Software Requirement Specifications

The production of the requirements stage of the software development process is **Software Requirements Specifications (SRS)** (also called a **requirements document**). This report lays a foundation for software engineering activities and is constructing when entire requirements are elicited and analyzed. **SRS** is a formal report, which acts as a representation of software that enables the customers to review whether SRS is according to their requirements. Also, it comprises user requirements for a system as well as detailed specifications of the system requirements.

The SRS is a specification for a specific software product, program, or set of applications that perform particular functions in a specific environment. It serves several goals depending on who is writing it. First, the SRS could be written by the client of a system. Second, the SRS could be written by a developer of the system. The two methods create entirely various situations and establish different purposes for the document altogether. The first case, SRS, is used to define the needs and expectation of the users. The second case, SRS, is written for various purposes and serves as a contract document between customer and developer.

Characteristics of good SRS

Following are the features of a good SRS document:

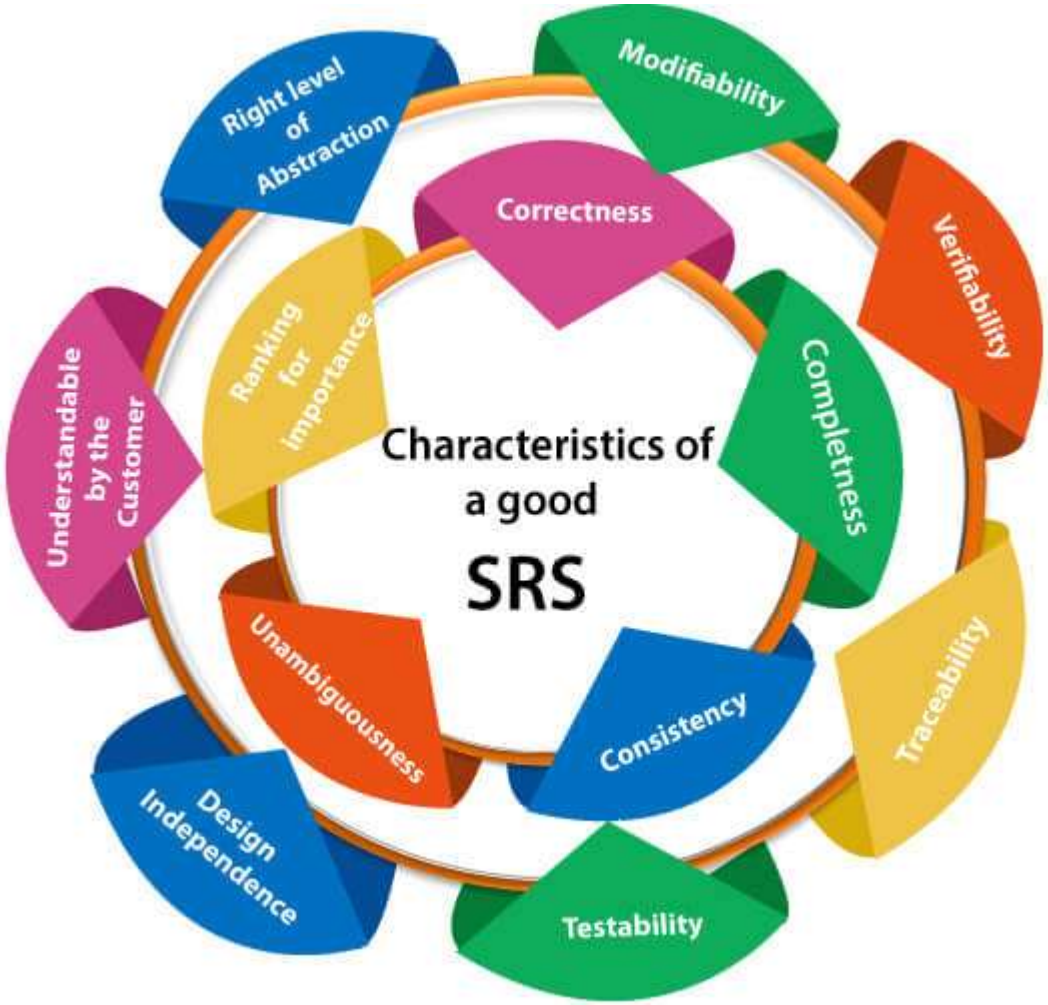
1. Correctness: User review is used to provide the accuracy of requirements stated in the SRS. SRS is said to be perfect if it covers all the needs that are truly expected from the system.

2. Completeness: The SRS is complete if, and only if, it includes the following elements:

(a). All essential requirements, whether relating to functionality, performance, design, constraints, attributes, or external interfaces.

(b). Definition of their responses of the software to all realizable classes of input data in all available categories of situations.

(c). Full labels and references to all figures, tables, and diagrams in the SRS and definitions of all terms and units of measure.



3. Consistency: The SRS is consistent if, and only if, no subset of individual requirements described in its conflict. There are three types of possible conflict in the SRS:

(1). The specified characteristics of real-world objects may conflicts. For example,

(a) The format of an output report may be described in one requirement as tabular but in another as textual.

(b) One condition may state that all lights shall be green while another states that all lights shall be blue.

(2). There may be a reasonable or temporal conflict between the two specified actions. For example,

(a) One requirement may determine that the program will add two inputs, and another may determine that the program will multiply them.

(b) One condition may state that "A" must always follow "B," while other requires that "A and B" co-occurs.

(3). Two or more requirements may define the same real-world object but use different terms for that object. For example, a program's request for user input may be called a "prompt" in one requirement's and a "cue" in another. The use of standard terminology and descriptions promotes consistency.

4. Unambiguousness: SRS is unambiguous when every fixed requirement has only one interpretation. This suggests that each element is uniquely interpreted. In case there is a method used with multiple definitions, the requirements report should determine the implications in the SRS so that it is clear and simple to understand.

5. Ranking for importance and stability: The SRS is ranked for importance and stability if each requirement in it has an identifier to indicate either the significance or stability of that particular requirement.

Typically, all requirements are not equally important. Some prerequisites may be essential, especially for life-critical applications, while others may be desirable. Each element should be identified to make these differences clear and explicit. Another way to rank requirements is to distinguish classes of items as essential, conditional, and optional.

6. Modifiability: SRS should be made as modifiable as likely and should be capable of quickly obtain changes to the system to some extent. Modifications should be perfectly indexed and cross-referenced.

7. Verifiability: SRS is correct when the specified requirements can be verified with a cost-effective system to check whether the final software meets those requirements. The requirements are verified with the help of reviews.

8. Traceability: The SRS is traceable if the origin of each of the requirements is clear and if it facilitates the referencing of each condition in future development or enhancement documentation.

There are two types of Traceability:

1. Backward Traceability: This depends upon each requirement explicitly referencing its source in earlier documents.

2. Forward Traceability: This depends upon each element in the SRS having a unique name or reference number.

The forward traceability of the SRS is especially crucial when the software product enters the operation and maintenance phase. As code and design document is modified, it is necessary to be able to ascertain the complete set of requirements that may be concerned by those modifications.

9. Design Independence: There should be an option to select from multiple design alternatives for the final system. More specifically, the SRS should not contain any implementation details.

10. Testability: An SRS should be written in such a method that it is simple to generate test cases and test plans from the report.

11. Understandable by the customer: An end user may be an expert in his/her explicit domain but might not be trained in computer science. Hence, the purpose of formal notations and symbols should be avoided too as much extent as possible. The language should be kept simple and clear.

12. The right level of abstraction: If the SRS is written for the requirements stage, the details should be explained explicitly. Whereas, for a feasibility study, fewer analysis can be used. Hence, the level of abstraction modifies according to the objective of the SRS.

Properties of a good SRS document

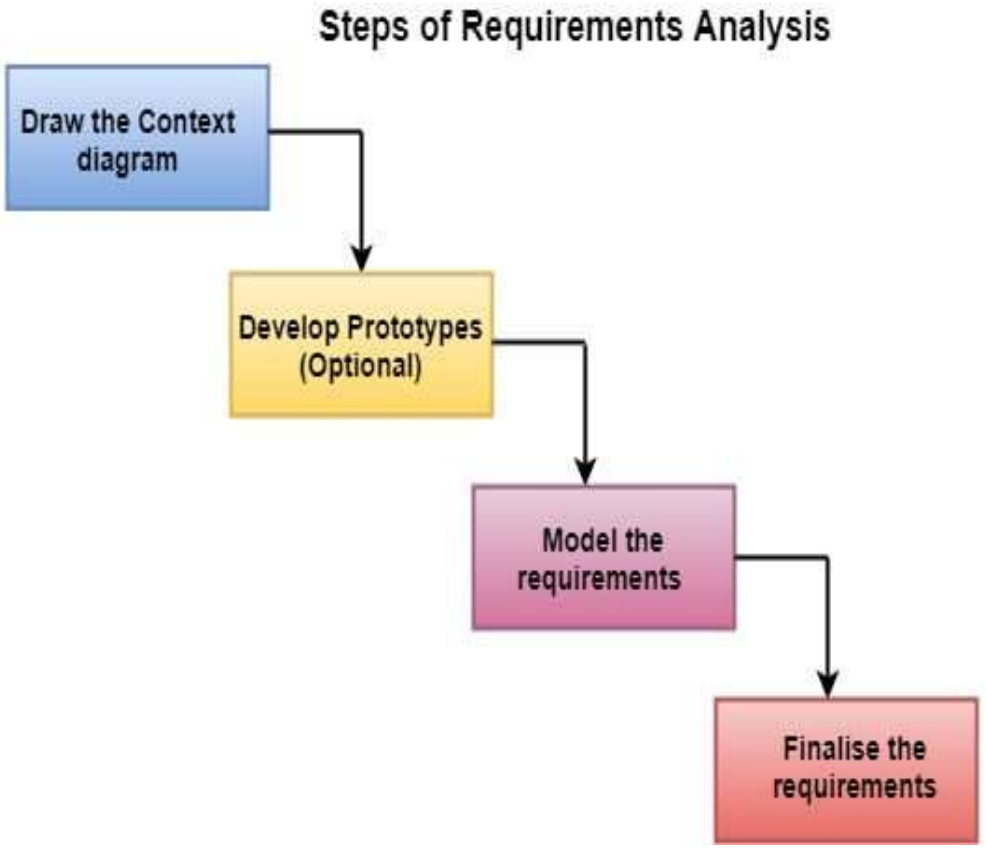
The essential properties of a good SRS document are the following:

- **Concise:** The SRS report should be concise and at the same time, unambiguous, consistent, and complete. Verbose and irrelevant descriptions decrease readability and also increase error possibilities.
- **Structured:** It should be well-structured. A well-structured document is simple to understand and modify. In practice, the SRS document undergoes several revisions to cope up with the user requirements. Often, user requirements evolve over a period of time. Therefore, to make the modifications to the SRS document easy, it is vital to make the report well-structured.
- **Black-box view:** It should only define what the system should do and refrain from stating how to do these. This means that the SRS document should define the external behavior of the system and not discuss the implementation issues. The SRS report should view the system to be developed as a black box and should define the externally visible behavior of the system. For this reason, the SRS report is also known as the black-box specification of a system.
- **Conceptual integrity:** It should show conceptual integrity so that the reader can merely understand it. Response to undesired events: It should characterize acceptable responses to unwanted events. These are called system response to exceptional conditions.
- **Verifiable:** All requirements of the system, as documented in the SRS document, should be correct. This means that it should be possible to decide whether or not requirements have been met in an implementation.

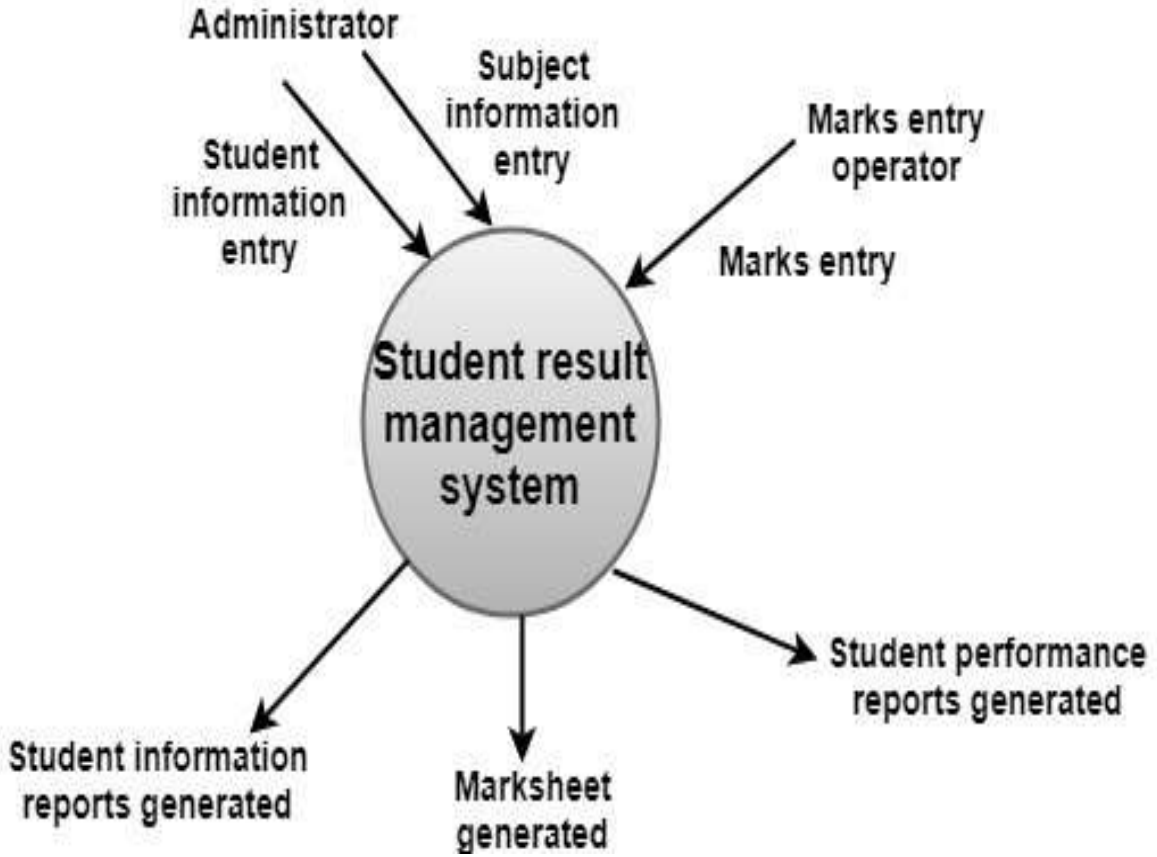
Requirements Analysis

Requirement analysis is significant and essential activity after elicitation. We analyze, refine, and scrutinize the gathered requirements to make consistent and unambiguous requirements. This activity reviews all requirements and may provide a graphical view of the entire system. After the completion of the analysis, it is expected that the understandability of the project may improve significantly. Here, we may also use the interaction with the customer to clarify points of confusion and to understand which requirements are more important than others.

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 drawn aside:



(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) Finalise 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.

Requirements analysis process

Follow these steps to complete a software requirements analysis:

1. Gather the requirements

To begin the requirements analysis process, communicate with users to gather the requirements. This phase is also known as "eliciting requirements." Analysts can use different techniques to gather the requirements, including:

- Conducting interviews
- Observing the workplace
- Holding focus groups or workshops
- Creating requirements lists

Customers should agree that the requirements document describes a product that meets their needs.

2. Analyze the requirements

In this phase, evaluate system feasibility, and confirm with the quality assurance team that the requirements are testable. The developers make sure the requirements are achievable and understandable. Then you can determine if the listed requirements are contradictory, incomplete, unclear or ambiguous, and solve those problems.

The goal of this phase is to decompose, analyze and detail the requirements across the system's design. Here are the attributes of good requirements to help you study and define your list:

- Unique
- Necessary
- Consistent
- Clear and concise
- Able to be validated
- Complete
- Technically feasible
- Traceable
- Verifiable
- Quantifiable
- Operationally effective

3. Improve the quality of the requirements

Next, you can improve the requirements' quality using these methods:

- **Visualization:** Use tools such as visualization and simulation that allow stakeholders to better understand the desired end product.
- **Document dependencies:** Document relationships and dependencies among requirements and any assumptions.
- **Consistent use of templates:** Produce consistent templates and models to document the requirements.

4. Model the requirements

In this phase, it’s time to create models of the requirements, which help stakeholders and customers visualize the potential system. Then you can present the requirements using flowcharts, graphs or models to ensure the system corresponds to the business' needs.

5. Document and review the requirements

Lastly, you can record the requirements in a document that is easy for both developers and users to understand. Note the changes your team makes to the requirements throughout the process. You can document the requirements in various formats, including:

- Software requirements specification (SRS)
- User cases
- Natural-language documents
- User stories
- Process specification

Requirements Analysis Problems

Problem 1: Customers Don’t Know What They Want

This is often true because much of development has to do with technology that’s beyond the customer’s knowledge. In terms of Web design, understanding the customer’s mission, vision, goals, and other important parts of the project. In software development — especially large and complex software with many interfaces — requirements don’t always affect customers. Requirements often focus on the back-end, processing and system interfaces. This is over marketing’s head.

What works is sending an email (BEFORE requirements discussions begin) to one contact from each major group that has ever been affected by the software. The contacts reply whether or not they need to be involved in the project. If they do, then they should attend the requirements meetings. What about the problem of having too many people in the meetings? That’s where having one contact helps. That contact should be communicating with the group.

Problem 2: Requirements Change During the Project

Always. I don’t think I can recall a project where this didn’t happen. Even for this little Web site. Anyone involved in development should have a change request process in place, even a one-person business. Accept that there will be changes and prepare a change request when this happens. Show the customer how it affects the milestones and get sign off. Another way is to have a phase 1 or soft launch and then add the new requirements for phase 2.

Problem 3: Timeline Trouble

I worked on a project that was supposed to launch in July 2006. Still working on it. The customer accepts responsibility for the delay. Be realistic. Map out the timeline based on an analysis of the requirements. If it's tight leaving no room for error or impossible — communicate this. Which would you rather have? No client because you said the timeline wasn't doable or having a client and missing deadlines that could hurt a company's reputation?

Problem 4: Communication Gaps

This is a simple thing that shouldn't happen, but it does. A software update impacted a group that was left out of requirements. As a result, it cost more to fix it later in the project. 37signals' Basecamp or a similar tool is a great way to record all communications between customer and designer. Basecamp is a web-based application that sends emails to affected parties along with a link to the message for replying. At the least, have one assigned point of contact for the customer and for the Web design firm. Ensure that contact is a reliable person and can do the job. No company wants to assign a contact whose answers will mostly be, "I don't know." It's OK not to have all the answers as you have a team to work with. The person should understand what goes on and respond appropriately.

Problem 5: Customer Organization Politics

This is a difficult problem to overcome with diversity of variables that can get in the way. One way is to communicate in terms of what's in it for the other person rather than your firm or someone else in your client's company. I haven't discovered any miracle solutions for overcoming politics.

SOFTWARE ANALYST

A software analyst is responsible for creating and designing software programs and applications, as well as modifying existing ones for optimization according to business requirements. Software analysts work with the technical team to draw system codes, analyze programming languages, and ensure the stability and efficiency of software navigation by running multiple quality checks to the system. They inspect the application's performance, configure servers, and improve software infrastructure according to quality findings. A software analyst records resolution reports and provides progress updates, ensuring that the project adheres to budget limitations and set timetables.

SOFTWARE ANALYST RESPONSIBILITIES

- Arrange project requirements in programming sequence by analyzing requirements; preparing a work flow chart and diagram using knowledge of computer capabilities, subject matter, programming language, and logic.
- Maintain, manage and modify all software systems, tools, and applications.
- Develop and analyze functional specifications.
- Be the interface between end-users and software consultants.
- Resolve complex issues relating to business requirements and objectives.
- Coordinate and support software professionals in installing and analyzing applications and tools.
- Develop, analyze and implement testing procedures, programming, and documentation.
- Train and develop other software analysts.
- Analyze, design, and develop modifications and changes to existing systems to enhance performance.
- Design efficient IT systems to meet business and technology needs.
- Coordinate with developers to build and implement technology solutions.
- Integrate multiple systems and reconcile the needs of different teams.
- Confirm program operation by conducting tests; modifying program sequence and/or codes.
- Research, evaluate and recommend solutions and appropriate technology to meet user's needs.
- Work with customers to maintain existing software as needed throughout their lifetime.
- On completion, you will install and train the customer to effectively use the program

Communication Skills

Clear, concise **communication is fundamental to the success of software engineering teams.** Although its relevance may not be immediately recognized as a top priority within a discipline largely based around a computational science, it is essential that teams collaborate effectively when striving to achieve superior results.
Effective communication can punctuate the success or failure of a software engineering team, a project, or an entire business operation. Poor communication (or miscommunication) often creates unnecessary expenditures. When solutions and ideas are communicated clearly across all channels, the overall budget will most certainly benefit from a cost-savings standpoint — and everyone can appreciate that!

7 tips to become a better communicator in software engineering

- **Be brief**

Good communication does not mean more communication. In many cases, quality over quantity is the key. We all have a variety of communication channels to manage these days and the clearer you can be with less words will often be appreciated by the recipient. You should aim to say fewer things, but without losing the impactful point of the message, and that's not easy.

- **Share information and improve communication**

One of the greatest things I've learned to do in improving my communication is by writing. Simple in theory, but a bit more difficult to do on a regular basis when the majority of us are multitasking and holding various conversations at once. It requires consistent practice. This can be done by writing to your team and asking them to provide feedback.

- **Listen carefully**

This one you've most certainly heard before. Listen carefully to what the other person is saying. Do not interrupt. Then, prior to adding your own comment to their thoughts, a great technique is to rephrase what you heard with your own words to confirm that you are fully understanding what the other person is attempting to communicate.

- **Ask questions**

Not only will you obtain more information, but again it will ensure that you are clearly absorbing the message being conveyed. Communication is not only what you say, but also how you say it and when you say it. Non-verbal communication can often relay messages that were not intended. With this in mind, be aware of your body language, facial expressions, and the tone of your voice, as well as the words you choose.

- **Understand customer requirements**

Being an effective communicator also relies heavily on taking the time to understand customer requirements and directions clearly. One thing that I often do to validate that I'm coding with the correct information, is to read the requirements before I start my work. However, I also read them again during the development phase and read them at the end of each task to confirm that I have completed the work correctly.

- **Attack the problem but not the person**

Sometimes a team member might have introduced an error in the code which causes a setback in the project. It is important that when you talk with your colleague, that you address the issue, without a personal criticism. If you criticize or attack the person, you risk damaging the ego and shutting down any open-mindedness. This eliminates any chance to move forward with a useful and collaborative solution.

- **Communication channel**

Choose the appropriate communication channel. Take a look at this image for example. There are different channels. Each channel has a ranking of effectiveness and cost. Depending on what you need to say, when you need to say it, and how important it is to avoid communication mistakes you will have a suggested means for the best channel to use.

REQUIREMENT ANALYSIS PRINCIPLES

To performed requirement analysis there must be some principles or guidelines to be followed. So there is a set of operational principles.

1. The information domain of a problem must be represented and understood.
2. The functions that the software is performing must be defined.
3. The behavior of the software (as a consequence of external events) must be represented.
4. The model is essentially a depiction of information. Function and behaviour which must be partitioned in a manner that uncovers detail in layered (or hierarchical) fashion.
5. The analysis process should move form essential information toward implementation details.

Principles of software development have illustrated a set of six guiding principle s which are given below:

1. **Understand the problem before you begin to create the analysis model:** there is a tendency to rush a solution, even before the problem is understood. This often leads to elegant software.
2. **Develop prototypes that enable a use to understand how human/ machine interaction will occur** an: To develop friendly and easy to use learn and understand, the interface prototyping are highly recommended.
3. **Record the origin of and the reason for every requirement:** this is the first step in establishing traceability back to the customer.
4. **Use multiple views of requirements:** Building data functional and behavioral models provide the software engineer with three different views.
5. **Ran requirements**: Priorities should be given to the requirements which one will be processed first. If an incremental process model is applied, those requirements to be delivered in the first requirement must be identified.
6. **Work to eliminate ambiguity**: Because mot requirements are described in a natural language, the opportunity for ambiguity abounds. The use of formal technical reviews is one way to UN cover and eliminates ambiguity.

Requirements Analysis Techniques:

There are different techniques used for business Requirements Analysis.
Below is a list of different business Requirements Analysis Techniques:

1. Business process modeling notation (BPMN)
2. UML (Unified Modeling Language)
3. Flowchart technique
4. Data flow diagram
5. Role Activity Diagrams (RAD)
6. Gantt Charts
7. IDEF (Integrated Definition for Function Modeling)
8. Gap Analysis

1- Business process modeling notation (BPMN)

This technique is similar to creating process flowcharts, although BPMN has its own symbols and elements. Business process modeling and notation is used to create graphs for the business process. These graphs simplify understanding the business process. BPMN is widely popular as a process improvement methodology.

2- UML (Unified Modeling Language)

UML consists of an integrated set of diagrams that are created to specify, visualize, construct and document the artifacts of a software system. UML is a useful technique while creating object-oriented software and working with the software development process. In UML, graphical notations are used to represent the design of a software project. UML also help in validating the architectural design of the software.

3- Flowchart technique

A flowchart depicts the sequential flow and control logic of a set of activities that are related. Flowcharts are in different formats such as linear, cross-functional, and top-down. The flowchart can represent system interactions, data flows, etc. Flow charts are easy to understand and can be used by both the technical and non-technical team members. Flowchart technique helps in showcasing the critical attributes of a process.

4- Data flow diagram

This technique is used to visually represent systems and processes that are complex and difficult to describe in text. Data flow diagrams represent the flow of information through a process or a system. It also includes the data inputs and outputs, data stores, and the various subprocess through which the data moves. DFD describes various entities and their relationships with the help of standardized notations and symbols. By visualizing all the elements of the system it is easier to identify any shortcomings. These shortcomings are then eliminated in a bid to create a robust solution.

5- Role Activity Diagrams (RAD)

Role-activity diagram (RAD) is a role-oriented process model that represents role-activity diagrams. Role activity diagrams are a high-level view that captures the dynamics and role structure of an organization. Roles are used to grouping together activities into units of responsibilities. Activities are the basic parts of a role. An activity may be either carried out in isolation or it may require coordination with other activities within the role.

6- Gantt Charts

Gantt charts used in project planning as they provide a visual representation of tasks that are scheduled along with the timelines. The Gantt charts help to know what is scheduled to be completed by which date. The start and end dates of all the tasks in the project can be seen in a single view.

7- IDEF (Integrated Definition for Function Modeling)

Integrated definition for function modeling (IDEFM) technique represents the functions of a process and their relationships to child and parent systems with the help of a box. It provides a blueprint to gain an understanding of an organization's system.

8- Gap Analysis

Gap analysis is a technique which helps to analyze the gaps in performance of a software application to determine whether the business requirements are met or not. It also involves the steps that are to be taken to ensure that all the business requirements are met successfully. Gap denotes the difference between the present state and the target state. Gap analysis is also known as need analysis, need assessment or need-gap analysis.

Software Quality Attributes

Software Quality Attributes help software architects to evaluate the performance of a software application. These quality attributes decide whether the software is of good quality or not. These quality attributes are also sometimes called “ilities” after the suffix most of the words related to system capability share such as availability, reliability, scalability, testability, etc. We have to do prioritize these system quality attributes based on our project needs.

1) Usability
It is described as how the user is utilizing a system effectively and the ease of which users can learn to operate or control the system. The well-known principle of usability is KISS (Keep It Simple Stupid). Software applications should be user-friendly.

2) Reliability
It is the ability of a system to continue to keep operating over time

3) Availability
It is the ratio of the available system time to the total working time it is required or expected to function.

4) Portability
It is the ability of a software application to run on numerous platforms such as data portability, hosting, viewing, etc.,

5) Testability
It shows how well the system or component facilitates to perform tests to determine whether the predefined test criteria have been met.

6) Scalability
It is the ability of a system to handle the demand for stress caused by increased usage without decreasing performance.

7) Flexibility
It is the ability of a system to adapt to future changes

8) Reusability
It is the use of existing software I more than one software with small or no change. It is a cost-efficient and time-saving quality attribute.

9) Maintainability
It is the ability of a software application to maintain easily and support changes cost-effectively.

10) Supportability
It is the ability of a system that satisfies necessary requirements and needs to identifying and solving problems.

11) Interoperability
It is the ability of two or more systems to communicate or exchange data easily and to use the data that has been exchanged.

12) Performance
It is the ability of a system in the form of responsiveness to various actions within a certain period of time

13) Security
It is the ability of a system to resist or block malicious or unauthorized attempts that destroy the system and at the same time provide access to legitimate users.

Basic Principles of Good Software Engineering

Some basic **principles** of good software engineering are –

- One of the basic software Engineering principles is Better Requirement analysis which gives a clear vision of the project. At last, a good understanding of user requirements provides value to its users by delivering a good software product that meets users' requirements.
- All designs and implementations should be as simple as possible mean the KISS (Keep it Simple, Stupid) principle should be followed. It makes code so simple as a result debugging and further maintenance become simple.
- Maintaining the vision of the project is the most important thing throughout complete development process for the success of a software project. A clear vision of the project leads to the development of the project in the right way.
- Software projects include a number of functionalities, all functionalities should be developed in a modular approach so that development will be faster and easier. This modularity makes functions or system components independent.
- Another specialization of the principle of separation of concerns is Abstraction for suppressing complex things and delivering simplicity to the customer/user means it gives what the actual user needs and hides unnecessary things.
- Think then Act is a must-required principle for software engineering means before starting developing functionality first it requires to think about application architecture, as good planning on the flow of project development produces better results.
- Sometimes developer adds up all functionalities together but later find no use of that. So following the Never add extra principle is important as it implements what actually needed and later implements what are required which saves effort and time.

- When other developers work with another’s code they should not be surprised and should not waste their time in getting code. So providing better Documentation at required steps is a good way of developing software projects.
- Law of Demeter should be followed as it makes classes independent on their functionalities and reduces connections and inter dependability between classes which is called coupling.
- The developers should develop the project in such a way that it should satisfy the principle of Generality means it should not be limited or restricted to some of cases/functions rather it should be free from unnatural restrictions and should be able to provide service to customers what actually they need or general needs in an extensive manner.
- Principle of Consistency is important in coding style and designing GUI (Graphical User Interface) as consistent coding style gives an easier reading of code and consistency in GUI makes user learning easier in dealing with interface and in using the software.
- Never waste time if anything is required and that already exists at that time take the help of Open source and fix it in your own way as per requirement.
- Performing continuous validation helps in checking software system meets requirement specifications and fulfills its intended purpose which helps in better software quality control.
- To exit in current technology market trends Using modern programming practices is important to meet users’ requirements in the latest and advanced way.
- Scalability in Software Engineering should be maintained to grow and manage increased demand for software applications.