Object Oriented Software Engineering

INTRODUCTION TO SOFTWARE ENGINEERING:

The term **software engineering** is composed of two words, software and engineering. **Software** is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be a collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called **software product.**

Engineering on the other hand, is all about developing products, using well-defined, scientific principles and methods.

So, we can define **software engineering** as an engineering branch associated with the development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

IEEE defines software engineering as:

The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software.

We can alternatively view it as a systematic collection of past experience. The experience is arranged in the form of methodologies and guidelines. A small program can be written without using software engineering principles. But if one wants to develop a large software product, then software engineering principles are absolutely necessary to achieve a good quality software cost effectively.

Without using software engineering principles, it would be difficult to develop large programs. In industry it is usually needed to develop large programs to accommodate multiple functions. A problem with developing such large commercial programs is that the complexity and difficulty levels of the programs increase exponentially with their sizes. Software engineering helps to reduce this programming complexity. Software engineering principles use two important techniques to reduce problem complexity: ***abstraction*** and ***decomposition*****.**

The principle of **abstraction** implies that a problem can be simplified by omitting irrelevant details. In other words, the main purpose of abstraction is to consider only those aspects of the problem that are relevant for certain purposes and suppress other aspects that are not relevant for the given purpose. Once the simpler problem is solved, then the omitted details can be taken into consideration to solve the next lower-level abstraction, and so on. Abstraction is a powerful way of reducing the complexity of the problem. The other approach

to tackle problem complexity is **decomposition**. In **decomposition** technique, a complex problem is divided into several smaller problems and then the smaller problems are solved one by one. However, in this technique any random decomposition of a problem into smaller parts will not help. The problem has to be decomposed such that each component of the decomposed problem can be solved independently and then the solution of the different components can be combined to get the full solution. A good decomposition of a problem should minimize interactions among various components. If the different subcomponents are interrelated, then the different components cannot be solved separately and the desired reduction in complexity will not be realized.

NEED OF SOFTWARE ENGINEERING:

The need for software engineering arises because of the higher rate of change in user requirements and the environment in which the software is working.

- **Large software -** It is easier to build a wall than to build a house or building, likewise, as the size of software becomes large engineering has to step to give it a scientific process.
- **Scalability-** If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one.
- **Cost-** As hardware industry has shown its skills and huge manufacturing has lowered the price of computer and electronic hardware. But the cost of software remains high if the proper process is not adapted.
- **Dynamic Nature-** The always growing and adapting nature of software hugely depends upon the environment in which the user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where software engineering plays a good role.
- **Quality Management-** Better process of software development provides better and quality software product.

CHARACTERESTICS OF GOOD SOFTWARE

A software product can be judged by what it offers and how well it can be used. This software must satisfy the following grounds:

- Operational
- Transitional
- Maintenance

Well-engineered and crafted software is expected to have the following characteristics:

Operational

This tells us how well software works in operations. It can be measured on:

- Budget
- Usability
- Efficiency
- Security
- Correctness
- Functionality
- Dependability
- Safety

Transitional

This aspect is important when the software is moved from one platform to another:

- Portability
- Reusability
- Interoperability
- Adaptability

Maintenance

This aspect briefs about how well a software has the capabilities to maintain itself in the ever- changing environment:

• Modularity	• Flexibility
• Maintainability	• Scalability

In short, Software engineering is a branch of computer science, which uses well-defined engineering concepts required to produce efficient, durable, scalable, in-budget and on-time software products.

TYPE OF SOFTWARE

There are many different types of software. One of the most important distinctions is between

- custom software,
- generic software and
- embedded software.

Custom software is developed to meet the specific needs of a particular customer and tends to be of little use to others (although in some cases developing custom software might reveal a problem shared by several similar organizations). Much custom software is developed in-house within the same organization that uses it; in other cases, the development is contracted out to consulting companies. Custom software is typically used

by only a few people and its success depends on meeting their needs. Examples of custom software include web sites, air-traffic control systems and software for managing the specialized finances of large organizations.

Generic software, on the other hand, is designed to be sold on the open market, to perform functions that many people need, and to run on general purpose computers. Requirements are determined largely by market research. There is a tendency in the business world to attempt to use generic software instead of custom software because it can be far cheaper and more reliable. The main difficulty is that it might not fully meet the organization's specific needs. Generic software is often called Commercial Off-The-Shelf software (COTS),

and it is sometimes also called shrink-wrapped software since it is commonly sold in packages wrapped in plastic. Generic software producers hope that they will sell many copies, but their success is at the mercy of market forces. Examples of generic software include word processors, spreadsheets, compilers, web browsers, operating systems, computer games and accounting packages for small businesses.

Embedded software runs specific hardware devices which are typically sold on the open market. Such devices include washing machines, DVD players, microwave ovens and automobiles. Unlike generic software, users cannot usually replace embedded software or upgrade it without also replacing the hardware. The open-market nature of the hardware devices means that developing embedded software has similarities to developing generic software; however, we place it in a different category due to the distinct processes used to

develop it.

SOFTWARE CRISIS:

Software engineering appears to be among the few options available to tackle the present software crisis.

Let us explain the present software crisis in simple words, by considering the following.

The expenses that organizations all around the world are incurring on software purchases compared to those on hardware purchases have been showing a worrying trend over the years (Fig 1.6)

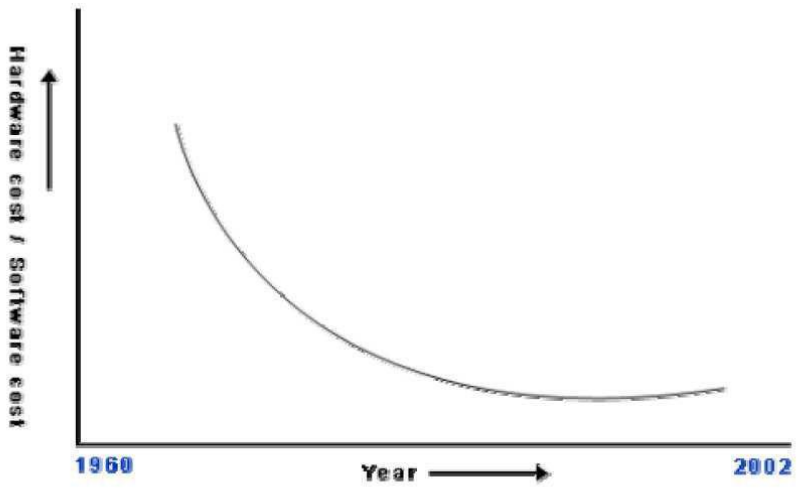


Fig. 1.6: Change in the relative cost of hardware and software over time

Organizations are spending larger and larger portions of their budget on software not only are the software products turning out to be more expensive than hardware, but also presented a lot of other problems to the customers such as: software products are difficult to alter, debug, and enhance; use resources non optimally; often fail to meet the user requirements; are far from being reliable; frequently crash; and are often delivered late.

Due to ineffective development of the product characterized by inefficient resource usage and time and cost over-runs. Other factors are larger problem sizes, lack of adequate training in software engineering, increasing skill shortage, and low productivity Improvements.

S/W crisis from programmer point of view: -

- Problem of compatibility.
- Proclaiming documentation.
- Problem of pirated s/w.
- Problem of Portability.
- Problem in co-ordination of work of different people.
- Problem of proper maintenance.

S/W crisis from user point of view: -

- s/w cost is very high.
- Lack of development specification.
- Problem of bugs or errors.
- Price of h/w grows down.
- Problem of different s/w versions.
- Problem of proper maintenance.

OVERVIEW OF SOFTWARE DEVELOPMENT ACTIVITIES:

Now, let us look at some major differences in the way software is being written now one is that the software engineering principles. The software writing problem is not the same as the problem that now we are solving.

The problems have become much more complex, but at the same time these are only incrementally different from the problem that has been solved. Let us let us look at these issues now; the differences between exploratory style and the modern software development practices. Compared to 1950s or 60s where the exploratory style was used, now the life cycle models are used every program is developed according to an adopted life cycle model, where there are a set of stages and the development occurs through those stages. The stages typically the requirements analysis specification design, coding, testing, and so, on.

One major difference between the way program was written 1950s and 60s to now is that the emphasis has shifted from error correction to error prevention. The earlier technique the exploratory style was to first write the program complete somehow and then start correcting the errors, there will be hundreds of errors keep on eliminating one by one until all errors are eliminated. But, now the emphasis is error prevention as the program is written we track the errors and remove it from there itself. So, that during testing we have very less number of errors to correct, or in other words as soon as we have a mistake committed by the programmer in program development, we try to detect it as close to the time of commitment of the error as possible the main advantage of this is that it is very cost effective.

If, you correct the error after testing then you will have to first find out where exactly the error has occurred make changes to the code again test and so on. Here, before testing we just identify the place where the error is there we do not have to look at we just look through or review the code and so on, review the specification review the design identify the error corrected there itself and do not wait for error to be discussed detected and determined during the testing phase. The software is being developed through number of phases and in each phase, wherever a program commits a mistake. It is detected and corrected in that same phase as far as possible.

The exploratory style program development was basically coding, start writing the code as soon as the problem is given, but right now in the modern development practice coding is actually a small part of the development activities. The major activities are specification design then coding and then testing.

Of course, lot of attention now is given to requirement specification then the design has to be done that, standard design techniques that are available. And, at the end of every phase

reviews are conducted the main idea behind review is to detect errors as soon as these are committed by the programmer, do not wait for the error to be detected during testing. Reviews help us to detect the errors much earlier, and this is an efficient way to detect errors and therefore, reduces cost of development and the time of development.

The testing techniques have become really sophisticated we will look at them later in this course, and there are many standard testing techniques and tools are available. Earlier the programmer just wrote the code and then never know when it will get completed you will just have to ask the programmer, how far he has done in the coding?

But, even if he say that I am nearly complete completing the code, but then he might have too many bugs and he may take years to correct them. But, right now there is a visibility of the development somebody can see that the design is done, code is done or let us say the requirements are done. So, these documents improve the visibility, earlier there are no visibility you just have to ask the programmer that how far he has done? Right now you can see the documents and determine with what stages of development it is.

The increased visibility makes software project management much easier, earlier the project manager has to just ask the programmer how he is doing, how long he is? And, naturally the projects were going hey we are one year project taking 5 years was not uncommon, but now due to the increased visibility the project manager can very accurately determine when the project at what stage it is and how long it will take to complete the work?. Because, good documents now are produced later maintenance becomes easier several metrics are being used, which are used to determine the quality of the software, software project management and so on. There are many project management techniques that are being used like estimation scheduling monitoring mechanisms and also case tools are being used extensively.

SOFTWARE DEVELPOMENT PROCESS MODELS:

- software development life cycle, various models are designed and defined. These models are called Software Development Process Models.
- On the basis of project motive, the software development process model is selected for development.

Following are the different software development process models:

1. Waterfall Model
2. RAD model
3. Incremental Model
4. Agile model
5. Prototype Model
6. Iterative model
7. V model
8. Spiral model

1. **Waterfall Model**
   - Used when requirements are well understood in the beginning
   - Also called classic life cycle and is a systematic, sequential approach to Software development
   - Begins with customer specification of Requirements and progresses through planning, modeling, construction and deployment.
   - In this model, each phase is executed completely before the beginning of the next phase. Hence the phases do not overlap in waterfall model.
   - This model is used for small projects.
   - In this model, feedback is taken after each phase to ensure that the project is on the right path.
   - Testing part starts only after the development is completed.

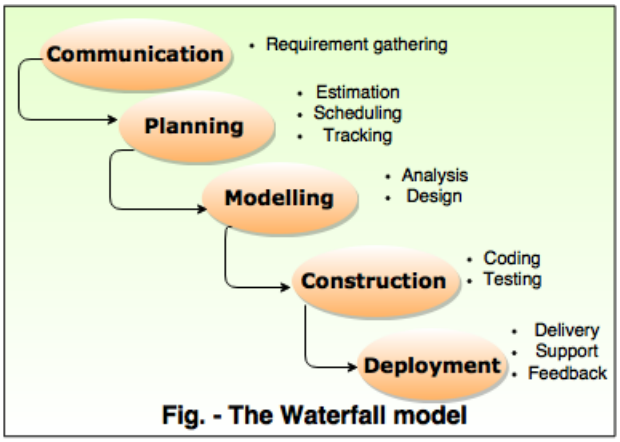


Fig. - The Waterfall model

Following are the phases in waterfall model:

1. **Communication**
   The software development starts with the communication between customer and developer.
2. **Planning**
   It consists of complete estimation, scheduling for project development.
3. **Modelling**

- Modelling consists of complete requirement analysis and the design of the project i.e algorithm, flowchart etc.
- The algorithm is the step-by-step solution of the problem and the flow chart shows a complete flow diagram of a program.

4. **Construction**

- Construction consists of code generation and the testing part.

- Coding part implements the design details using an appropriate programming language.
- Testing is to check whether the flow of coding is correct or not. Testing also checks that the program provides desired output.

5. Deployment

- Deployment step consists of delivering the product to the customer and taking feedback from them.
- If the customer wants some corrections or demands for the additional capabilities, then the change is required for improvement in the quality of the software.

Advantages of Waterfall model	**Disadvantages of the Waterfall model**
• The waterfall model is simple and easy to understand, to implement, and use.	• This model is not good for complex and object oriented projects.
• All the requirements are known at the beginning of the project, hence it is easy to manage.	• In this model, the changes are not permitted so it is not fit for moderate to high risk changes in project.
• It avoids overlapping of phases because each phase is completed at once.	• It is a poor model for long duration projects
• This model works for small projects where the requirements are easily understood.	• The problems with this model are uncovered, until the software testing.
• This model is preferred for those projects where the quality is more important as compared to the cost of the project.	• The amount of risk is high.

2. V Model

- V model is known as Verification and Validation model.
- This model is an extension of the waterfall model.
- In the life cycle of V-shaped model, processes are executed sequentially.
- Every phase completes its execution before the execution of next phase begins.

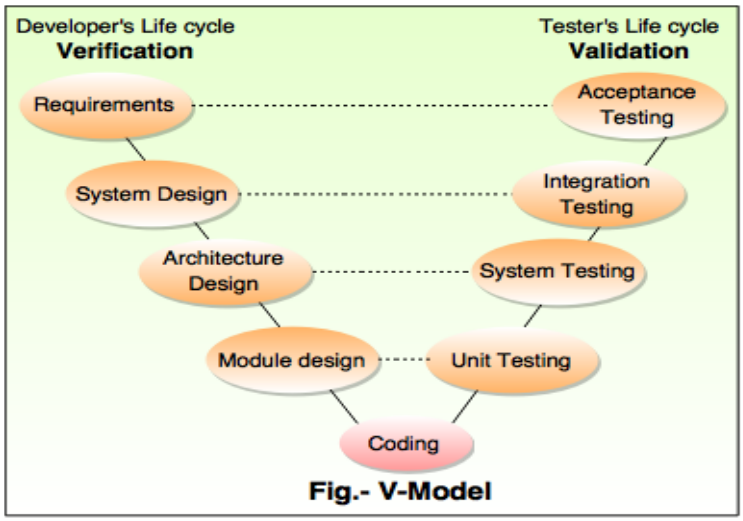


Fig.- V-Model

Following are the phases of V-model:

1. **Requirements**

- The requirements of product are understood from the customers point of view to know their exact requirement and expectation.
- The acceptance test design planning is completed at requirement stage because, business requirements are used as an input for acceptance testing.

2. **System Design**

- In system design, high level design of the software is constructed.
- In this phase, we study how the requirements are implemented their technical use.

3. **Architecture design**

- In architecture design, software architecture is created on the basis of high level design.
- The module relationship and dependencies of module, architectural diagrams, database tables, technology details are completed in this phase.

4. **Module design**

- In module phase, we separately design every module or the software components.
- Finalize all the methods, classes, interfaces, data types etc.
- Unit tests are designed in module design phase based on the internal module designs.
- Unit tests are the vital part of any development process. They help to remove the maximum faults and errors at an early stage.

5. **Coding Phase**

- The actual code design of module designed in the design phase is grabbed in the coding phase.
- On the basis of system and architecture requirements, we decide the best suitable programming language.
- The coding is executed on the basis of coding guidelines and standards.

Advantages of V-model

- V-model is easy and simple to use.
- Many testing activities i.e. planning and test design are executed in the starting, it saves more time.
- Calculation of errors is done at the starting of the project hence, less chances of error occurred at final phase of testing.
- This model is suitable for small projects where the requirements are easily understood.

Disadvantages of V-model

- V-model is not suitable for large and composite projects.
- If the requirements are not constant then this model is not acceptable.

3. **Incremental Model**

- Linear sequential model is not suited for projects which are iterative in nature Incremental model suits such projects
- Used when initial requirements are reasonably well-defined and compelling need to provide limited functionality quickly
- Functionality expanded further in later releases
- Software is developed in increments The Incremental Model
- The incremental model combines the elements of waterfall model and they are applied in an iterative fashion.
- The first increment in this model is generally a core product.
- Each increment builds the product and submits it to the customer for suggesting any modifications.
- The next increment implements the customer's suggestions and add additional requirements in the previous increment.
- This process is repeated until the product is completed. For example, the word-processing software is developed using the incremental model.

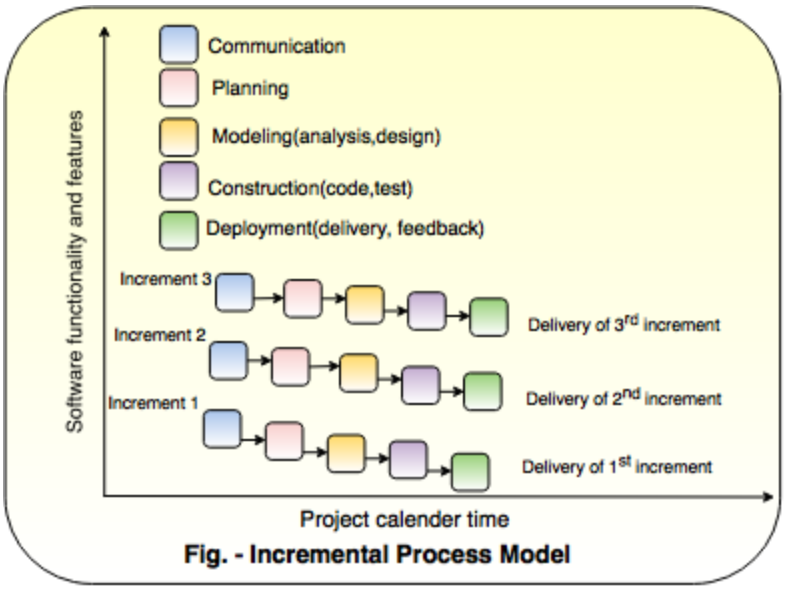


Fig. - Incremental Process Model

Following are the phases of Incremental model:

I. **Communication** The software development starts with the communication between customer and developer.

II. **Planning** It consists of complete estimation, scheduling for project development.

III. **Modeling**
   - Modeling consists of complete requirement analysis and the design of the project like algorithm, flowchart etc.
   - The algorithm is a step-by-step solution of the problem and the flow chart shows a complete flow diagram of a program.

IV. **Construction**
   - Construction consists of code generation and the testing part.
   - Coding part implements the design details using an appropriate programming language.
   - Testing is to check whether the flow of coding is correct or not.
   - Testing also checks that the program provides desired output.

V. **Deployment**
   - Deployment step consists of delivering the product to the customer and taking feedback from them.
   - If the customer wants some corrections or demands for the additional capabilities, then the change is required for improvement in the quality of the software.

Advantages of Incremental model	Disadvantages of the incremental model
• This model is flexible because the cost of development is low and initial product delivery is faster.	• The cost of the final product may cross the cost initially estimated.
• It is easier to test and debug in the smaller iteration.	• This model requires a very clear and complete planning.
• The working software is generated quickly in the software life cycle.	• The planning of design is required before the whole system is broken into smaller increments.
• The customers can respond to its functionalities after every increment.	• The demands of customer for the additional functionalities after every increment causes problem in the system architecture.

4. RAD Model

- RAD is a Rapid Application Development model.
- An incremental software process model
- Having a short development cycle
- High-speed adoption of the waterfall model using a component-based construction approach
- Creates a fully functional system within a very short span time of 60 to 90 days
- Using the RAD model, software products are developed in a short period of time.
- The initial activity starts with communication between customer and developer.
- Planning depends upon the initial requirements and then the requirements are divided into groups.
- Planning is more important to work together on different modules.

The RAD model consists of following phases:

1) Business Modelling
2) Data modelling
3) Process modelling
4) Application generation
5) Testing and turnover

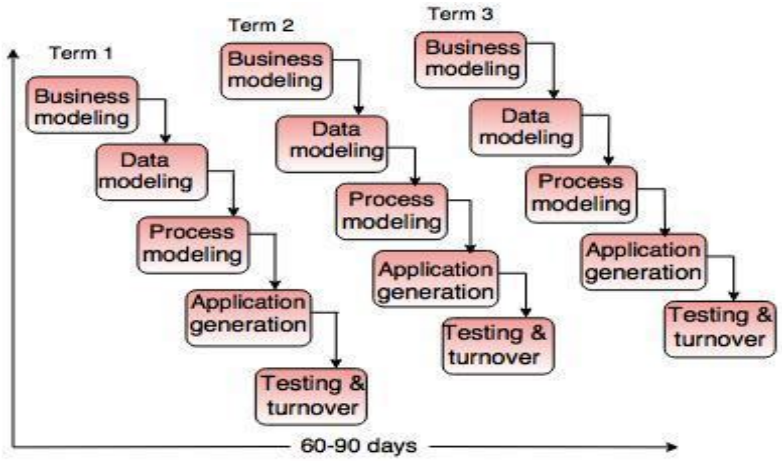


Fig. - RAD Model

1) **Business Modelling**
   - Business modelling consists of the flow of information between various functions in the project. **For example,** what type of information is produced by every function and which are the functions to handle that information.
   - It is necessary to perform complete business analysis to get the essential business information.
2) **Data modelling**
   - The information in the business modelling phase is refined into the set of objects and it is essential for the business.
   - The attributes of each object are identified and defined the relationship between objects.
3) **Process modelling**
   - The data objects defined in the data modelling phase are changed to fulfil the information flow to implement the business model.
   - The process description is created for adding, modifying, deleting or retrieving a data object.
4) **Application generation**
   - In the application generation phase, the actual system is built.
   - To construct the software the automated tools are used.
5) **Testing and turnover**
   - The prototypes are independently tested after each iteration so that the overall testing time is reduced.
   - The data flow and the interfaces between all the components are fully tested. Hence, most of the programming components are already tested.

Advantages of RAD Model	Disadvantages of RAD Model
• The process of application development and delivery are fast.	• The feedback from the user is required at every development phase.
• This model is flexible, if any changes are required.	• This model is not a good choice for long term and large projects.
• Reviews are taken from the clients at the staring of the development hence there are lesser chances of missing the requirements.	• Requires a number of RAD teams
	• Requires commitment from both developer and customer for rapid-fire completion of activities
	• Not suited when technical risks are high

5. Agile Model

The meaning of Agile is swift or versatile."Agile process model" refers to a software development approach based on iterative development. Agile methods break tasks into smaller iterations, or parts do not directly involve long term planning. The project scope and requirements are laid down at the beginning of the development process. Plans regarding the number of iterations, the duration and the scope of each iteration are clearly defined in advance.

- Agile model is a combination of incremental and iterative process models.
- This model focuses on the users satisfaction which can be achieved with quick delivery of the working software product.
- Agile model breaks the product into individual iterations.
- Every iteration includes cross functional teams working on different areas such as planning, requirements, analysis, design, coding, unit testing and acceptance testing.
- At the end of an iteration working product shows to the users.
- With every increment, features are incremented and the final increment hold all the features needed by the customers.
- The iterations in agile process are shorter in duration which can vary from 2 weeks to 2 months.

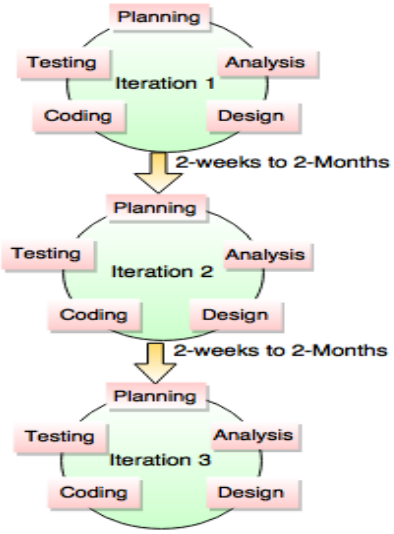


Fig. - Graphical Representation of Agile Model

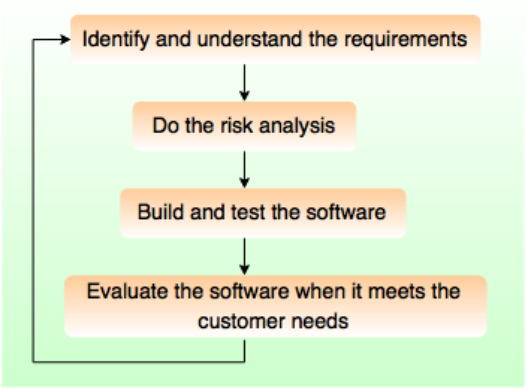
Advantages of Agile model	**Disadvantages of Agile model**
• Frequent Delivery	• Due to the shortage of formal documents, it creates confusion and crucial decisions taken throughout various phases can be misinterpreted at any time by different team members.
• Face-to-Face Communication with clients.	• Due to the lack of proper documentation, once the project is completed and the developers are allotted to another project, maintenance of the finished project can become a difficulty.
• Efficient design and fulfils the business requirement.	• It totally depends on customer interaction. If the customer is not clear with their requirements, the development team can go in the wrong direction.
• Anytime changes are acceptable.	
• It reduces total development time.	

6. **Iterative Model**
   - In Iterative model, the large application of software development is divided into smaller chunks and smaller parts of software which can be reviewed to recognize further requirements are implemented. This process is repeated to generate a new version of the software in each cycle of a model.
   - With every iteration, development module goes through the phases i.e requirement, design, implementation and testing. These phases are repeated in an iterative model in a sequence.
     1. **Requirement Phase** In this phase, the requirements for the software are assembled and analyzed. Generates a complete and final specification of requirements.
     2. **Design Phase** In this phase, a software solution meets the designed requirements which can be a new design or an extension of an earlier design.
     3. **Implementation and test phase** In this phase, coding for the software and test the code.
     4. **Evaluation** In this phase, software is evaluated, the current requirements are reviewed and the changes and additions in the requirements are suggested.

Advantages of an Iterative Model	Disadvantages of an Iterative Model
• Produces working software rapidly and early in the software life cycle.	• The system architecture is costly.
• This model is easy to test and debug in a smaller iteration.	• This model is not suitable for smaller projects.
• It is less costly to change scope and requirements.	

7. **Spiral model**
   - It is a combination of prototype and sequential or waterfall model.
   - This model was developed by Boehm.
   - It is used for generating software projects. This model is a risk driven process model.
   - Every phase in the Spiral model is start with a design goal and ends with the client review.
   - The development team in this model begins with a small set of requirements and for the set of requirements team goes through each development phase.
   - The development team adds the functionality in every spiral till the application is ready.

Following are the steps involved in spiral model:



Phases of Spiral model are:

1) Planning
2) Risk Analysis
3) Engineering
4) Evaluation

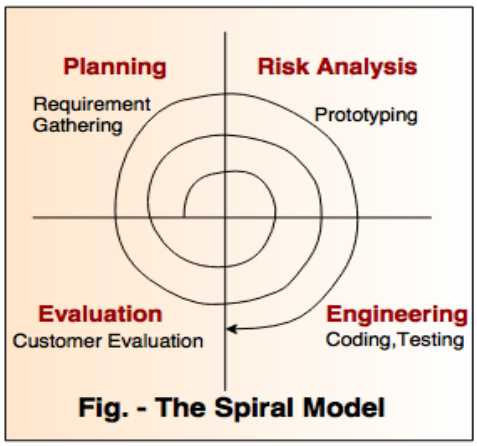


Fig. - The Spiral Model

1) Planning

- This phase, studies and collects the requirements for continuous communication between the customer and system analyst.
- It involves estimating the cost and resources for the iteration.

2) Risk Analysis

This phase, identifies the risk and provides alternate solutions if the risk is found.

3) Engineering

In this phase, actual development, i.e. coding of the software is completed. Testing is completed at the end of the phase.

4) Evaluation

Get the software evaluated by the customers. They provide feedback before the project continues to the next spiral.

Advantages of Spiral Model	Disadvantages of Spiral Model
• It reduces high amount of risk.	• It can be costly to develop a software model.
• It is good for large and critical projects.	• It is not used for small projects.
• It gives strong approval and documentation control.	
• In spiral model, the software is produced early in the life cycle process.	

8. Prototype Model:

- Prototype is defined as first or preliminary form using which other forms are copied or derived.
- Used when customer defines a set of objective but does not identify input, output, or processing requirements
- Developer is not sure of:
  – efficiency of an algorithm adaptability of an operating system
  – human/machine interaction
- Prototype model is a set of general objectives for software.
- It does not identify the requirements like detailed input, output.
- It is software working model of limited functionality.
- In this model, working programs are quickly produced.

The different phases of Prototyping model are:

1) Communication
2) Quick design
3) Modelling and quick design
4) Construction of prototype
5) Deployment, delivery, feedback

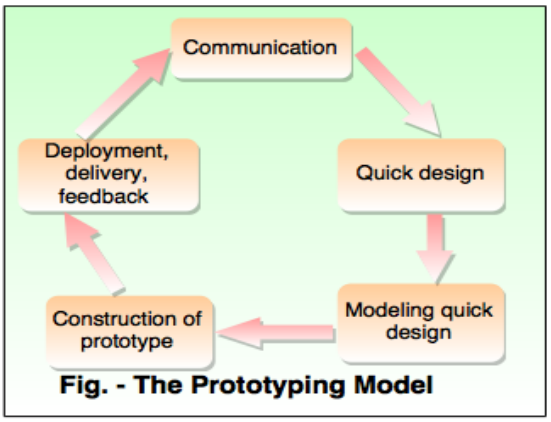


Fig. - The Prototyping Model

1. **Communication**
   In this phase, developer and customer meet and discuss the overall objectives of the software.

2. **Quick design**

- Quick design is implemented when requirements are known.
- It includes only the important aspects i.e input and output format of the software.
- It focuses on those aspects which are visible to the user rather than the detailed plan.
- It helps to construct a prototype.

3. **Modelling quick design**

- This phase gives the clear idea about the development of software as the software is now constructed.
- It allows the developer to better understand the exact requirements.

4. **Construction of prototype**
   The prototype is evaluated by the customer itself.

5. **Deployment, delivery, feedback**

- If the user is not satisfied with current prototype then it is refined according to the requirements of the user.
- The process of refining the prototype is repeated till all the requirements of users are met.
- When the users are satisfied with the developed prototype then the system is developed on the basis of final prototype.

Advantages of Prototyping Model

- In the development process of this model users are actively involved.
- The development process is the best platform to understand the system by the user.
- Earlier error detection takes place in this model.
- It gives quick user feedback for better solutions.
- It identifies the missing functionality easily. It also identifies the confusing or difficult functions.

Disadvantages of Prototyping Model

- The client involvement is more and it is not always considered by the developer.
- It is a slow process because it takes more time for development.
- Many changes can disturb the rhythm of the development team.
- It is a throw away prototype when the users are confused with it.

HOW TO CHOOSE THE BEST SDLC MODEL FOR THE PROJECT

Choosing the Best Software Development Life Cycle (SDLC) model for your project is a critical decision that can significantly impact the project's success. Each SDLC model has its own set of advantages, disadvantages, and suitability for different types of projects.

- **Choosing SDLC Model on the Basis of Project Requirements:**
    - **Clarity of Requirements:** If your project requirements are well-understood and unlikely to change, a traditional model like "Waterfall" may be suitable.
    - **Evolving Requirements:** For projects with evolving or unclear requirements, consider "Agile" or "Iterative" models.
- **Choosing SDLC Model on the Basis of Project Size and Complexity:**
    - **Small Projects:** "Waterfall" model is suitable for smaller projects with straightforward requirements.
    - **Large and Complex Projects:** "Agile", "Scrum", or other "iterative" models are often preferred for large and complex projects due to their adaptability.
- **Choosing SDLC Model on the Basis of Flexibility and Adaptability:**
    - **Flexibility Requirements:** If your project requires flexibility and the ability to adapt to changes, consider "**Agile"**, "**Scrum"**, or "**Iterative"** models.
    - **Fixed Requirements:** For projects with fixed and stable requirements, "**Waterfall"** may be appropriate.

- **Choosing SDLC Model on the Basis of Customer Involvement:**
    - **Customer Collaboration:** If continuous customer involvement and feedback are crucial, "**Agile",** "**Scrum"**, or "**Iterative"** models are suitable.
    - **Limited Customer Involvement:** For projects where customer involvement is limited, "**Waterfall"** might be sufficient.

- **Choosing SDLC Model on the Basis of Risk Tolerance:**
    - **High Risk Tolerance:** If your project can accommodate higher levels of risk and uncertainty, consider "**Agile" , "Spiral"** or "**Iterative"** models.
    - **Low Risk Tolerance:** For projects with low risk tolerance and a need for predictability, "**Waterfall"** might be more appropriate.

- **Choosing SDLC Model on the Basis of Time Constraints:**
    - **Strict Deadlines:** If your project has strict deadlines and fixed timelines, "**Waterfall"** may help with better planning.
    - **Flexibility in Timelines:** "**Agile"** and "**Iterative"** models offer flexibility in timelines and can adapt to changing priorities.

- **Choosing SDLC Model on the Basis of Team Expertise:**
    - **Cross-Functional Teams:** "**Agile"** and "**Scrum"** work well with cross-functional teams.
    - **Specialized Teams:** "**Waterfall"** may suit projects where teams have specialized roles.
- **Choosing SDLC Model on the Basis of Client Involvement and Approval:**
    - **Periodic Client Approval:** "**Iterative"** or "**Agile"** models involve periodic client approval, ensuring client expectations are met.
    - **Final Approval at End:** "**Waterfall"** involves client approval mainly at the end of the project.

- **Choosing SDLC Model on the Basis of Regulatory Compliance:**
    - **Stringent Regulations:** If your project requires stringent regulatory compliance, "**Waterfall"** may be preferable for its documentation and traceability.

- **Pros and Cons of each SDLC Model with Comparison Table:**

Here's a table comparing multiple Software Development Life Cycle (SDLC) models with selection criteria.

SDLC Model	Selection Criteria	Advantages	Disadvantages
Waterfall Model	Well-defined and stable requirements	Simple and easy to understand	Lack of flexibility, no room for changes
Iterative Model	Large and complex projects	Allows for incremental development	Requires good planning and design upfront
Incremental Model	Deliver partial implementations	Early, partial product releases	Dependencies between components
V-Model (Verification and Validation)	Emphasis on verification and validation	Each phase has specific deliverables	Can be rigid and difficult to accommodate changes
Spiral Model	High-risk projects	Incorporates risk analysis and management	Complex and may require expertise
Agile Model	Dynamic and rapidly changing requirements	Customer involvement throughout the development	Requires experienced and self-disciplined team
Scrum Model	Projects with a clear product backlog	Regular reviews and adaptations	Can be challenging for large teams

- **Choosing SDLC Model if Project specifics are unknown (General Model Recommendation)**

  If specific details about the project are not known or if flexibility is a priority, the "**Agile Model"** is often a good general choice. Agile allows for adaptive planning, iterative development, and customer collaboration, making it suitable for a wide range of project scenarios.

It's important to note that the choice of an SDLC model depends on various factors, including project size, complexity, budget, timeline, and client requirements. The table provides a high-level overview, and the actual decision-making process involves a detailed analysis of these factors.

Choosing the best SDLC model involves a careful analysis of your project's unique characteristics, requirements, and constraints. It's often beneficial to involve key stakeholders, including project managers, developers, and clients, in the decision-making process. Additionally, consider the possibility of hybrid models that combine elements from different SDLC approaches to meet specific project needs. Regularly reassess and adapt the chosen model as the project progresses to ensure alignment with changing requirements and priorities

REQUIREMENT GATHERING AND ANALYSIS

The analyst starts requirements gathering and analysis activity by collecting all information from the customer which could be used to develop the requirements of the system. The following basic questions pertaining to the project should be clearly understood by the analyst in order to obtain a good grasp of the problem:

- What is the problem?
- Why is it important to solve the problem?
- What are the possible solutions to the problem?
- What exactly are the data input to the system and what exactly are the data output by the system?
- What are the likely complexities that might arise while solving the problem?
- If there are external software or hardware with which the developed software has to interface, then what exactly would the data interchange formats with the external system be?

1. **Requirement Gathering:**

   A requirement gathering is the process of identifying your project's exact requirements from start to finish. This process occurs during the project initiation phase but you'll continue to manage your project requirements throughout the entire project timeline.

   The following are some of the well-known requirements gathering techniques –

   Brainstorming

   - Brainstorming is used in requirement gathering to get as many ideas as possible from group of people. Generally used to identify possible solutions to problems, and clarify details of opportunities.

   Document Analysis

   - Reviewing the documentation of an existing system can help when creating AS–IS process document, as well as driving gap analysis for scoping of migration projects. In an ideal world, we would even be reviewing the requirements that drove creation of the existing system – a starting point for documenting current requirements. Nuggets of information are often buried in existing documents that help us ask questions as part of validating requirement completeness.

Focus Group

- A focus group is a gathering of people who are representative of the users or customers of a product to get feedback. The feedback can be gathered about needs/opportunities/ problems to identify requirements, or can be gathered to validate and refine already elicited requirements. This form of market research is distinct from brainstorming in that it is a managed process with specific participants.

Interface analysis

- Interfaces for a software product can be human or machine. Integration with external systems and devices is just another interface. User centric design approaches are very effective at making sure that we create usable software. Interface analysis – reviewing the touch points with other external systems is important to make sure we don't overlook requirements that aren't immediately visible to users.

Interview

- Interviews of stakeholders and users are critical to creating the great software. Without understanding the goals and expectations of the users and stakeholders, we are very unlikely to satisfy them. We also have to recognize the perspective of each interviewee, so that, we can properly weigh and address their inputs. Listening is the skill that helps a great analyst to get more value from an interview than an average analyst.

Observation

- By observing users, an analyst can identify a process flow, steps, pain points and opportunities for improvement. Observations can be passive or active (asking questions while observing). Passive observation is better for getting feedback on a prototype (to refine requirements), where active observation is more effective at getting an understanding of an existing business process. Either approach can be used.

Prototyping

- Prototyping is a relatively modern technique for gathering requirements. In this approach, you gather preliminary requirements that you use to build an initial version of the solution - a prototype. You show this to the client, who then gives you additional requirements. You change the application and cycle around

with the client again. This repetitive process continues until the product meets the critical mass of business needs or for an agreed number of iterations.

Requirement Workshops

- Workshops can be very effective for gathering requirements. More structured than a brainstorming session, involved parties collaborate to document requirements. One way to capture the collaboration is with creation of domain-model artifacts (like static diagrams, activity diagrams). A workshop will be more effective with two analysts than with one.

Reverse Engineering

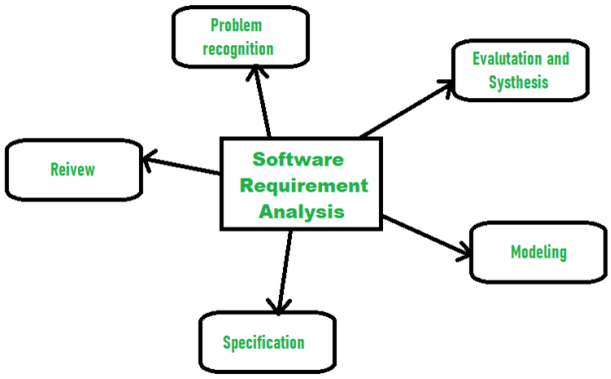
- When a migration project does not have access to sufficient documentation of the existing system, reverse engineering will identify what the system does. It will not identify what the system should do, and will not identify when the system does the wrong thing.

Survey/Questionnaire

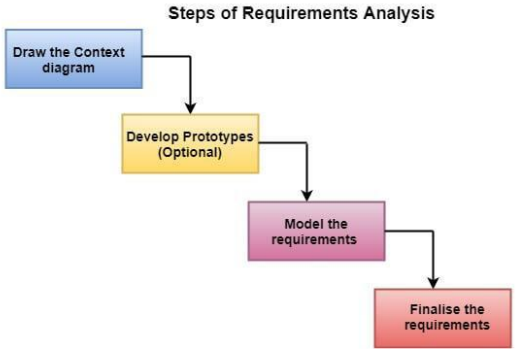
- When collecting information from many people – too many to interview with budget and time constraints – a survey or questionnaire can be used. The survey can force users to select from choices, rate something ("Agree Strongly, agree…"), or have open ended questions allowing free-form responses. Survey design is hard – questions can bias the respondents.

2. **Requirement Analysis:**

Software requirement analysis simply means complete study, analyzing, and describing software requirements so that requirements that are genuine and needed can be fulfilled to solve problem. There are several activities involved in analyzing Software requirements. Some of them are given below

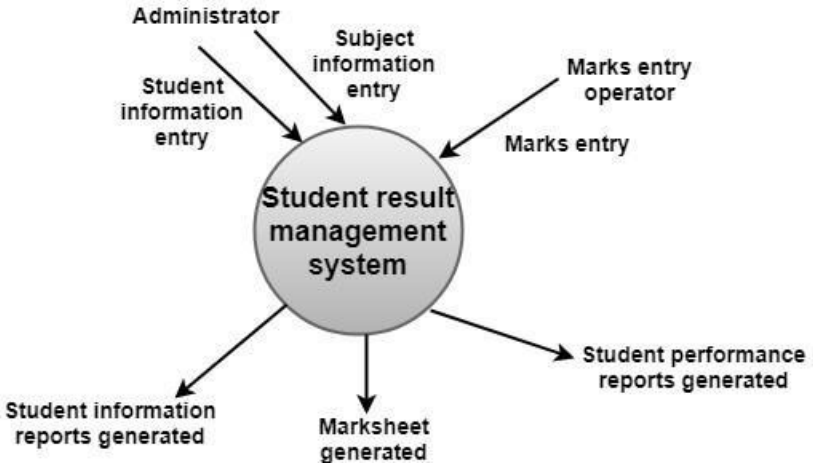


The various steps of requirement analysis are shown in fig:



(i)Draw the context diagram:

The context diagram is a simple model that defines the boundaries and interfaces of the proposed systems with the external world. It identifies the entities outside the proposed system that interact with the system. The context diagram of student result management system is given below:



(ii) Development of a Prototype (optional):

One effective way to find out what the customer wants is to construct a prototype, something that looks and preferably acts as part of the system they say they want.

We can use their feedback to modify the prototype until the customer is satisfied continuously. Hence, the prototype helps the client to visualize the proposed system and increase the understanding of the requirements. When developers and users are not sure about some of the elements, a prototype may help both the parties to take a final decision.

Some projects are developed for the general market. In such cases, the prototype should be shown to some representative sample of the population of potential purchasers. Even

though a person who tries out a prototype may not buy the final system, but their feedback may allow us to make the product more attractive to others.

The prototype should be built quickly and at a relatively low cost. Hence it will always have limitations and would not be acceptable in the final system. This is an optional activity.

(iii) Model the requirements: This process usually consists of various graphical representations of the functions, data entities, external entities, and the relationships between them. The graphical view may help to find incorrect, inconsistent, missing, and superfluous requirements. Such models include the Data Flow diagram, Entity-Relationship diagram, Data Dictionaries, State-transition diagrams, etc.

(iv) Finalize the requirements: After modeling the requirements, we will have a better understanding of the system behavior. The inconsistencies and ambiguities have been identified and corrected. The flow of data amongst various modules has been analyzed. Elicitation and analyze activities have provided better insight into the system. Now we finalize the analyzed requirements, and the next step is to document these requirements in a prescribed format.

FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS:

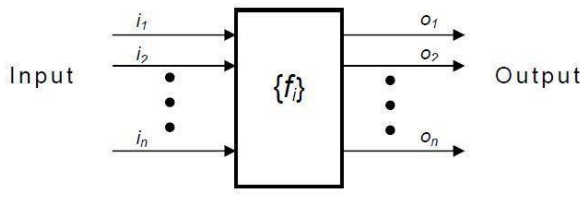
Functional requirements: -

These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract.

These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.

Example:

- What are the features that we need to design for this system?
- What are the edge cases we need to consider, if any, in our design?



View of a system performing a set of functions

Nonfunctional requirements:-

These are the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to another. They are also called non-behavioral requirements. They deal with issues like:

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

Example:

- Each request should be processed with the minimum latency?
- System should be highly valuable.

Difference between Functional Requirements and Non-Functional Requirements

Functional Requirements	Non Functional Requirements
A functional requirement defines a system or its component.	A non-functional requirement defines the quality attribute of a software system.
It specifies "What should the software system do?"	It places constraints on "How should the software system fulfill the functional requirements?"
Functional requirement is specified by User.	Non-functional requirement is specified by technical peoples e.g. Architect, Technical leaders and software developers.
It is mandatory.	It is not mandatory.
It is captured in use case.	It is captured as a quality attribute.
Defined at a component level.	Applied to a system as a whole

Goals of implementation:-

The goals of implementation part documents some general suggestions regarding development. These suggestions guide trade-off among design goals. The goals of implementation section might document issues such as revisions to the system functionalities that may be required in the future, new devices to be supported in the future, reusability issues, etc. These are the items which the developers might keep in their mind during development so that the developed system may meet some aspects that are not required immediately.

Identifying functional requirements from a problem description

The high-level functional requirements often need to be identified either from an informal problem description document or from a conceptual understanding of the problem. Each high- level requirement characterizes a way of system usage by some user to perform some meaningful piece of work. There can be many types of users of a system and their requirements from the system may be very different. So, it is often useful to identify the different types of users who might use the system and then try to identify the requirements from each user's perspective.

Example: - Consider the case of the library System, where **–**

F1: Search Book function

Input: an author's name

Output: details of the author's books and the location of these books in the library

So the function Search Book (F1) takes the author's name and transforms it into book details.

Functional requirements actually describe a set of high-level requirements, where each high-level requirement takes some data from the user and provides some data to the user as an output. Also each high-level requirement might consist of several other functions.

Documenting functional requirements

For documenting the functional requirements, we need to specify the set of functionalities supported by the system. A function can be specified by identifying the state at which the data is to be input to the system, its input data domain, the output data domain, and the type of processing to be carried on the input data to obtain the output data. Let us first try to document the withdraw-cash function of an ATM (Automated Teller Machine) system. The withdraw-cash is a high-level requirement. It has several sub-requirements

corresponding to the different user interactions. These different interaction sequences capture the different scenarios.

Example: - Withdraw Cash from ATM

R1: withdraw cash

Description: The withdraw cash function first determines the type of account that the user has and the account number from which the user wishes to withdraw cash. It checks the balance to determine whether the requested amount is available in the account. If enough balance is available, it outputs the required cash; otherwise it generates an error message.

R1.1 select withdraw amount option

Input: “withdraw amount” option

Output: user prompted to enter the account type

R1.2: select account type

Input: user option

Output: prompt to enter amount

R1.3: get required amount

Input: amount to be withdrawn in integer values greater than 100 and less than 10,000 in multiples of 100.

Output: The requested cash and printed transaction statement.

Processing: the amount is debited from the user’s account if sufficient balance is available, otherwise an error message displayed

SOFTWARE REQUIREMENT SPECIFICATION (SRS):

A software requirements specification (SRS) is a document that captures complete description about how the system is expected to perform. It is usually signed off at the end of requirements engineering phase.

Qualities of SRS:

- Correct
- Unambiguous
- Complete
- Consistent

- Ranked for importance and/or stability
- Verifiable
- Modifiable
- Traceable

Types of Requirements:

The below diagram depicts the various types of requirements that are captured during SRS.

Parts of a SRS document:

The important parts of SRS document are:

a. Functional requirements of the system
b. Non-functional requirements of the system, and
c. Goals of implementation

Properties of a good SRS document:

The important properties of a good SRS document are the following:

- **Concise.** The SRS document should be concise and at the same time unambiguous, consistent, and complete. Verbose and irrelevant descriptions reduce readability and also increase error possibilities.
- **Structured.** It should be well-structured. A well-structured document is easy to understand and modify. In practice, the SRS document undergoes several revisions to cope up with the customer requirements. Often, the customer requirements

evolve over a period of time. Therefore, in order to make the modifications to the SRS document easy, it is important to make the document well-structured.

- **Black-box view.** It should only specify what the system should do and refrain from stating how to do these. This means that the SRS document should specify the external behavior of the system and not discuss the implementation issues. The SRS document should view the system to be developed as black box, and should specify the externally visible behavior of the system. For this reason, the SRS document is also called the black-box specification of a system.
- **Conceptual integrity.** It should show conceptual integrity so that the reader can easily understand it.
- **Response to undesired events.** It should characterize acceptable responses to undesired events. These are called system response to exceptional conditions.
- **Verifiable.** All requirements of the system as documented in the SRS document should be verifiable. This means that it should be possible to determine whether or not requirements have been met in an implementation.

Problems without a SRS document:

The important problems that an organization would face if it does not develop a SRS document are as follows:

- Without developing the SRS document, the system would not be implemented according to customer needs.
- Software developers would not know whether what they are developing is what exactly required by the customer.
- Without SRS document, it will be very much difficult for the maintenance engineers to understand the functionality of the system.
- It will be very much difficult for user document writers to write the users' manuals properly without understanding the SRS document.

IEEE 830 GUIDELINES:

This is a standard format of writing the SRS document. Most developers use the standard and of course, there are small variations to the standard; but if we know the IEEE 830 standard should be able to have small variations of this. Now let us look at what are the sections of the IEEE 830 standard title that is name of the software table of contents and then, there are 3 main sections; one is the Introduction, Overall Description and the Specific Requirements and then, there are the Appendices and Index.

The introduction should have a purpose; subsection called as Purpose, Scope, Definition, Acronym, Abbreviations, References and Overview. The purpose should say that what the

system will solve. So, general introduction and describe who are the users. Scope very overall idea about what the system will do; what it will not do, very crisply written and overall description of what is expected of the system and what it should not do.

Then, there are the various terminologies; the definitions, acronyms, abbreviations that will be used in the SRS document. But, often variation to the standard, these are put under the appendices. Similarly, a setup documents that are referred in the SRS documents. These references can also be put under a separate appendices and then, an overview of the document or organization of the document; how the SRS is organized and that is about the introduction. Now, let us look at the overall description. This is the section 2. The sub sections here are the Product Perspective, Product Functions, User Characteristics, Constraints, Assumptions and Dependencies. In the product perspective, we write here the business case that is how it will help the customer; how it will benefit how it will be used and so on and even met write about the external interfaces. Briefly the overview of what type of users, hardware, software etcetera will be used and then, any constraints in the memory are operational constraints site constraints and so on.

The product functions this not really the functional requirement specification; but a brief overview or a summer summarize a summary of the functional capabilities. For example, in a library, we might just write very briefly saying that the library software should serve the users for their issuing of books returning for the librarian to create members manage books and so on and for the accountant to manage the financial aspects of the library. The user characteristics are the technical skills of each user class. This is also required because finally, the user interface development we will need the user characteristics. If it is software is to be used by factory workers as well and programmers, we need to have different user interfaces. So, need to identify the user characteristics and how much what are the technical skills how will their convergent with the computers and so on. The constraints here, what are the platforms that will be used? The database that will be used; the network development standards and so on.

And then, if there are any assumptions here dependencies need to mention that and then let us look at the third section. This is possibly the most important section. Here we have the external interfaces functions; this is the major section here. This contains the functionalities the functional requirement. The non-functional requirement Logical Database Requirement, Design Constraints, Quality Attributes and any Object Oriented Models; in this section has to be written very carefully and in sufficient detail. So, that the designers can have all the information they need to carry out the development work and for the testers to be able to write the test cases. It is a very important section. While writing the functional requirement,

we should make sure that we have as far as possible all the input or stimulus to the system; output that is produced by the system and the functionalities.

Now, first let us look at the external interface. Here, we have to write about all the user interface maybe you can give a GUI screenshots, the different file formats that may be needed. It should actually there are some part that were mentioned in the section 2; external interface. It should actually elaborate that and should not duplicate information there. This is the main section the functional requirements. This will contain the detailed specification of each functionality or use case including collaboration and other diagrams we will see these diagrams later.

Then, the Performance Requirements; the Logical Database Requirement; what are the different data that will be stored in the database the relationship? The Design Constraints are standard complaints. For example, we may say that I have to use UML 2.0 and have to use let us say (Refer Time: 16:08) for writing the code. The quality attributes and then any object oriented models like class diagram, state collaboration diagram, activity diagram etcetera can be included.

But then, section 3 that we just saw now is just one example and IEE 830 specifies that there can be different organization of the section 3 based on the modes of the software that is software maybe having a expert mode, a novies mode and so on. If there are modes are there then, we might have to organize it such that the modes the software is written in terms of the modes. Sorry, the SRS document is written in terms of the modes. Similarly, depending on different user Classes, Concepts, Features, Stimuli; there can be slightly different organization of section three.

DECISION TABLES:

A decision table is used to represent the complex processing logic in a tabular or a matrix form. The upper rows of the table specify the variables or conditions to be evaluated. The lower rows of the table specify the actions to be taken when the corresponding conditions are satisfied. A column in a table is called a rule. A rule implies that if a condition is true, then the corresponding action is to be executed.

Example: -

Consider the previously discussed LMS example. The following decision table (bellow fig) shows how to represent the LMS problem in a tabular form. Here the table is divided into two parts, the upper part shows the conditions and the lower part shows what actions are taken. Each column of the table is a rule.

Conditions				
Valid selection	No	Yes	Yes	Yes
New member	-	Yes	No	No
Renewal	-	No	Yes	No
Cancellation	-	No	No	Yes
Actions				
Display error message	x	-	-	-
Ask member's details	-	x	-	-
Build customer record	-	x	-	-
Generate bill	-	x	x	-
Ask member's name & membership number	-	-	x	x
Update expiry date	-	-	x	-
Print cheque	-	-	-	x
Delete record	-	-	-	x

Decision table for LMS

From the above table you can easily understand that, if the valid selection condition is false then the action taken for this condition is 'display error message'. Similarly, the actions taken for other conditions can be inferred from the table.

DECISION TREE

A decision tree gives a graphic view of the processing logic involved in decision making and the corresponding actions taken. The edges of a decision tree represent conditions and the leaf nodes represent the actions to be performed depending on the outcome of testing the condition.

Example: -

Consider Library Membership Automation Software (LMS) where it should support the following three options**:**

- **New member**
- **Renewal**
- **Cancel membership**

New member option-

Decision: When the 'new member' option is selected, the software asks details about the member like the member's name, address, phone number etc.

Action: If proper information is entered then a membership record for the member is created and a bill is printed for the annual membership charge plus the security deposit payable.

Renewal option-

Decision: If the 'renewal' option is chosen, the LMS asks for the member's name and his membership number to check whether he is a valid member or not.

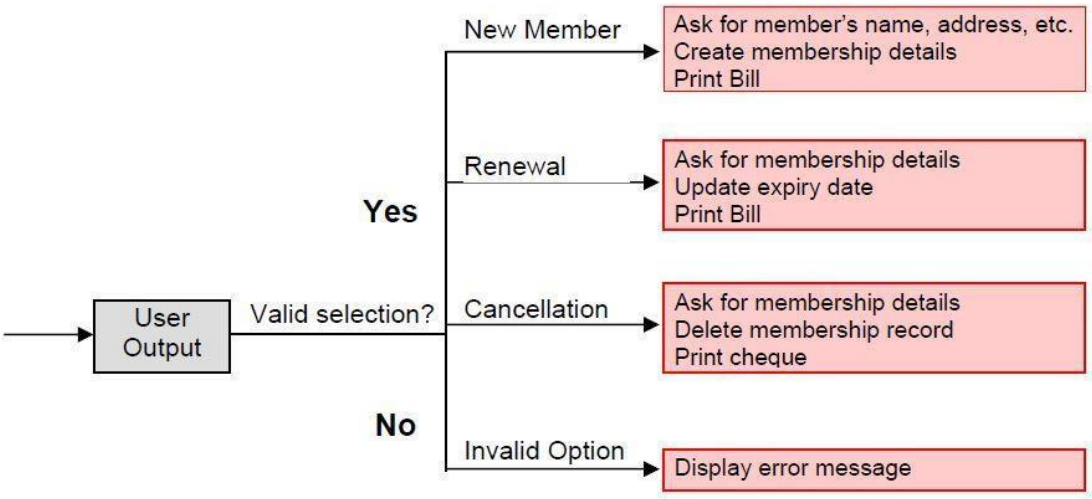
Action: If the membership is valid then membership expiry date is updated and the annual membership bill is printed, otherwise an error message is displayed.

Cancel membership option-

Decision: If the 'cancel membership' option is selected, then the software asks for member's name and his membership number.

Action: The membership is cancelled, a cheque for the balance amount due to the member is printed and finally the membership record is deleted from the database.

The following tree (fig) shows the graphical representation of the above example.



Decision Tree of LMS