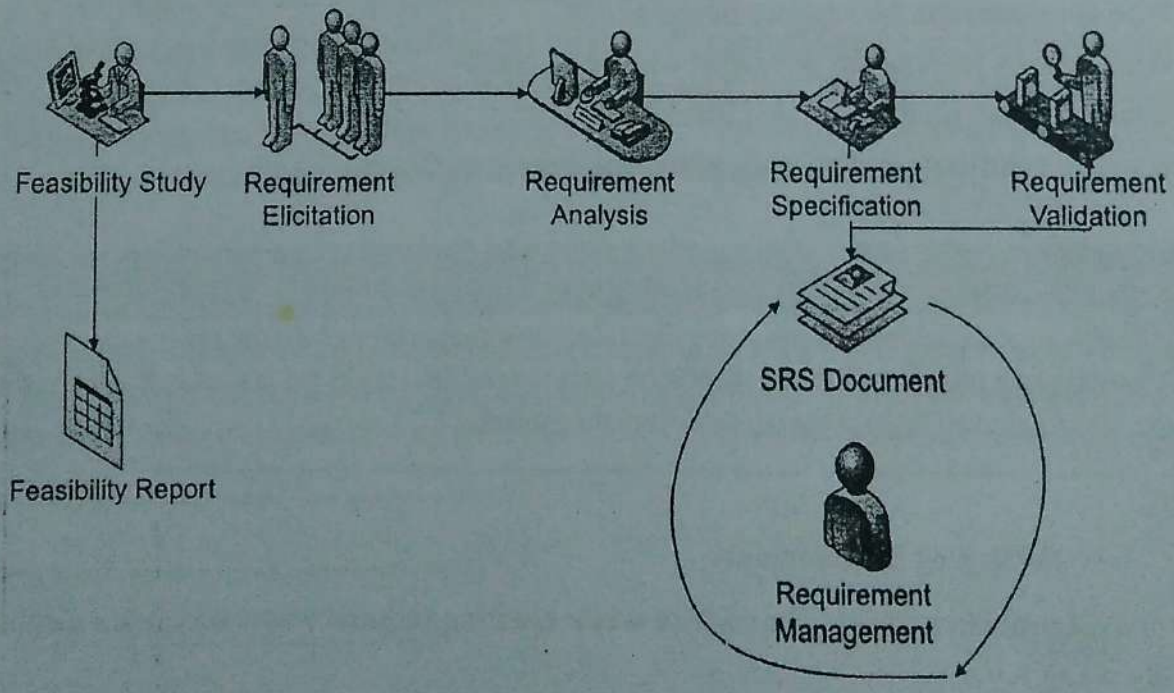


Figure 2.7 Requirements Engineering Process

2.5.1 Feasibility Study

Feasibility study is the first step of the requirements engineering process. The worthiness of the proposed software system should be determined before spending efforts on building the system.

A feasibility study (Figure 2.8) helps in deciding whether the proposed system is worthwhile and has the following characteristics:

- Whether the system contributes to organizational objectives
- Whether the system can be developed using the current available technology and within the specified time and budget
- Whether the system can easily be integrated with other surrounding systems as required by the overall architecture

Thus, feasibility can be classified into three broad categories (Figure 2.9): operational feasibility, technical feasibility and economic feasibility.

Operational feasibility: This checks the usability of the proposed software. If the operational scope is high, then the proposed system will be used more ensuring the acceptance from the sponsors of the project.

Technical feasibility: This checks whether the level of technology required for the development of the system is available with the software firm, including hardware resources, software development platforms and other software tools.

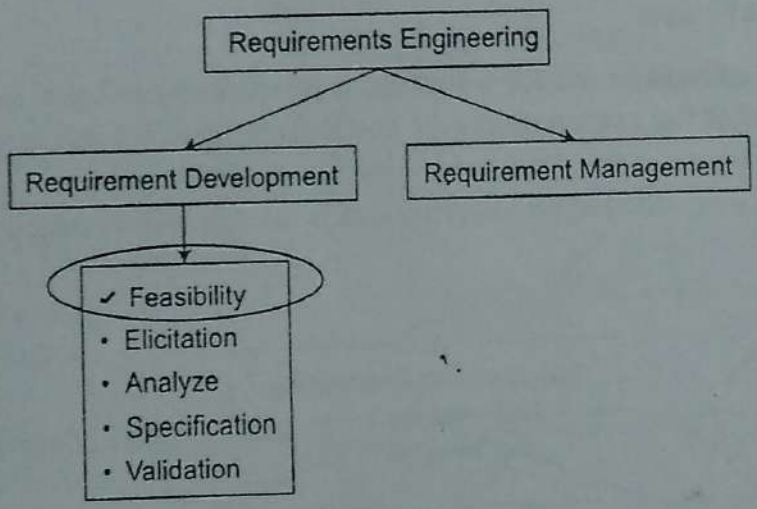


Figure 2.8 Requirements Feasibility

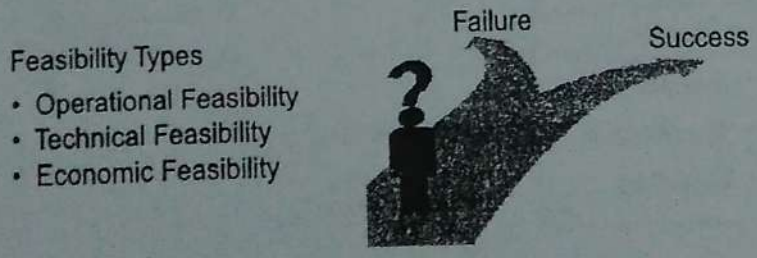


Figure 2.9 Feasibility Types

Economic feasibility: This checks whether there is scope for enough return by investing in this software system. All the costs involved in developing the system, including software licenses, hardware procurement and manpower cost, needs to be considered while doing this analysis. The cost–benefit ratio is derived as a result and based on the justification the stakeholders make a decision of going ahead with the proposed project.

POINTS TO PONDER

Before investing in any project, the sponsors will be keen to know whether the investment is worth and the project can succeed technically as well as economically. Thus, a feasibility study step provides an insight to all stakeholders at the very beginning.

2.5.2 Requirements Elicitation

Requirements elicitation is the second step of the requirements engineering process (Figure 2.10).

In this phase, the source for all the requirements are identified and then by using these sources, the user's needs and all the possible problem statements are identified. Although it looks simple, this phase is often iterative and at the end of each iteration the customer needs may become clearer and new needs may emerge. The stakeholders involved during this phase include end users, managers, development and test engineers, domain experts and business analysts.

2.5.2.1 Identifying Stakeholders

Before the requirements are gathered for a system, it is important to know the people who can contribute and help gather the requirements. If key stakeholders are not identified during the requirements elicitation phase, it can lead to nonidentification of important requirements causing rework at a later stage. Thus, stakeholder identification is the first step in starting the requirements elicitation.

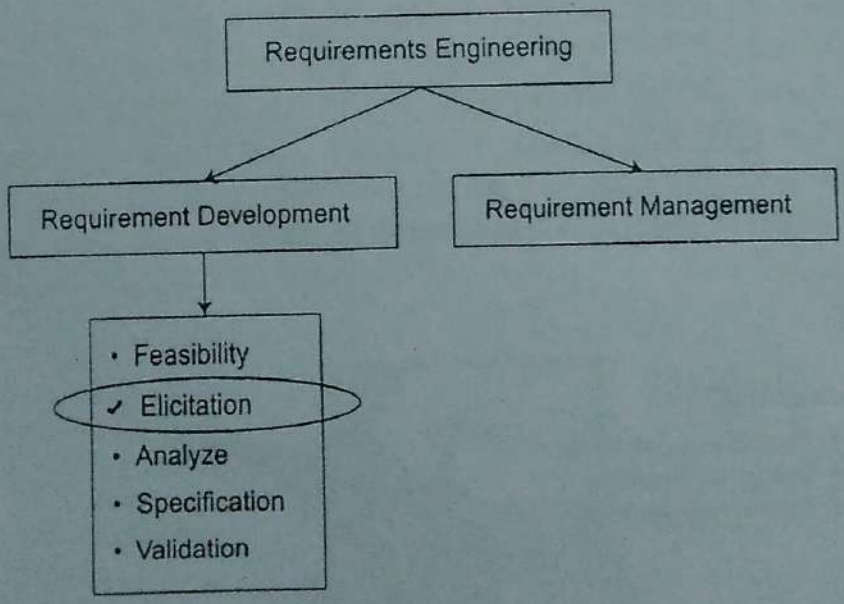


Figure 2.10 Requirements Elicitation

Stakeholders are people or entities (organizations) actively involved in the projects. They are affected by outcomes of the defined project.

Discussion Question

Can you think of the important stakeholders for a medical insurance claim processing software development project?

1. _____
2. _____
3. _____
4. _____

2.5.2.2 Characteristics of Stakeholders

Stakeholder interests may be either positively or negatively impacted by the performance of the project. Stakeholders may have an influence on the project and its results. Thus, it is important for the project manager to identify all the stakeholders and their requirements (sometimes requirements may be implicit and not explicitly stated – the project manager should also try to understand such requirements). Stakeholders may have conflicting interests and objectives; therefore, managing them may not be easy. Involving stakeholders in the project phases improves the probability of success of the project.

Stakeholder involvement is always situation-specific; what works in one situation may not be appropriate in another. Ask a few simple questions to identify appropriate and required stakeholders. Although it is not intended to provide an exhaustive list of questions here, this will give you a fare idea of the type of questions appropriate for this purpose:

1. Who are the people likely to be affected/benefited?
2. Who is responsible for what we intend to do?
3. Who is likely to move for or against what is intended?
4. Who can make what is intended better?
5. Who can contribute to financial and technical resources?

POINTS TO PONDER

Properly identifying stakeholders and managing the stakeholder's expectation hold the key to success of any project. Maintaining a prioritized list of stakeholders according to their say in the project helps in avoiding the bottlenecks created by them at the later stages of the project.

2.5.2.3 Problems in Eliciting Requirements

There are several problems which can surface while eliciting requirements, which are discussed below (Figure 2.11).

- **Users not sure about the requirements:** This situation arises very often when the system to be developed is a completely new system and there is no corresponding manual system or any

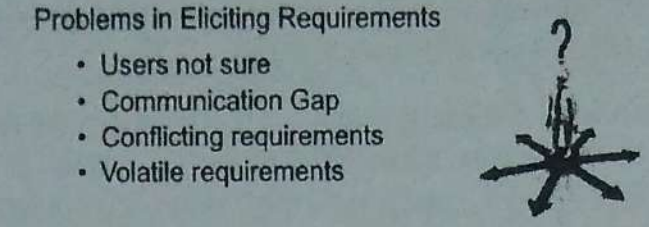


Figure 2.11 Problems in Eliciting Requirements

previous software system that can be analyzed to get the requirement. In this situation, the users will be looking for either similar systems developed elsewhere or mock-up screens, workflows, prototypes, etc. during the requirements gathering stage, which will aid their imagination in coming up with the requirements for the new system.

- **Communication gap:** Even if the end users and stakeholders are clear about the need for developing the new system, they may find it difficult to express it in a concrete manner due to their less exposure to the computing systems. They may state the requirements in an ambiguous or nontestable fashion. Sometimes the obvious basic requirements may be omitted and they concentrate on explaining the peripheral requirements. The system engineer needs to drive the stakeholders in the right direction throughout the requirements gathering stage so that they focus only on the most important requirements.
- **Conflicting requirements:** This problem increases with the number of stakeholders involved in the project. There may be conflicting needs and priorities for each stakeholder based on the business areas they are covering. For example, a stakeholder from the marketing team will try to provide requirements that will expedite the creation of new products, while the end users of the system will look for efficient screens that help easy data capture. During the analysis phase, the requirements analyst and the client should collaboratively discuss and resolve any conflicting requirements from multiple stakeholders.
- **Volatile requirements:** Requirements may get changed during the elicitation stage due to the entry of new stakeholders who have different perspectives of the system. Although this is helpful in clarifying the requirements better, it sometimes creates friction between the requirements engineer and the stakeholders due to continuous change in the scope.

Discussion Questions

Suppose you are the end user of a mainframe-based banking software that is going to be replaced by a Java-based software. The requirements engineers from this new company have approached you to understand the requirements for the new system? How you will ensure that they have noted down all your requirements correctly?

1. _____
2. _____
3. _____
4. _____

2.5.2.4 Requirements Elicitation Techniques

Several techniques (Figure 2.12) can be employed for eliciting the requirements as listed below. Based on the project scope, domain, customer preparedness, etc., one or a combination of techniques needs to be used for gathering the requirements in an organized manner.

- Interviewing
- Focus groups
- Facilitated workshops
- Prototyping
- Questionnaires
- Brainstorming
- Direct observation
- Apprenticing

Interviewing Begin the interviews with the stakeholders who are believed to have complete understanding of the requirements. Face-to-face interactions with users through individual interviewing is the primary source of requirements (Figure 2.13).

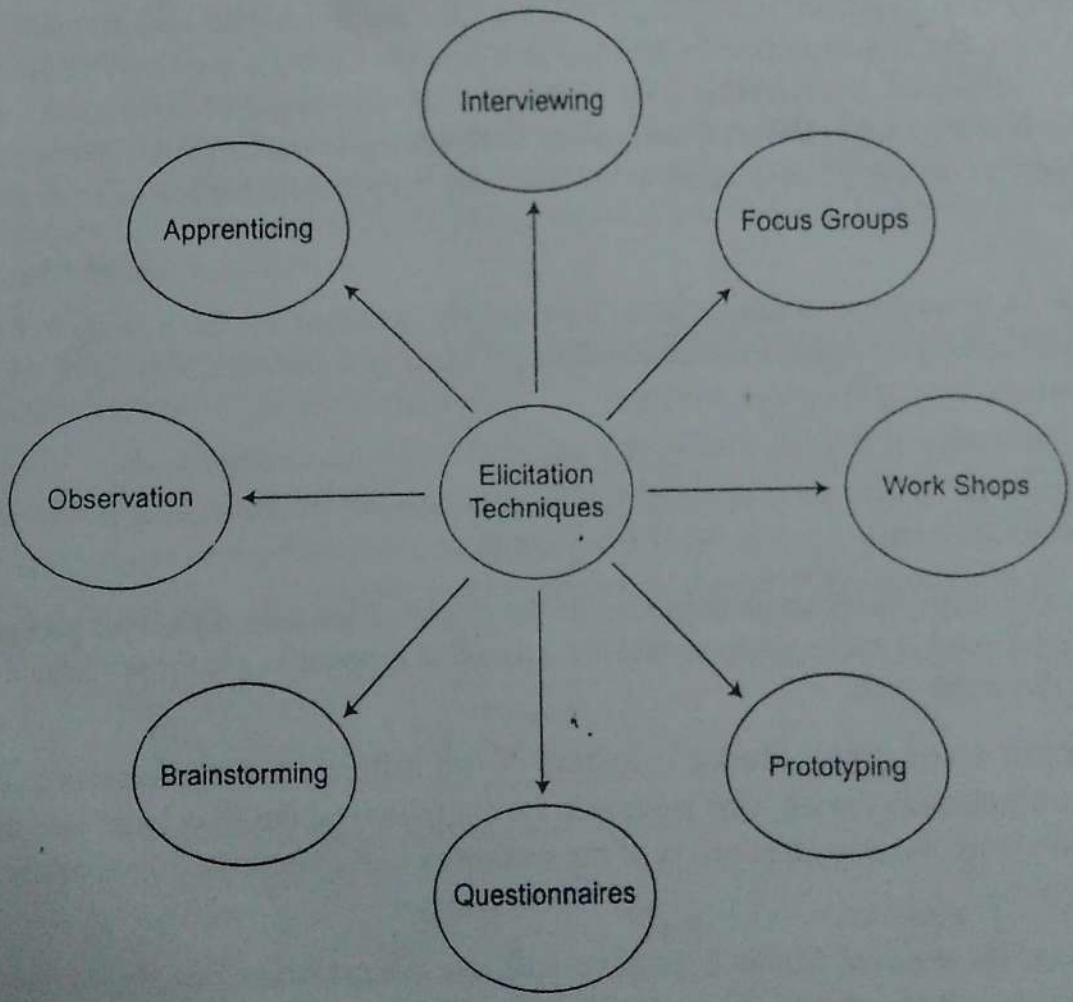


Figure 2.12 Requirements Elicitation Techniques

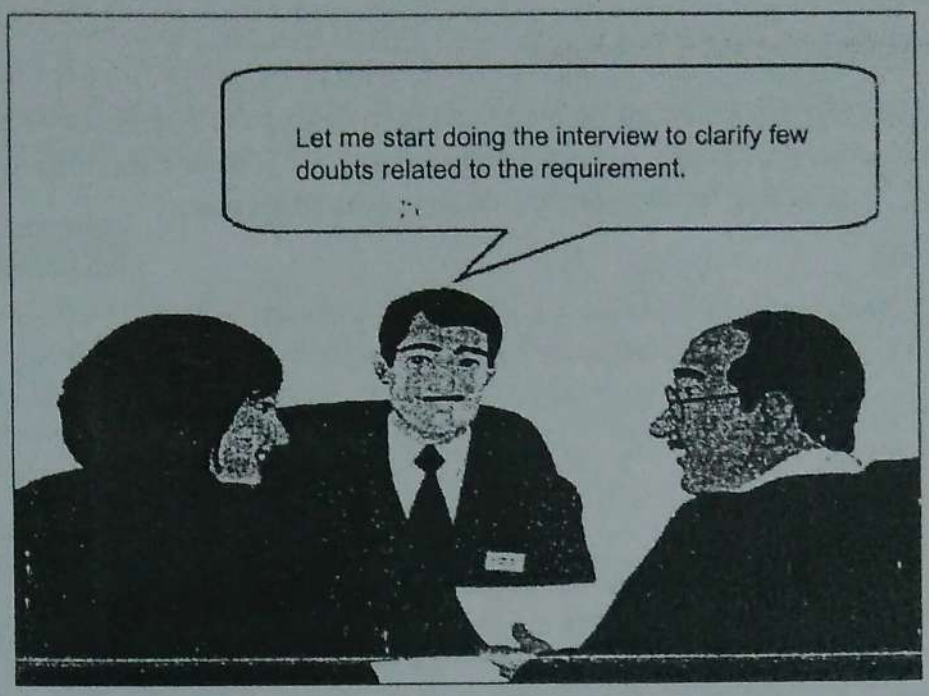


Figure 2.13 Interviewing Technique

Interviewing is an important and effective way to gather and validate the requirements. Interviewing is used when the requirements are detailed and differing opinions are likely or are sought. In distributed agile projects, interviewing is the main technique used to gather requirements. Even if the team and the customer are collocated, we can use interviewing to clarify doubts. During the execution of the project, at any point in time, interviews may happen with the product owner to clarify doubts.

Focus Groups A focus group is similar to the interviewing technique, but has a group of 6 to 10 people at a time. We can get a lot of information during a focus group session because of the enhanced level of discussion among the group members.

Types of Focus Groups There are several types of focus groups that facilitate group discussions, which are discussed below.

Two-way focus group: As the name suggests, two focus groups are used. One focus group will continuously watch the other focus group so that the interaction happens as per the predefined rules and proper discipline is followed.

Dual moderator focus group Model 1: Instead of two different focus groups as in the above model, two moderators do the job. One looks into the discipline and the other looks into the content flow. This will ensure the smooth progress of the session as well as the entire topic of discussion is covered.

Dual moderator focus group Model 2: In this model also two moderators are present but they take completely opposite stands, that is, if one says that something is possible, the other says why it is not possible. This is more helpful in finding the pros and cons of both views and will be helpful to make a final decision.

Respondent moderator focus group: Respondents are asked to act as the moderator temporarily and because of this they take ownership of the session. Hence, we can collect the requirements successfully and the outcome will be helpful (Figure 2.14).

Client participant focus group: Client representatives participate in the discussion and will ensure ownership from the client on the decision taken. Most of the projects fail at the last stage of the project life cycle, particularly in the User Acceptance Testing phase. The reason being the client may not take ownership as the end users have more ownership at that stage. However, client participant focus groups help overcome these kinds of problems.

Mini focus group: Groups are restricted to four or five members rather than 8 to 12 and is helpful to avoid confusion and manage and gather the requirements quickly.

Teleconference focus group: Telephone network is used to facilitate the discussion sessions.

Online focus group: Computers connected via the internet are used to facilitate the discussion sessions.

Facilitated Workshops Workshops (Figure 2.15) can be used for rapidly pulling together a good set of requirements. A workshop is a very quick and best way to gather requirements compared with all other techniques. A workshop is expensive because it involves many people, but it saves a large amount of time. Requirements are discussed at a high level. Workshops are useful in situations where

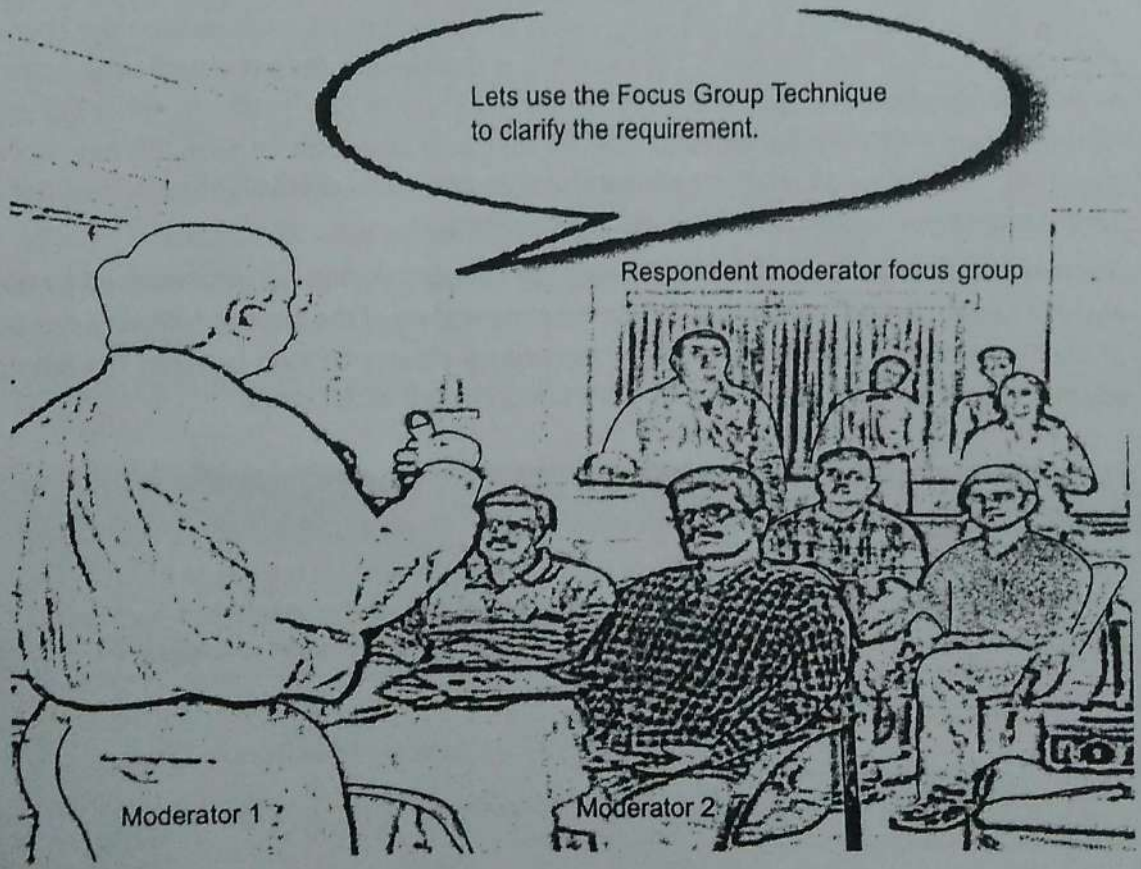


Figure 2.14 Focus Group Technique

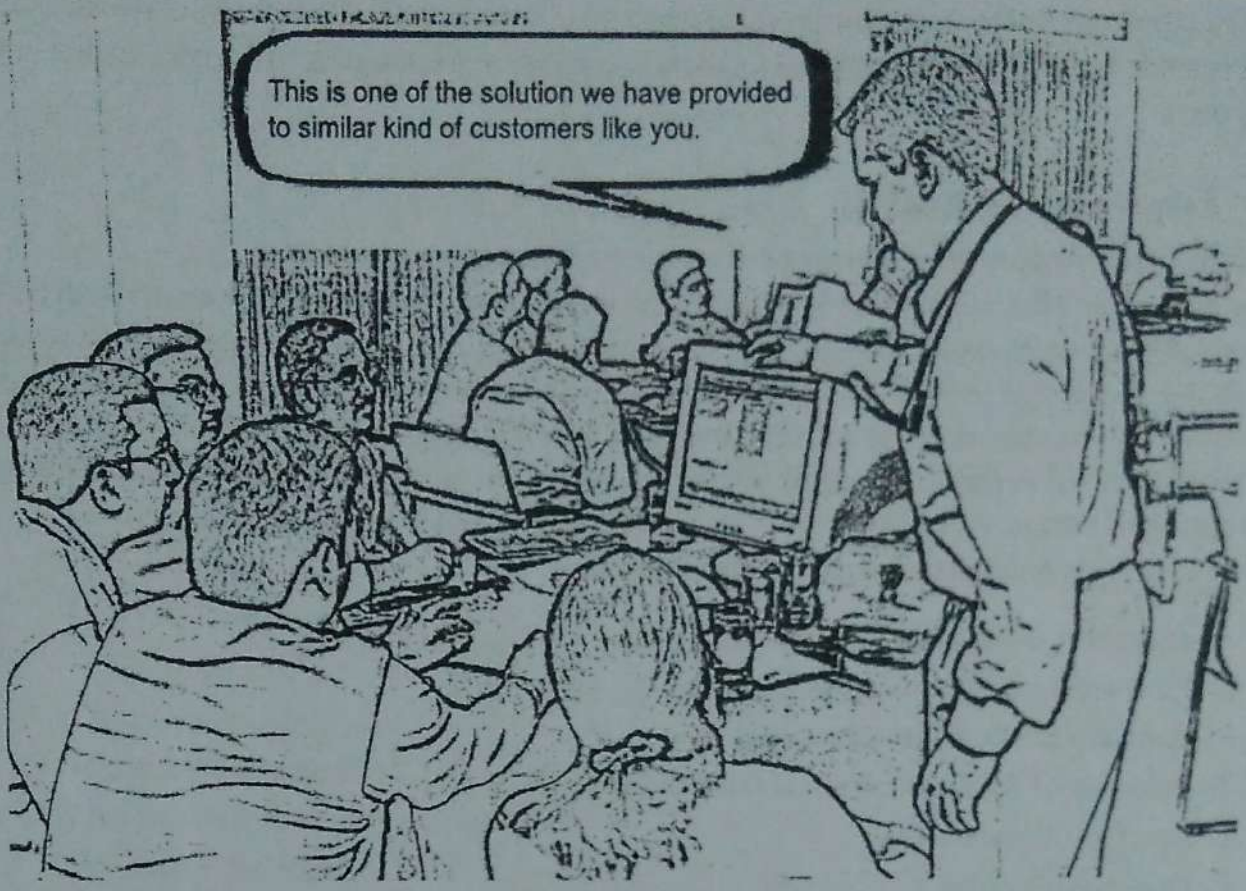


Figure 2.15 Facilitated Workshop Technique

the requirements are focused on one area of business in which the participants have knowledge and consensus is being sought. In workshops, different alternatives are given to the customers so that they choose the appropriate options.

Prototyping The word "prototype" is derived from the Greek word "prototypon." Prototypes and models are the best ways of presenting ideas to users. They give users a glimpse of what they might get. More requirements are likely to emerge when users are able to see what they will get as an outcome of their suggestion. This technique aims to get users to express their requirements. Prototyping is used to get feedback from users. Business logic may not be coded in prototyping and experimental systems are developed using prototyping. It is actually the initial version of the system.

POINTS TO PONDER

During the requirements phase, the prototype is generally document-based, where the user interface design, use cases, overall look and feel, navigation, etc. are discussed with the end users. The scope of a working prototype normally comes at the construction phase.

Types of Prototypes As discussed in Chapter 1, prototypes are classified into two broad categories, namely, evolutionary prototype and throw away prototype. The former type is reused and the changes are implemented in that particular prototype. The new changes and corrections are updated in the

existing prototype until the final list of requirements is obtained, without wasting the used resources. The latter type, as the name suggests, destroys the wrong prototypes. Throw away prototypes are proven to be wrong and thus the new prototypes are developed in the subsequent stages. The resources used in the previous prototypes are either freed or rejected.

Figure 2.16 shows the different types of prototypes, which are discussed below.

Proof-of-principle prototype (bread board): Only the intended design is tested and visual appearance is not in the scope of this prototype. For understanding purposes, a few working codes are part of this prototype.

Form study prototype: Only visual appearance is considered and the functionalities are not considered.

Visual prototype: It simulates the appearance, color, fonts and surface textures of the intended product, but it does not represent of the final function(s) of the final product.

Functional prototype: It is also called a working prototype, which means it simulates the actual functionality of the intended work.

Questionnaires Questionnaires can be used to collect input from multiple stakeholders quickly, which can be consolidated to create a list of requirements. As the stakeholders targeted for the questionnaire need not be physically present with the requirements elicitation team, this process can be run with a large number of participants in quite an inexpensive fashion.

However, the questionnaires have their own limitations such as designing an exhaustive questionnaire becomes difficult especially if the complexity of the system is high. As the stakeholders are not in direct contact with the elicitation team, there may be ambiguity in questions that lead to erroneous responses. Moreover, the response rate may be low and may need a lot of follow-up for getting sufficient input through this technique.

Brainstorming Brainstorming is a powerful technique using which a large number of requirements or ideas can be generated in a short period of time.

- In these sessions, members meet face-to-face and rely on both verbal and nonverbal interactions to communicate with each other.
- The main purpose of the brainstorming technique is to get as many ideas as possible to obtain different views of the requirements, thereby helping to capture better requirements.

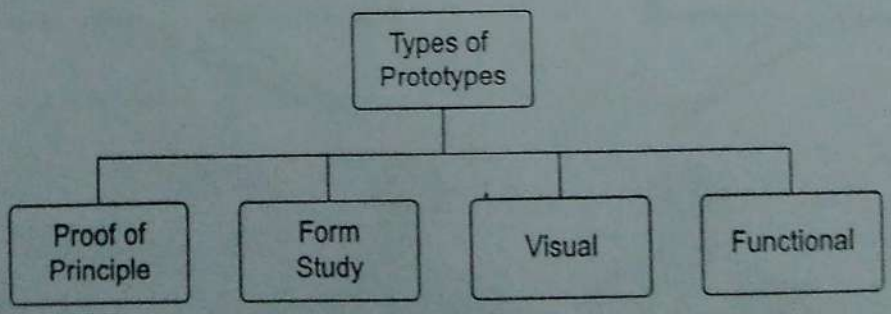


Figure 2.16 Types of Prototypes

In a typical brainstorming session, about six people are involved and the group leader states the problem in a clear and understandable manner so that it is understood by all the participants involved. Members are allowed to suggest as many alternatives as they can in a given period of time. No criticism is allowed. Alternative ideas generated are also recorded and are used for later analyses and discussions. Brainstorming, however, is merely a process used to generate ideas and may not be effective for decision-making purposes.

Discussion Questions

Do you think the brainstorming sessions can help in a situation where the end users do not have a clear idea of the features they want from the software to be developed?

1. _____
2. _____
3. _____
4. _____

Direct Observation and Apprenticing In many cases a new system is developed to replace an existing system, in which either a lot of manual intervention is involved or it is not efficient and suitable for doing the job. In such scenarios, it is very useful to gather knowledge of the requirements by observing (Figure 2.17) what is actually done by the user of the system. This is far more powerful than going through the documents of the capability of the existing system as it provides an opportunity to capture the real objective behind creating the new system.

A better technique will be the apprenticing model where the requirements engineer performs the tasks that the user of the system carries out. Although this requires the understanding of the business

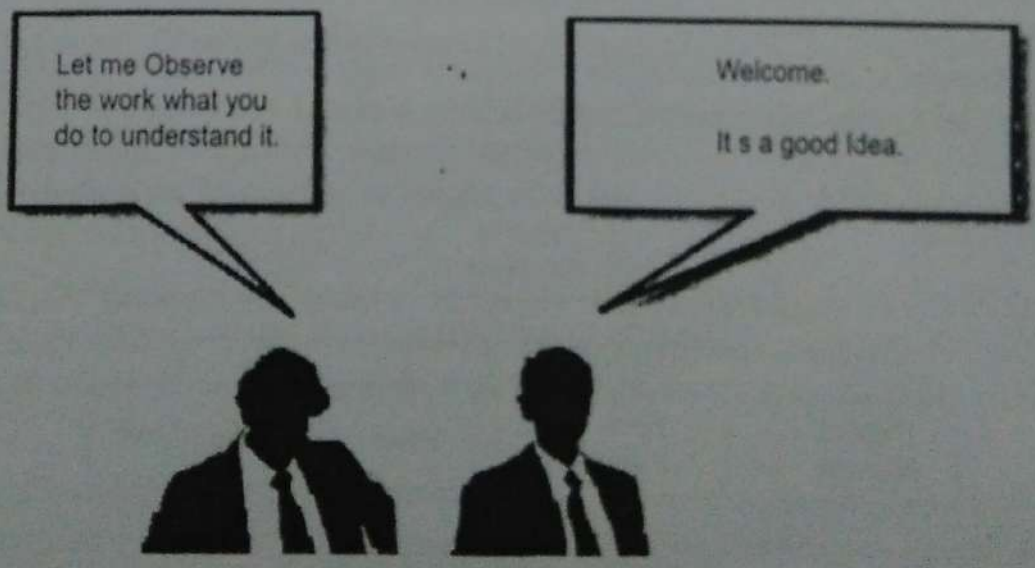


Figure 2.17 Observation Technique

process carried out by the user, this has a huge benefit of getting an in-depth insight into what tasks the users perform with the current system and the areas they need improvement or automation in the new system.

2.5.3 Requirements Analysis

Requirements analysis is the third step of the requirements engineering process.

The aim of the elicitation phase is to gather the requirements, whereas in the analysis phase the requirements are understood in detail. This is the phase that bridges the gap between requirements engineering and design (Figure 2.18). Thus, the requirements analyst refines the data and functional and behavioral constraints of the software, which acts as the input to the design. Various modeling techniques are used during the analysis phase, which helps in problem evaluation and solution synthesis during this stage.

The focus areas for the analyst during this stage are

a. Externally observable data objects
b. Flow and content of information
c. Software functionality elaboration
d. Behavior of the software on an external input
e. Interface requirements
f. Design constraints

After analyzing each of these areas, the analyst starts working on the solution part. The system architecture, database requirements, etc. are discussed with the customer to ensure that the system design is based on the customer's need. For example, the data flow analysis may reveal that a real-time interface is not required in the system, which will reduce the complexity and cost of the proposed solution. The analyst creates models of the system to capture the data requirement and flow, functional processing, operational behavior and information content. These models become the stepping stones for the software design phase.

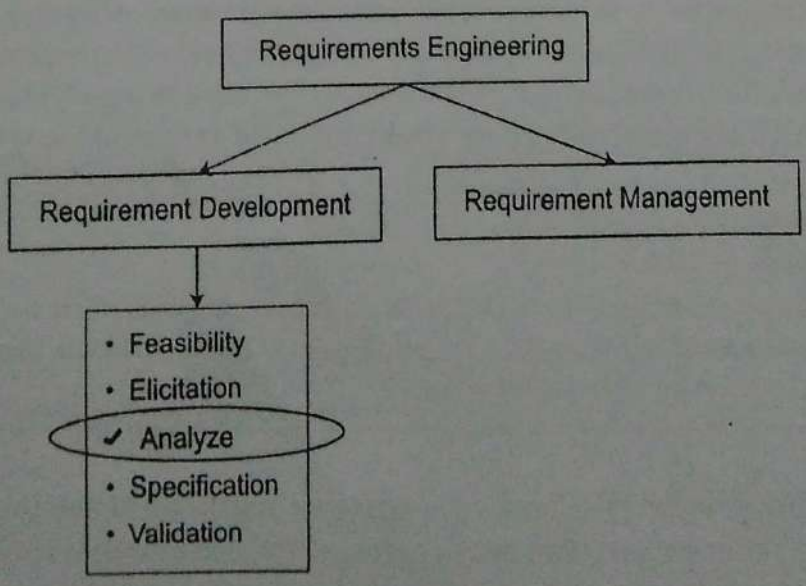
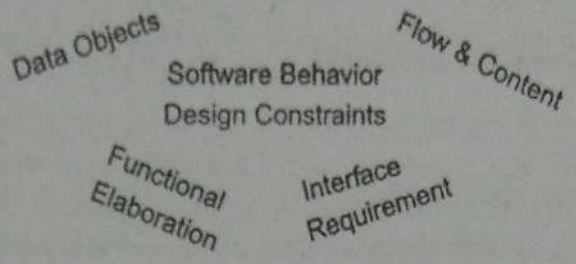


Figure 2.18 Analysis Phase of Requirements Engineering



The following modeling techniques are widely used in requirements analysis and are discussed in Chapter 3.

- Use case models
- System models (context, behavioral, data, object, structure methods)
- Data modeling
  - Data flow diagram
  - Data dictionary
  - Entity relationship model
  - Control flow model
  - Decision tables
- Information modeling
- Object-oriented analysis
- Scenario-based modeling
- Flow-oriented modeling
- Class diagram

2.5.4 Specifying Requirements

After the successful completion of requirements elicitation and analysis, the next step is to put all the knowledge gathered during these steps in a clear, concise and unambiguous manner so that the design and development teams can use it going forward. This is called the requirements specification phase (Figure 2.19).

While specifying the requirements, it is also important that it is validated with the customers and users to confirm if all the requirements are captured at this stage. Software Requirement Specification (SRS) is the outcome of this requirements specification and validation stage.

2.5.4.1 Specifying Requirements in the SRS Document

Once created, the SRS document becomes the main source for any requirements clarification and dissemination to all stakeholders. The SRS document can be used for several purposes:

- It may form the basis for the contractual agreement between the customer and the suppliers. The suppliers in an unambiguous fashion describe what will be developed as part of the project, thus any change in this agreed understanding will be renegotiated between the customer and the suppliers.
- This is the foundation document for the test engineers to develop their strategy on how to test the proposed system fully.

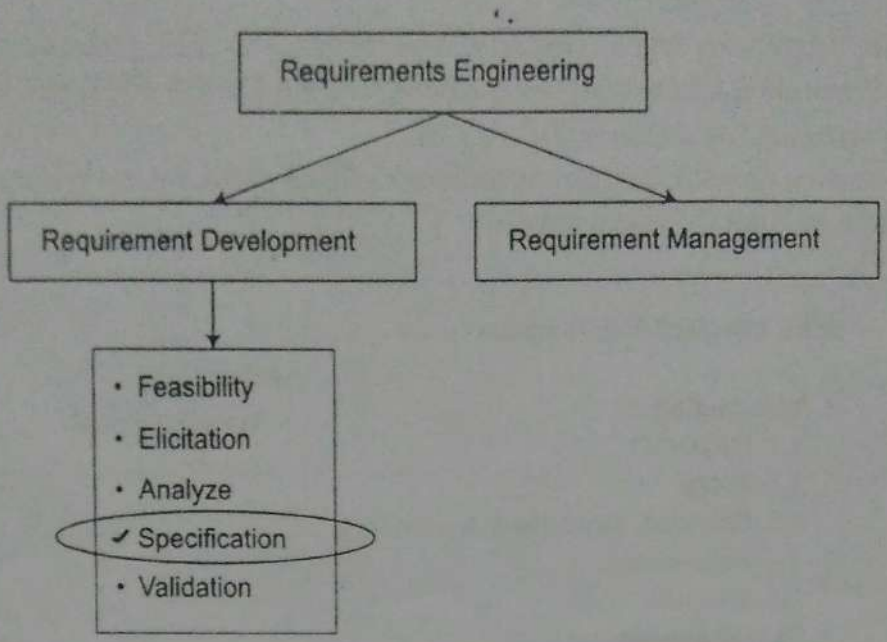


Figure 2.19 Specification Phase of Requirements Engineering

- The design and development team start their work based on this document. As all the stakeholders think through what they need from the system and get it documented in the SRS. The chance of redesign or reconstruction is minimized if the SRS is developed well.
- The resourcing, budgeting, scheduling and pricing for the project are based on the volume and complexity of the requirements captured in this document. The modern estimation techniques take direct input from the use cases and models developed in the SRS.
- A good SRS ensures future maintainability of the project and enables the customer to switch to a different supplier during the maintenance phase of the project. The new supplier's team will use the SRS document as the primary artifact to understand the capabilities required by the stakeholders from the system and can validate this against the developed capabilities.

Discussion Questions

List down the qualities of an SRS document that will help it to become the primary reference document for all the requirements?

1. _____
2. _____
3. _____
4. _____

2.5.4.2 IEEE Standard SRS Template

The aim of the SRS document is to provide an understanding of what the stakeholders want from the system and how the system should behave to cater to these needs. As discussed above, this document forms the foundation for downstream processes such as design, construction and testing, and thus it is

very important to take proper care while creating this document. The IEEE Standard "*IEEE Recommended Practice for Software Requirements Specifications*" (IEEE Std 830-1993) provides a template for creating the SRS document as shown in Figure 2.20.

Based on the application domain, complexity and involvement of third-party systems, the sections may get added or removed from the above template.

IEEE standard SRS Template

1. Introduction
- 1.1. Purpose
- 1.2. Scope
- 1.3. Definition, acronyms & abbrevations
- 1.4. References
- 1.5. Overview

2. Overall description
- 2.1. Product perspective
  - 2.1.1. System interfaces
  - 2.1.2. User interfaces
  - 2.1.4. Software interfaces
  - 2.1.5. Communictions interfaces
  - 2.1.6. Memory constraints
  - 2.1.7. Operations
  - 2.1.8. Site adaptation requirement
- 2.2. Product functions
- 2.3. User characteristics
- 2.4. Constraints
- 2.5. Assumptions and dependencies
- 2.6. Apportioning of requirements

3. Specific Requirements
- 3.1 External interface requirments
  - 3.1.1 User interfaces
  - 3.1.2 Hardware interface
  - 3.1.3 Software interfaces
  - 3.1.4 Communication interface
- 3.2 Specific requirments
  - 3.2.1 Sequence diagrams
  - 3.2.2 Classes for classification of specific requirments
- 3.3 Performance equirments
- 3.4 Design constraints
- 3.5 Software system attributes
  - 3.5.1 Reliability
  - 3.5.2 Avilability
  - 3.5.3 Security
  - 3.5.4 Maintainability
- 3.6 Other requirments

4. Supporting information
- 4.1 Table of contents and index
- 4.2 Appendixes

Figure 2.20 IEEE Standard SRS Template

2.5.4.3 Checklist for Writing a Good SRS Document

The consumers of the SRS document are all the stakeholders of the project – the customer, end users, project engineers, interface developers, managers, etc. Thus, ensuring correctness and completeness of this document is crucial for the success of each stage of the project. The checkpoints listed in Figure 2.21 provide a guideline on how to create an SRS document that serves its purpose.

1. **Correctness**
   a) Will the requirements meet the customer's need?
   b) Do the requirements capture how the system should transform, produce and provide the exact outcome expected from the system?
   c) The correctness of requirements typically refers to the accuracy of the requirements and whether everything agreed is delivered.

2. **Completeness**
   a) Are all the requirements captured which will form a system that fully provides a solution to the customer's problem?
   b) Are each requirement detailed enough and supported by necessary diagrams, figures, data and use cases so that all the stakeholders get their necessary input from the requirement?
   c) Are all functional, nonfunctional and interface requirements captured for the system?

3. **Consistency**
   a) Are there requirements that conflict with each other? This may happen when multiple stakeholders have different views of the proposed system and provide requirements that create conflicts while grouped together.
   b) Are the terminology, diagrams, tables and figures consistent and provide a consistent view of the proposed system?

4. **Verifiability**
   a) Are the requirements verifiable -- that is, will the test engineers be able to check whether the requirement fulfills the customer's need in entirety before the product is shipped to the customer.

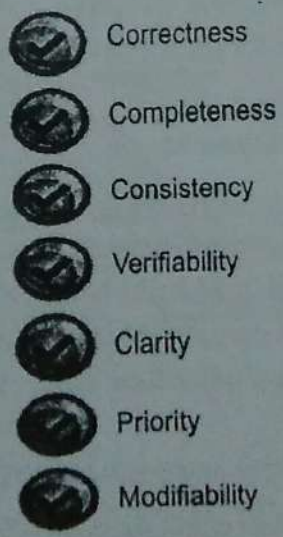


Figure 2.21 Checklist for Writing a Good SRS Document

b) Are the requirements validated and verified by all the stakeholders before they are baselined for consumption of downstream processes?

5. **Clarity**
   a) Is the requirements statement explicit enough and can be interpreted only in one way? Along with unambiguous language, it is important to take help of pictorial representations such as diagrams, figures and use cases to eliminate ambiguity in requirements.
   b) Will the design and construction engineer be able to determine the single way of implementation from the requirements document? In case there are multiple interpretations of a requirement possible, then there is a possibility that it is implemented in a way not desired by the customer and thus causes rework.

6. **Priority**
   a) Is the relative priority of the requirements vis-a-vis the others captured? Prioritization of the requirements is important as it drives the plan for implementing these. In the case when the project is under time or budget constraint, the priority of the requirements becomes important as it will provide a clear guideline of which requirements should be tackled first, and which requirements are less important.

7. **Modifiability**
   a) How much dependency does the requirement have on other requirements? This will depict how easily the requirements can be modified. A requirement that has dependency on several other requirements should be structured in a fashion so that the impact of any modification to this requirement across other requirements can be easily traceable. There are several requirements management tools such as Doors and RequisitePro that are available which help in this regard.

POINTS TO PONDER

Note the seven points to be taken care of in a good SRS document – correctness, completeness, consistency, verifiability, clarity, priority and modifiability.

2.5.5 Validating Requirements

Validating requirements (Figure 2.22) is an important step as invalid requirements may cost more to rectify at later stages of the project life cycle. The main aim of requirements validation is to ensure that the customer needs are captured completely, clearly and consistently.

The validation step should provide enough confidence to all the stakeholders that the proposed system will provide all the features required and will be completed within the time and budget allocated for the project. The different stakeholders who should be involved in the validation process are:

1. Customer
2. End user
3. Domain expert
4. Architect
5. Construction and test engineer
6. Third-party systems interacting with the proposed system

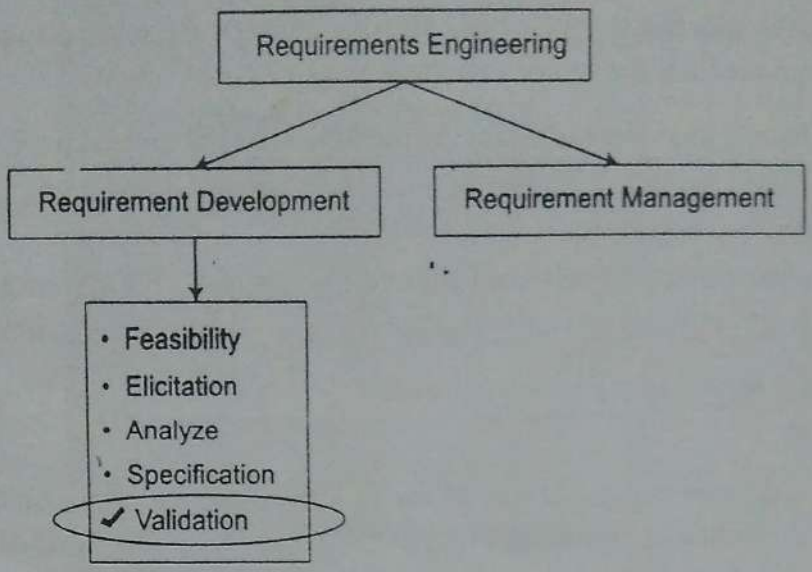


Figure 2.22 Validation Phase of Requirements Engineering

There are different approaches taken for validating the requirements, which are discussed below.

Walkthrough, Review and Inspection

There is some subtle difference between the validation processes – walkthrough, review and inspection (Figure 2.23).

Walkthrough: This is an informal process where the author of the document presents the document to a group of people, and the observations or issues found by the group are reported and corrected after the walkthrough session. Generally, minutes of the meeting are not maintained for the walkthrough sessions.

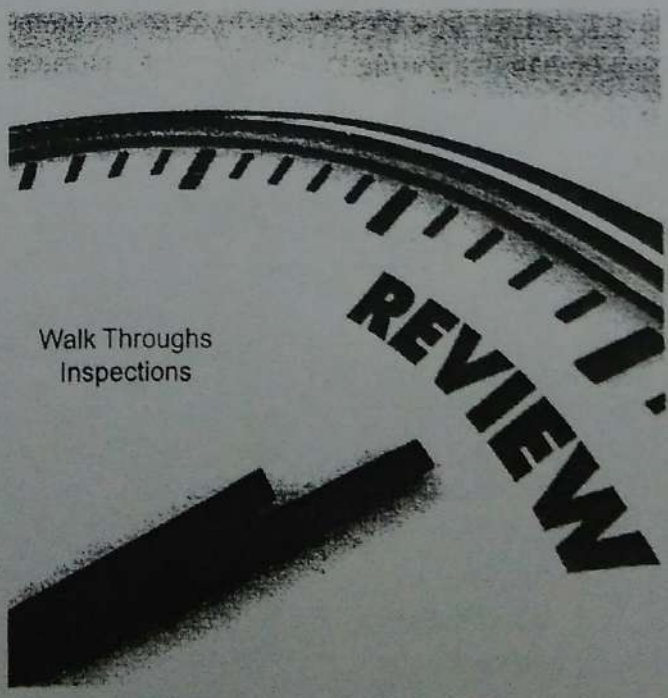


Figure 2.23 Walkthrough, Review and Inspection

Review: This may be either a formal or an informal process where the group of stakeholders will critically go through the SRS document and check that the following:

a) The standards and guidelines available in the organization are followed in the document.
b) The completeness, consistency and other points mentioned in the Checklist for Writing a Good SRS Document are followed.
c) All the requirements could be traced back to their origin. This will ensure that only the valid requirements are captured and implemented and not just any wish list from the user groups.

In this process, the minutes of the review sessions are captured and all the findings are tracked until satisfactory closure.

Inspection: This is a more structured way of review where the nonconformances found in the SRS document based on the points discussed above are listed as defects and tracked until the defects are closed satisfactorily.

Modeling and Prototyping

The other approaches for validating the requirements include creating models and prototypes.

Models: Use case modeling using the Unified Modeling Language (UML) is the most frequently used technique for capturing and validating the business case, interactions and flows in the proposed system.

Prototypes: As discussed in the Requirements Elicitation section, the prototyping method is a very effective way to validate the technical feasibility and the user's need because a smaller version of the actual system is developed and showcased to the stakeholders. Therefore, prototypes can be considered as the *executable* models used for requirements validation.

2.5.6 Requirements Management

The requirements management (Figures 2.24 and 2.25) discipline deals with how the existing requirements can be stored and tracked and how the changes to these requirements and also the introduction of new requirements can be handled during the requirements phase as well as throughout the life cycle of the project.

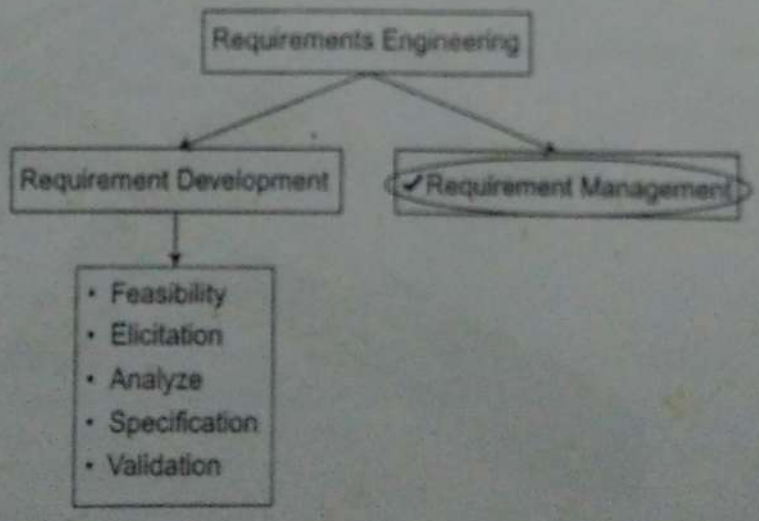


Figure 2.24 Requirement Engineering-Management

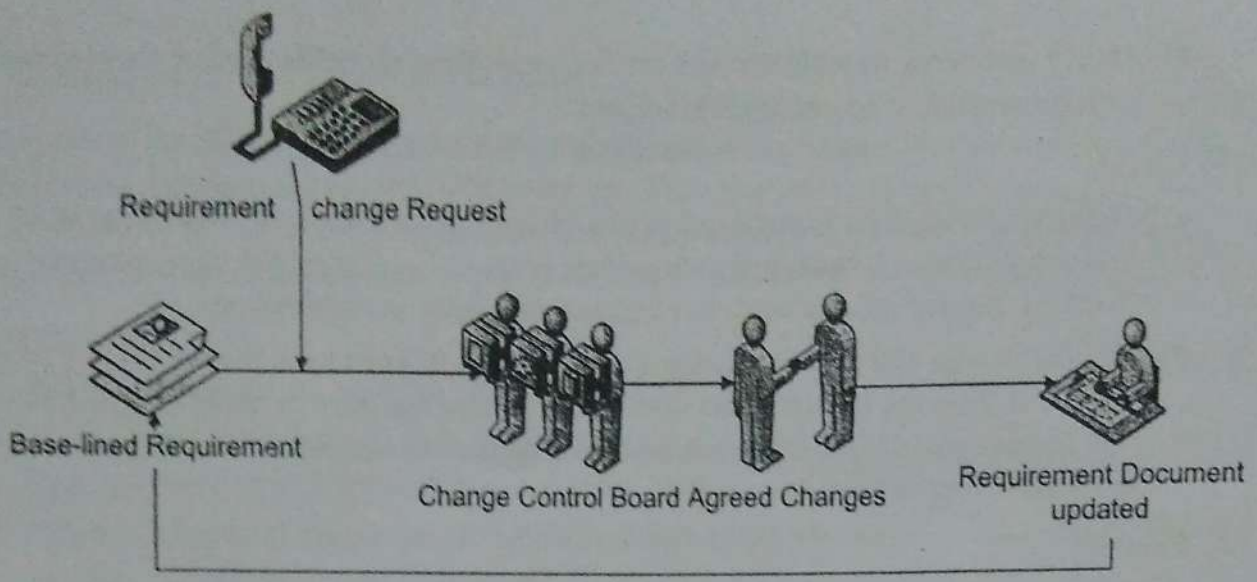


Figure 2.25 Requirements Management

Discussion Questions

If there is no requirements management process defined for a project what will be the problems faced by the team?

1. _____
2. _____
3. _____
4. _____

2.5.6.1 Requirements Management Plan

During requirements engineering, the plan for managing the requirements should mention how to manage the following:

- Requirements identification
- Requirements change
- Requirements status tracking
- Requirements traceability

2.5.6.2 Requirements Identification

Requirements identification is the first step in requirements management, which is discussed in detail in the first few sections of this chapter. This creates the baseline set of requirements on which the implementation team starts working.

2.5.6.3 Requirements Change

Requirements are bound to change during the course of time. There can be several reasons for the change in requirements:

1. The customer and the users of the system get more insight into the system as they see a few working prototypes of the system. This leads to change in existing requirements and emergence of new requirements. This is a very common phenomenon as no human being can foresee a future system fully just by thinking about it. Let us consider a simple example of designing the interior of your house. While providing the specification, you have to imagine how the different furniture will be oriented, the colors of the walls, electrical fittings, etc. However, when you see the first model design created by the designer you get a more concrete idea on how things will look and you make modifications to the design more confidently. The changes may occur until the different pieces of the interior are built and fixed. In the similar way, as more insight comes out of the system as we progress on the project, there may be changes or clarifications to the existing requirements.
2. The market and the business needs are ever changing. To cope with the changing market needs, the specification of the product to be developed may have to be changed.
3. The priority of the requirements may change due to change in market scenario, business priority change and change in viewpoint of the different stakeholders of the system.

A strong change management process is essential in tracking the changes in requirements. This will ensure that all the changes are incorporated during the project and minimize the chance of missing out on the new requirements or changes that have been found after the requirements phase of the project. The stages of requirements change management are mentioned below.

Change Control Process Once the initial requirements are agreed upon with different stakeholders, it is important that any change in these requirements are controlled well. Changes in requirements are almost always related to change in project effort, timeline and cost. By introducing a *Change Control Board*, all the stakeholders try to ensure that everyone gets good transparency on the changes agreed upon. A change control board is a group of people comprising members from each stakeholder group who are directly linked with the project outcome. This group sets forth the steps to be followed whenever a change or a new requirement crops up after the initial set of requirements are signed off by everyone.

2.5.6.4 Requirements Status Tracking

After the requirements gathering phase, the set of requirements get baselined, which can be modified after the change control board approves a requirement for implementation. It is important to track the status of all these requirements throughout the life cycle of the project, that is, until the requirements are available to the customer. This is to ensure that at any point in time the stakeholders should be able to find out the progress on these requirements. Figure 2.26 shows a sample of the change request tracker.

2.5.6.5 Requirements Traceability

The most commonly used artifact to track the requirements in each stage of the project life cycle such as design, construction, unit testing and system testing is a traceability matrix that maintains the tracing information as shown in Figure 2.27.

Note that each column of this table can be used to capture and maintain the traceability in the downstream phases for particular business and system requirements. Forward traceability provides

Change Request Details											Change Request Implementaion Details					
CR Id	Requested By	Requested Date	Priority	Change Description	Impact				CR Approved By	CR Approved date	CR status	Reason (if Rejected / Deferred/ Cancelled)	Planned Start Date	Planned End Date	Actual Start Date	Actual End Date	Actual Effort (in PH)	CR Implementation Status
					Sl. No.	CI Id	Name	Existing Version (in PH)										
					1													
					2													
					3													
					4													
					5													

Figure 2.26 Change Request Tracker

Require-ment Clarifi-cation	Business Require-ments	System Re-quirements		NFR Re-quire-ments	Related Require-ments	User Interface Proto-type	System Testing		Performance Testing		Design				Coding		Unit Testing		Integration Tesing	
Clarifi-cation ID	Business Requir-ment ID	System Requir-ment ID	System Use Case ID	NFR Requir-ment ID	Related Requir-ment ID	Screen /Wire-frame ID	Docu-ment Name (SCI ID)	Test Case ID	Docu-ment Name (SCI ID)	Test Case ID	Com-ponent/ Class Diagram ID	Se-quence Diagrm	Use Interface Specifica-tionScreen ID/Name	Data-base Table	Program/ Source Code File Name(s)	Method/ (Proce-dures)	Docu-ment Name (SCI ID)	Test Case ID	Docu-ment Name (SCI ID)	Test Case ID

Figure 2.27 Requirements Traceability Matrix

the ability to track a requirement up to the system test case and through backward traceability the source for each of the system test or code or design can be traced back up to the requirements. This ensures that ALL and ONLY the agreed upon requirements are delivered to the customer.

POINTS TO PONDER

Due to the complexities involved in handling the changes, there is a general tendency to try to avoid requirement changes. But for a long-duration and complex project requirement change is inevitable, and thus rather than pushing back the customer to resist the requirement change it more advisable to plan for a well-structured change management process.

Summary

- The requirements engineering phase creates the foundation for any software development project. Thus, it is important to follow a systematic approach in this phase to avoid issues with the software at later stages.
- Feasibility study is conducted at the beginning to ensure that all the expectations of the stakeholders are analyzed and a consensus is reached on how much of that can be created with the current time and cost constraints of the project.
- Requirements elicitation techniques are used to ensure that all the needs from the system are captured. The different and conflicting viewpoints of the different stakeholders and user groups of the system need to be merged and agreed upon before proceeding to the next step.
- Modeling techniques are used to analyze the requirements and synthesize a solution, which becomes the input to the design phase. System analysts will critically analyze the information flow and content, functional behavior, data flow behavior and external system interfacing requirements to create a model with which both the customer and the engineers are comfortable.
- Specifying the requirements in a correct, complete, clear, verifiable and prioritized fashion is critical to ensure that the outcome of the requirements engineering phase is useful for the subsequent stages of the software development life cycle. Review and inspection methodologies are used to ensure the quality of the SRS document.
- Throughout the project life cycle, tracking the requirements for proper implementation and any change is crucial. The requirement traceability matrix provides a standard way of tracking the requirements throughout the project life cycle. Apart from this, a sound change management process and change tracker is recommended to track the changes to the initial requirements and avoid requirements creep.

Case Study

An SRS review checklist is a critical item in the requirements phase of the project. A sample checklist that covers the aspects discussed in the SRS creation and validation sections is provided below.