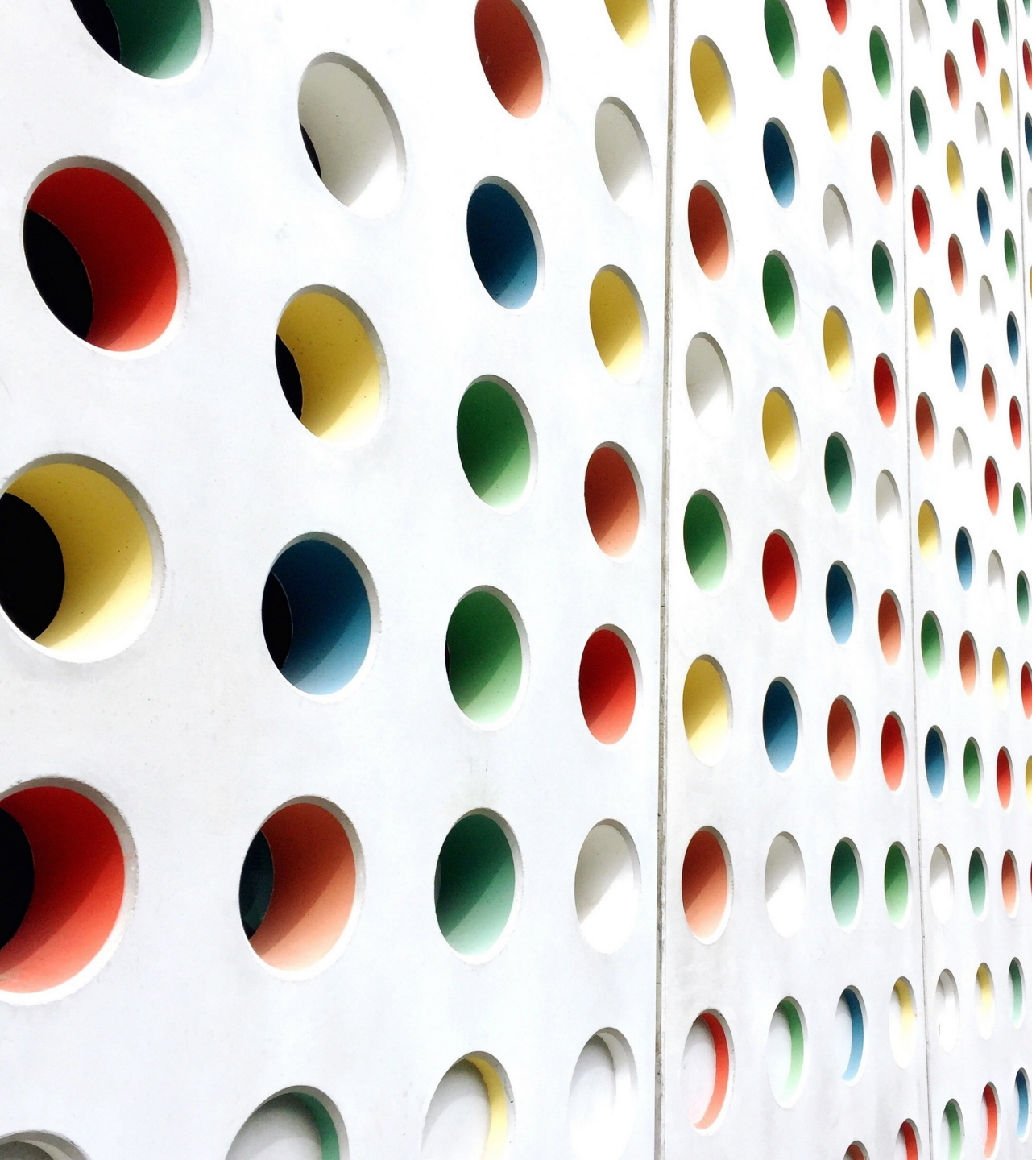


Software Engineering Practices

Software Engineering Practices - Definition, Importance

➢ **Definition :**

- It is a systematic process of developing software . Practice is a collection of concepts, principles, methods , and tools that are considered while planning and developing the software .
- Practices are used by software practitioners on day to day basis for developing software product.
- The software engineer implements this collection on daily basis for throughout development process of the software .
- Software engineers and managers apply the practice of software engineering.

➢ Importance :

- Software Engineering Practices play very important role in development of software product
- It needs scientific approach to be used for development . It is not easy .
- It is concepts, methods , principles, guidelines which helps in developing software product .
- The most important thing is customer satisfaction has to be achieved by delivering a software product that fulfills all customer requirements by ensuring the quality and standards .
- To deliver best products and thus sustain in the market for long time .
- Practices help at every stage of development including requirement analysis , modelling and designing, coding and implementation .It also specifies guidelines related to maintenance.

Core principles of Software Engineering

- Software Engineering core principles are important for successful software engineering process.
- Core principles support perfect software engineering approach.
- David hooker has proposed seven core principles .

1 . “ The reason at all exists “

The software can be called as complete software if it satisfies all the requirements of customer . So customer satisfaction is important that decides whether the work which is going to be carried out is going to add value to existing system or not .

2. “ Keep IT Simple Stupid / Keep it simple but perfect “.

Software designing cannot be done randomly .While designing, many factors are supposed to be taken into consideration .The design should be as simple as possible.

If there are too many interfaces in the system, it’s difficult to construct and implement such type of system .

System can be maintained easily . The complicated designing of system becomes very difficult to maintain as well as understand .

3. “ Maintain the vision “

Development vision is imperative for its success . There should be integrity in thoughts of people who work on it towards the vision of project .

If the architectural vision of software system is not perfect or if its weak then eventually it will break a good designed system . A good architect can hold the vision and confirms very successful software project.

4.” What you produce , others will consume”

It states about the transparency of the project designing and planning.Different teams work with same project but participating in different activities.

The project should be transparent so that the persons who work on it understand what has done. After deployment or use of software for customer, the software enhancement may be required as per customers requirement.

5. “ Be open to the future “

The system with long lifetime is always better and more preferable. As hardware technology changes, the software has to change for compatibility.

There should be a question “ what if”. The question tries to find out all the possibilities which gives generalized solutions and not specific.

6. “ Plan ahead for reuse “

The ‘ Reusable’ software can be used in development or other applications even. If reusable modules are available , then system can be developed very fast .

Due to reusability , the time and efforts are reduced .

7. “ Think “

When we think , we get more ideas . This improves quality of the system .Thinking before executing produces good results. When you actually think you gain knowledge , it becomes research .

When there is a clear thought then system is perfect.

Communication Practices

- **Concept , Need of Communication**

- The initial stage is to collect or gather requirements from customer .
- The requirement told by customer may not be exactly compatible with software or computer based solution . Developer responses to the customer accordingly.
- It is necessary that technical peers , customers , or users should communicate effectively
- Requirement analysis needs a huge communication . If there is misinterpretation in requirement analysis, the wrong design will be implemented . Hence customers requirement can’t be validated with expected results .

- **Principles**

1. “ Listen carefully “

It is necessary to become good listener . Listening ensures proper data collection from speaker . Concentration on speaker’s word is required to avoid chances of misinterpretation .

2. “ Be prepared before you communicate “

It is necessary to prepare the agenda for the meeting . Agenda contains points to be discussed in the meeting .

3. “Be prepared to facilitate the activity “

The successful communication meeting held should have a leader or facilitator who keeps conversation moving in productive direction .

4. “ Face to face communication”

Face to face communication is better but even if some documents related to that project the drawing and documents speaks better.

5. “ Take notes and document decisions “

Write down all the important points and issues raised in the meeting One participant of the meeting may act as ‘recorder ‘. These points can be referred to next meeting to evaluate progress achieved .

6.” Strive for collaboration “

For better team work collaboration in team members is essential. Collaboration is achieved when collective knowledge of members of the team is united to illustrate product of system functions or system failure.

Each and every collaboration is useful to build faith among team members and it creates generic goal for team.

7. “ Stay focused , modularize your direction “

It is necessary to keep limited number of people while discussing in the meeting .The facilitator should direct meeting in required direction so that important topics will be focused and discussion will modularize.

8. “ If something is unclear , draw a picture “

When verbal communication becomes difficult, a sketch or drawing can often help in giving clarity when words fail to do the job.

9. “ Keep the discussion to move on”

There are certain conditions when this move on approach should be used

i. Once there is an agreement to do something.

ii. If you can’t agree to something.

iii. If a feature or function is not clear .

In the meeting , there will be many people with different issues to discuss. Other members might not agree the opinion, But let it be. “ The show must go on”.

10. “Negotiation is successful when both parties agree “

Customer and developer must negotiate the function priorities and delivery dates . If team collaborates well, then they can compromise easily .

If there is negotiation then parties have same goal and they work towards it. This is beneficial for both customer as well as organization .

Planning Practices

- **Concept , Need of Planning**

- A big task needs planning for its successful execution .Without Planning it is difficult or almost impossible to undertake a big project.
- Planning becomes guidelines to team .
- Software planning includes complete estimation and scheduling and risk analysis .
- The planning activity is a set of management and technical practices that facilitates software team to create a road map as it moves towards its strategic aim and planned objectives.
- There are three types of people- Minimalists, Traditionalists, Agilists

- Planning activity includes :

i. Identification of resources which are required .

ii. Exact calculation of how many resources are required .

iii. Cost estimation

iv. Project schedule with alternative plans

v. Risk identification and management solutions

vi. Identification of technology related issues.

vii. Delivery date

- **Principles**

1. “Understand the scope of the project “

Having perfect planning , will lead to faster progress . Project should be very well planned and planning should be done by understanding future scope properly .

Scope is to understand the range in which project is going to be developed . It shows the boundaries of the project in which team have to work .

2. “Involve the customer in planning activities “

Software planning includes complete estimation and scheduling .Planning must be started with customers participation so that customer’s convenience can be considered regarding delivery date , increments, etc . Customer can clearly state their priorities and can understand the hurdles while developing the project .

3. “ Recognize that planning is iterative”

Incase of incremental model of software process, customer may suggest modifications after every delivery so accordingly planning must be changed . Thus , planning changes time to time.

Planning should accommodate continuous changes .

4. “Estimate based on what you know”

Estimation should be reliable and convenient for the team .If information is vague or unreliable , estimates will be equally unreliable .

5.” Consider risk as you define the plan “

Risk is an event that may or may not occur. It is an unwanted event, but if this event happens , it results to unwanted effects or outcome . Proper risk management solves this problem .

6.” Be realistic”

The project should be realistic . As human beings are involved in project , there are chances of delays , mistakes , wrong decisions. The schedule estimation of the project plan must be realistic .

7. “ Adjust granularity as you define the plan “

Granularity means ‘ level of detail’ .Some activities are repeated and some do not require for longer time . Thus , activities which do not occur for longer time do not require granularity .

A good granularity plan provides important work or activity as compare to plan which are planned in short time. The activities which do not occur for many months do not need for granularity .

8.” Define how you are going to achieve quality “

The most effective quality assurance mechanisms is to conduct Formal Technical Reviews (FTR) . FTR is meeting conducted by technical staff . FTR is applied to find out defects in any stage , because removing defects will improve software quality .

9. “ Describe how you aim to accommodate change”

If change is requested in early stages of software development , it can be easily carried out . But if change is requested in later stages , cost of change rapidly changes .

If change is requested in maintenance , modifications are required in design , coding , testing .Hence cost of modification rapidly increases .

Thus changes can’t be easily absorbed. But good software needs to be planned and accommodate changes demanded by the customer in any stage of the software development .

10.” Always keep track of the plan and make changes as required “

Access the progress of project development on daily basis . If it is found plan is lagging, take actions to recover it . All the team members should participate in planning activity. Time schedules are important in the software project . So it should be tracked to view the progress on day to day basis.

The watch should be kept on problem areas and situations in which schedule does not get complete .

Modeling Practices

- **Software Modeling**

- Models are created for better understanding of the design.
- Models must accomplish objectives at different level of abstraction . Two classes of models are created : analysis models and design models
- It must be capable of representing software at technical level.

A) Design models:

- It represents customer requirements by depicting the software in three domains.
- The information represents characteristics or attributes of the software that assist or help practitioners to construct efficiently .
- Three domains are:

a) Architecture

b) User interface

c) Component level detail

➢ **Principles :**

1. **“Design should be traceable from analysis model” .**

Using elements of analysis models , design model is constructed .

Ex: from ‘information domain ‘ of ‘ analysis model’ , the ‘ architectural design’ is developed .

2. **“Consider the architecture of the system to be built.”**

- Data centred architecture : Client software accessing central database .
- Data flow architecture : The input data flow after processing generates output data .
- Main/ subprogram architecture : Showing hierarchical relationship between main program and its subprograms . If architectural view is clear , its easy to implement using any language.

3. “ Design of data is as important as design of functions.”

- Design of data is important factor . The data designing shows relationship between different data objects .
- The design model includes the “Entity Relationship Model” to show the relationship among different data objects” .Thus data modelling is as important as “functional modelling” .

4. “ Internal as well as external Interfaces must be designed”.

- In given system , data may flow from one component to another component . Also, may flow from external environment to the system . These internal and external interfaces must be designed properly .

5. **“ User interface design must satisfy all need of end user.”**

The design must be as as per convenience of the user of the system . Hence user interface should be as simple as possible .

6. **“Component level design should be functionally independent .”**

- Functions should be designed such that all tasks of single software component i.e. module should be closely related . This is called ‘ cohesion’
- Cohesion measures the ‘functional strength’ of modules . Hence good design , all functions in single module should be highly cohesive.

7. **“ Components should be loosely coupled to one another and to the external environment”**

For good design , the interconnection among different components should be minimum. These interconnections among different modules are called as ‘coupling ‘. Hence for good design , the coupling should be low.

8. **“Designed modules should be easy to understand.”**

The design is implemented using appropriate programming language .Hence , design should be as simple as possible. Simple modules are easy to understand , easy to test , easy to debug ,and to easy to modify .

9. **“Accept that design behavior is iterative .”**

In case of incremental model of software process customer may suggest modifications after each delivery of the increment. Accordingly , design model is required to change .Good design should be capable of absorbing the changes as per requirement .

➢ **Qualities of Good design :**

- Correctness
- Efficiency
- Understandability
- Maintainibility

- **Design Constraints**

1. Technical constraints

    a) Programming language

    b)OS

    c)Framework

2. Business constraints

a) Schedule

b) Budget

c)Team composition

d)Requirements

B) Analysis Modeling

- Analysis model represents customer requirements
- To represent requirements using analysis model, three domains are used :

i. Information domain

ii. Functional domain

iii. Behavioral domain

- For analysis modeling , principles are :

1. “The information domain of problem must be clearly represented.”

Analysis model uses ‘ Data flow Diagram (DFD) ‘. Information domain includes: Input flow and output flow .

2. “ The functions of the software must be defined clearly “.

Functions are processes that transform input flow to output flow .The specifications must be defined clearly .

3.” Behavior of the system must be defined clearly “.

Analysis model uses ‘ State transition diagrams ‘ to represent behaviour of the system shows how the system makes transition from one state to another on occurrence of some external event.

4.” The clear hierarchy among information , functions and behaviour must be shown “.

The proper hierarchy of analysis model leads to easy design .It can be represented using layers or levels .

5.”Analysis should be clear enough to convert it to design modeling”.

The next of modeling is to convert analysis model to design model .Hence if analysis of requirements is clear and simple then design will be easy .

Flow oriented modeling

- **Data Dictionaries**

A data dictionary is a file or a set of files that includes a database's metadata. The data dictionary hold records about other objects in the database, such as data ownership, data relationships to other objects, and other data. The data dictionary is an essential component of any relational database. Ironically, because of its importance, it is invisible to most database users. Typically, only database administrators interact with the data dictionary.

The data dictionary, in general, includes information about the following:

- Name of the data item
- Aliases
- Description/purpose

- Related data items
- Range of values
- Data structure definition/Forms

➢ Decision tables :

- **Decision table** is a brief visual representation for specifying which actions to perform depending on given conditions. The information represented in decision tables can also be represented as decision trees or in a programming language using if-then-else and switch-case statements.

- A decision table is a good way to settle with different combination inputs with their corresponding outputs and also called cause-effect table. Reason to call cause-effect table is a related logical diagramming technique called cause-effect graphing that is basically used to obtain the decision table.

DFD

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It can be manual, automated, or a combination of both.

It shows how data enters and leaves the system, what changes the information, and where data is stored.

The objective of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communication tool between a system analyst and any person who plays a part in the order that acts as a starting point for redesigning a system. The DFD is also called as a data flow graph or bubble chart.

DFD consists of processes, flows, warehouses, and terminators.

Process

The process (function, transformation) is part of a system that transforms inputs to outputs. The symbol of a process is a circle, an oval, a rectangle or a rectangle with rounded corners (according to the type of notation). The process is named in one word, a short sentence, or a phrase that is clearly to express its essence.[2]

Data Flow

Data flow (flow, dataflow) shows the transfer of information (sometimes also material) from one part of the system to another. The symbol of the flow is the arrow. The flow should have a name that determines what information (or what material) is being moved. Exceptions are flows where it is clear what information is transferred through the entities that are linked to these flows. Material shifts are modeled in systems that are not merely informative. Flow should only transmit one type of information (material)

The arrow shows the flow direction (it can also be bi-directional if the information to/from the entity is logically dependent - e.g. question and answer). Flows link processes, warehouses and terminators.

Warehouse

The warehouse (datastore, data store, file, database) is used to store data for later use. The symbol of the store is two horizontal lines, the other way of view is shown in the DFD Notation. The name of the warehouse is a plural noun (e.g. orders) - it derives from the input and output streams of the warehouse. The warehouse does not have to be just a data file, for example, a folder with documents, a filing cabinet, and optical discs. Therefore, viewing the warehouse in DFD is independent of implementation. The flow from the warehouse usually represents the reading of the data stored in the warehouse, and the flow to the warehouse usually expresses data entry or updating (sometimes also deleting data). Warehouse is represented by two parallel lines between which the memory name is located (it can be modeled as a UML buffer node).

Terminator

The Terminator is an external entity that communicates with the system and stands outside of the system. It can be, for example, various organizations (eg a bank), groups of people (e.g. customers), authorities (e.g. a tax office) or a department (e.g. a human-resources department) of the same organization, which does not belong to the model system. The terminator may be another system with which the modeled system communicates.[2]

Scenario based modeling

- **Use Case Diagram**

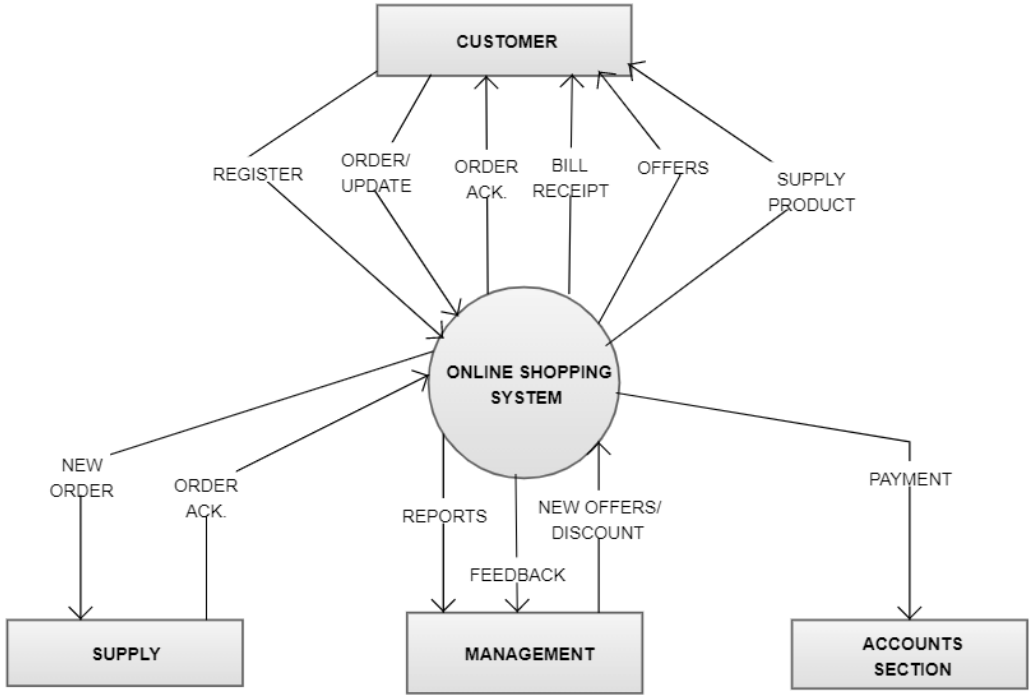
Use case diagrams are usually referred to as behavior diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors). Each use case should provide some observable and valuable result to the actors or other stakeholders of the system.

Purpose of Use case:

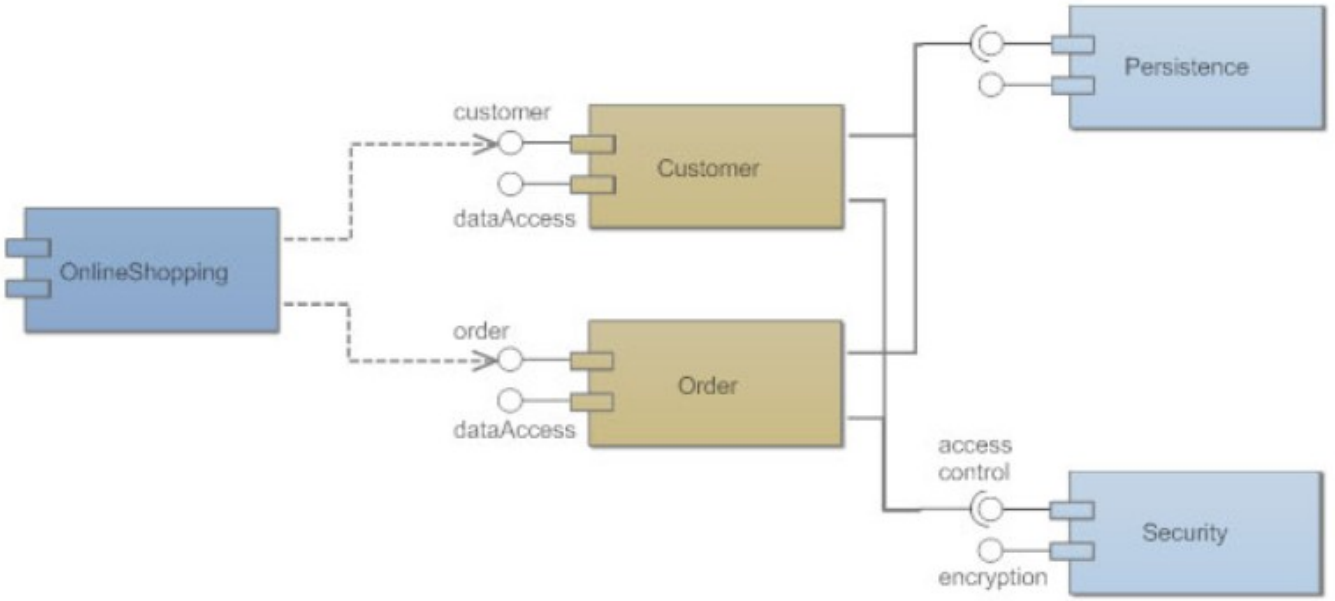
The purpose of the use case diagrams is simply to provide the high level view of the system and convey the requirements in laypeople's terms for the stakeholders. Additional diagrams and documentation can be used to provide a complete functional and technical view of the system.

Ex: Online Shopping

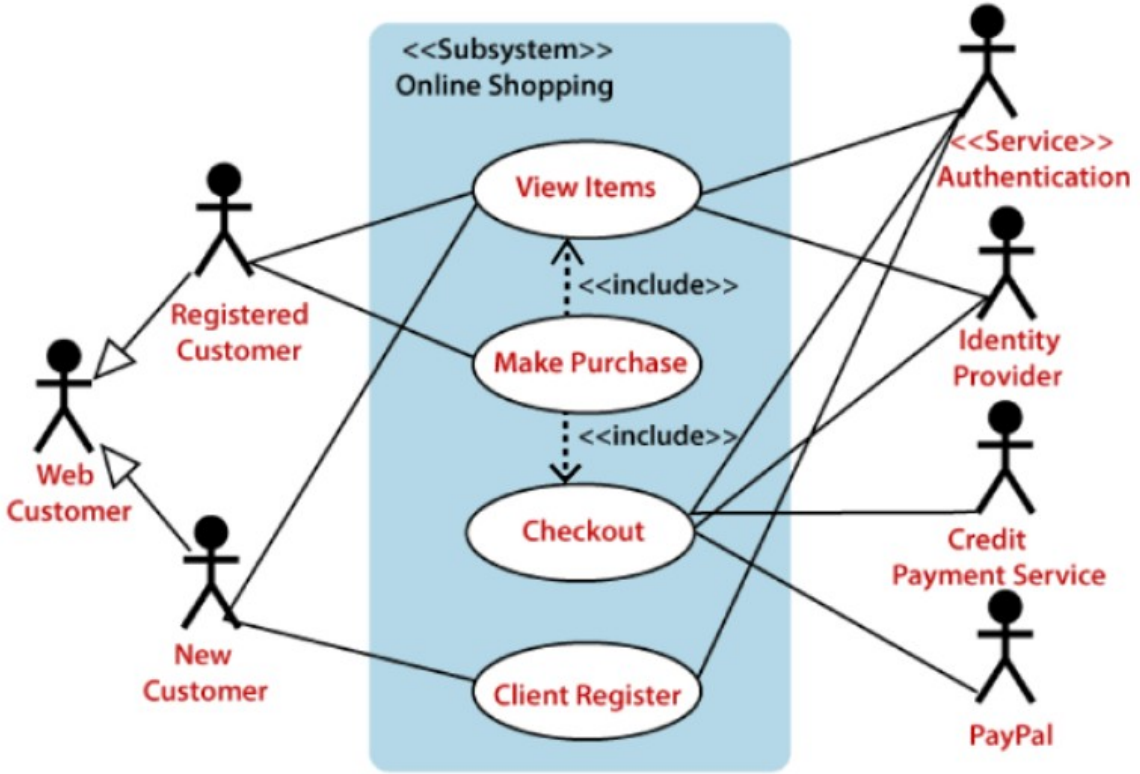
- **DFD**



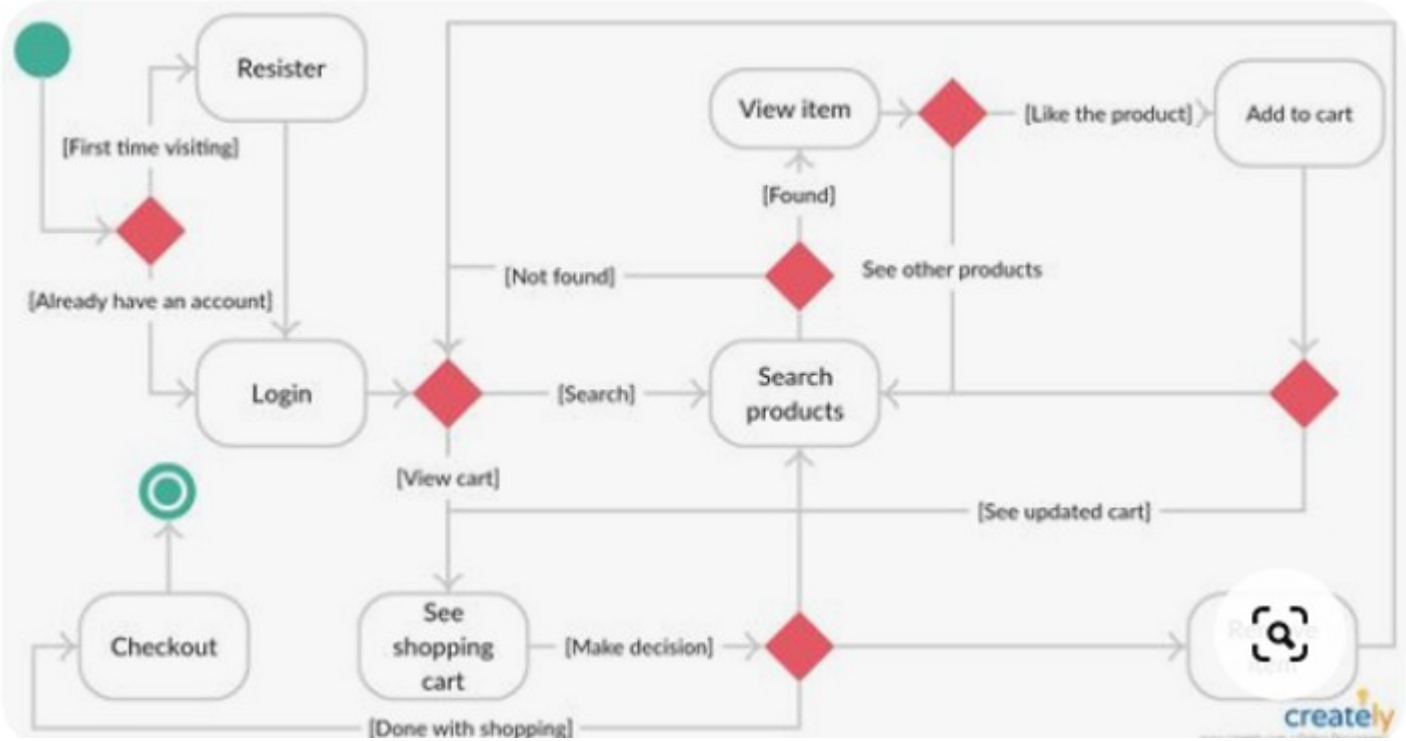
Component level diagram :



Use Case Diagram



State transition diagram



C) Behavioural Modelling

- Behavioral Modelling is represented by state transition diagram by depicting its states and events that cause the system to change .
- The states represent a mode of behavior of the system and state transition diagram represents how the system moves from state to state.
- Each arrow of the state transition diagram is name with ruled expression . The value at the top indicates the events that cause the transition to happen.
- The value at the bottom represents the action that occurs as a result of event.

Construction Practices

- The term software construction refers to the detailed creation of working software through a combination of coding, verification, unit testing, integration testing, and debugging.
- Software construction fundamentals includes:

  minimizing complexity

  anticipating change

  constructing for verification

  reuse

  standards in construction.

➢ Coding :

- Design details are implemented using appropriate programming language .Coding is implemented :
- Using suitable programming language to generate source code eg C, COBOL ,Visual Basics .
- Using automated tools like Microsoft Front Page where code is automatically generated.
- Using fourth generation programming language like VC++ to generate executable code.

- Principles :

Preparation Principles :

1. Understand the exact problem for which you are trying .
2. Understand the basic design
3. Select programming language
4. Select programming environment that will be suitable for writing the code
5. Create set of unit tests those will be applied after completion of unit code .

Actual coding principles:

1. Algorithm used must be follow the programming language
2. Use data structures those will fit in coding.
3. Software architecture and interfaces are to be implemented as per design specifications.
4. Keep minimum nested conditions and loops.
5. Code must be self documented

- **Validation :**

To be followed after completing first coding pass:

1. Conduct unit test. Test whether the module is producing exact output.
2. Refactor the code if necessary.

➢ **Testing :**

- Testing is done to:

i. Check whether flow of coding is correct .

ii. check out the errors .

iii. check whether the program is giving expected output as per input specifications .

- Sequential steps for testing process :

i. Unit testing:

ii. Integration Testing

iii. Validation testing

iv. Acceptance testing

- Software testing must ensure :

i. Quality of the software

ii. Customer satisfaction

iii. Removal of errors

iv. Verification and validation of the product

v. Compatibility of the software

- Principles :

1. Tests must be conducted to validate customers requirements
2. Tests should be well planned before testing work
3. The Pareto principle is applicable for software testing
4. Testing should begin “in the small” and progress towards testing “in the large”.
5. Accept that “testing of every combination of paths is not possible”.

Software Deployment

- Deployment activity consists of three activities that are delivery cycle, support cycle , feedback cycle.
- Deployment takes place many times as software moves towards completion . Deployment does not happen only once .
- Each and every delivery cycle facilitates end user and customer with help of operational software increment .It provides useful functions and features included during all deployment cycles to date. In software project , important milestone is delivery of software increment .
- Every feedback cycle offers significant guidance that result in editing or modifying to the functions and approaches for next increment .

- Support cycle includes removing bugs if any, maintenance , making changes as per user requirements , working based on feedback , etc.

➢ **Deployment Principles :**

1. “ Manage customer’s expectations”

It always happen that customer wants more than he stated earlier as his requirements . It may be the case that customer is disappointed , even after getting all his requirements satisfied .Hence , at time of software delivery, developer must have skills to manage the customers expectations .

2. “ Assemble and test complete delivery package”

The customer must get all supporting and essential help from developer’s side .

3. “Record keeping mechanism must be established for customer support “

‘Customer support’ is important factor in deployment phase . If proper support is not provided , customer will not be satisfied , Hence support should be well planned and with proper record-keeping mechanism.

4. “Provide essential instructions ,documentations and manual “

‘When project is successful , deliverable product is only working program ‘ is what developer thinks . Actual delivery includes all documentation , help files and guidance for handling the software by user .

5. “Don’t deliver any defective or buggy software to the customer “

In incremental types , software may deliver some defective to the customer by giving assurance that defects.

SRS (Software Requirements Specifications)

- After the analyst has gathered all the required information regarding the software to be developed, and has removed all incompleteness, inconsistencies, and anomalies from the specification, he starts to systematically organize the requirements in the form of an SRS.
- The SRS document usually contains all the user requirements in a structured though an informal form. Among all the documents produced during a software development life cycle, SRS document is probably the most important document and is the toughest to write.
- One reason for this difficulty is that the SRS document is expected to cater to the needs of a wide variety of audience. In the following subsection, we discuss the different categories of users of an SRS document and their needs from it.

➢ Characteristics of Good SRS document :

- Concise
- Implementation independent
- Modifiable
- Traceable
- Verifiable

➢ **Format of SRS:**

- Introduction

1. Purpose: This section should describe where the software would be deployed and and how the software would be used.
2. Project scope: This section should briefly describe the overall context within which the software is being developed. For example, the parts of a problem that are being automated and the parts that would need to be automated during future evolution of the software.
3. Environmental characteristics: This section should briefly outline the environment (hardware and other software) with which the software will interact

- Overall description of organisation of SRS document

1. Product perspective: This section needs to briefly state as to whether the software is intended to be a replacement for a certain existing systems, or it is a new software. If the software being developed would be used as a component of a larger system, a simple schematic diagram can be given to show the major components of the overall system, subsystem interconnections, and external interfaces can be helpful.
2. Product features: This section should summarize the major ways in which the software would be used. Details should be provided in Section 3 of the document. So, only a brief summary should be presented here.
3. User classes: Various user classes that are expected to use this software are identified and described here. The different classes of users are identified by the types of functionalities that

they are expected to invoke, or their levels of expertise in using computers.

4. Operating environment: This section should discuss in some detail the hardware platform on which the software would run, the operating system, and other application software with which the developed software would interact.

5. Design and implementation constraints: In this section, constraints on the design and implementation are discussed. These might include—corporate or regulatory policies; hardware limitations (timing requirements, memory requirements); interfaces to other applications; specific technologies, tools, and databases to be used; specific programming language to be used; specific communication protocols to be used; security considerations; design conventions or programming standards.

6. User documentation: This section should list out the types of user documentation, such as user manuals, on-line help, and trouble-shooting manuals that will be delivered to the customer .

- Functional requirements for organization of SRS document

This section can classify the functionalities either based on the specific functionalities invoked by different users, or the functionalities that are available in different modes, etc., depending what may be appropriate. (Detail explanation verbally)

- External interface requirements

1. User interfaces:

This section should describe a high-level description of various interfaces and various principles to be followed. The user interface description may include sample screen images, any GUI standards or style guides that are to be followed, screen layout constraints, standard push buttons (e.g., help) that will appear on every screen, keyboard shortcuts, error message display standards, etc. The

details of the user interface design should be documented in a separate user interface specification document.

2. Hardware interfaces: This section should describe the interface between the software and the hardware components of the system. This section may include the description of the supported device types, the nature of the data and control interactions between the software and the hardware, and the communication protocols to be used.

3. Software interfaces: This section should describe the connections between this software and other specific software components, including databases, operating systems, tools, libraries, and integrated commercial components, etc. Identify the data items that would be input to the software and the data that would be output should be identified and the purpose of each should be described

4. Communications interfaces:

This section should describe the requirements associated with any type of communications required by the software, such as e-mail, web access, network server communications protocols, etc. This section should define any pertinent message formatting to be used. It should also identify any communication standards that will be used, such as TCP sockets, FTP, HTTP, or SHTTP. Specify any communication security or encryption issues that may be relevant, and also the data transfer rates, and synchronization mechanisms.

- Non-functional requirements

1. Performance requirements: Aspects such as number of transaction to be completed per second should be specified here. Some performance requirements may be specific to individual functional requirements or features. These should also be specified here.
2. Safety requirements: Those requirements that are concerned with possible loss or damage that could result from the use of the software are specified here. For example, recovery after power failure, handling software and hardware failures, etc. may be documented here.
3. Security requirements: This section should specify any requirements regarding security or privacy requirements on data used or created by the software. Any user identity authentication requirements should be described here. It should also refer to any external policies or regulations concerning the security issues.

➢ Need of SRS :

- Forms an agreement between the customers and the developers:

A good SRS document sets the stage for the customers to form their expectation about the software and the developers about what is expected from the software.

- Reduces future reworks: The process of preparation of the SRS document forces the stakeholders to rigorously think about all of the requirements before design and development get underway. This reduces later redesign, recoding, and retesting. Careful review of the SRS document can reveal omissions, misunderstandings, and inconsistencies early in the development cycle.

- Provides a basis for estimating costs and schedules: Project managers usually estimate the size of the software from an analysis of the SRS document. Based on this estimate they make other

estimations such as the effort required to develop the software and the total cost of development. The SRS document also serves as a basis for price negotiations with the customer. The pr oject manager also uses the SRS document for work scheduling.

- Provides a baseline for validation and verification: The SRS provides a baseline against which compliance of the developed software can be checked. It is also used by the test engineers to create the test plan.

- Facilitates future extensions: The SRS document usually serves as a basis for planning future enhancements. Before we discuss about how to write an SRS document, we first discuss the characteristics of a good SRS document and the pitfalls that one must consciously avoid while writing an SRS document.

➢ Requirement Engineering

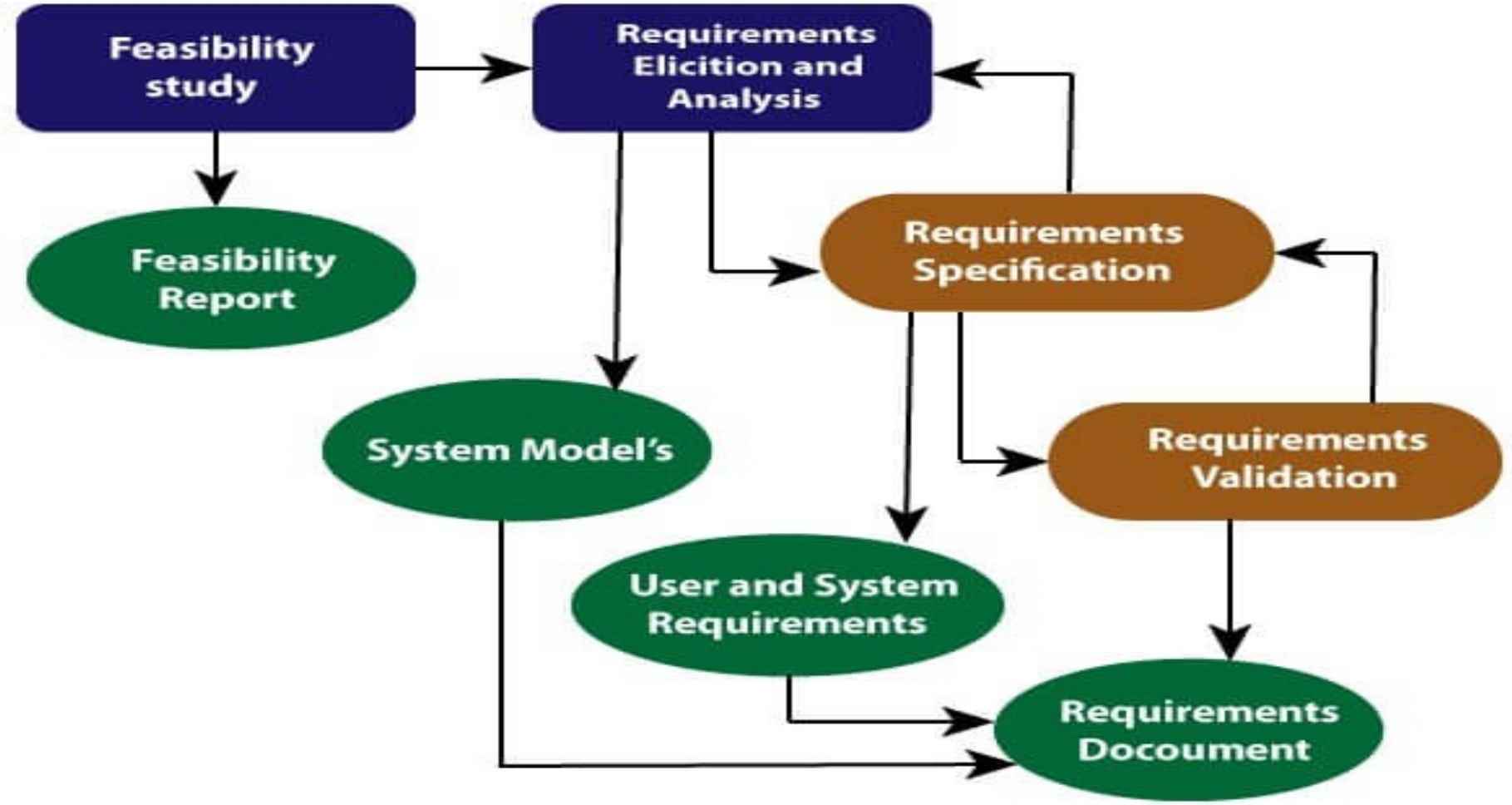
Requirement Engineering is the process of defining, documenting and maintaining the requirements. It is a process of gathering and defining service provided by the system.

It is a common role in systems engineering and software engineering. In the waterfall model, requirements engineering is presented as the first phase of the development process.

Requirement engineering provides the appropriate mechanism to understand what the customer desires, analyzing the need, and assessing feasibility, negotiating a reasonable solution, specifying the solution clearly, validating the specifications and managing the requirements as they are transformed into a working system.

Thus, requirement engineering is the disciplined application of proven principles, methods, tools, and notation to describe a proposed system's intended behavior and its associated constraints.

process



Requirement Engineering Process

Requirement Gathering

Requirements gathering is one of the most essential parts of any project and adds value to a project on multiple levels. When it comes to smaller budgets, tighter timelines and limited scopes, exact documentation of all the project requirements become crucial.

Requirements gathering is easier said than done, it is generally an area that is given far less attention than it needs. Many projects start with basic lists of requirements only to find out down the line that many of the customers' needs may not have been fully understood and implemented.

Requirement gathering process

Assigning Roles

As the project manager, you will need to decide who is going to do what. Firstly, the team must know your role. Secondly, ensure that every person understands their role and knows to come to you with all the project updates.

You will need to interview the stakeholders that you identified, asking them the following questions.

What is your vision or goal for the project?

What do you want from the project that hasn’t been done previously?

What changes to the product would convince you to recommend it to others?

What tools do you need for the project to be successful?

Gathering And Documentation

It is highly recommended to write everything down. Record every answer and try to create an accessible system that can be accessed by others when they require information from the gathering phase.

Make Lists Of All Expectations And Requirements

Once you have successfully documented expectations and objectives, you can create a cohesive requirements management plan that is measurable, quantifiable, and actionable. Questions you will need to answer include:

The length of the project timeline

Who will be involved in the project?

Monitor the Process And Feedback

Once you receive stakeholder approval and feedback, you will be able to start implementing adjustments into the project timeline. It is important to make sure you have systems and methods in place to monitor and track requirements across all teams, to ensure that the risk stays low.

Feedback is also crucial. You can use this data to indicate progress to stakeholders, department managers, and other team members. By doing so, you will ensure the project is on track regarding time, scope, and budget.

Requirement Analysis

In systems engineering and software engineering, requirements analysis focuses on the tasks that determine the needs or conditions to meet the new or altered product or project.

The requirements are analyzed to identify inconsistencies, defects, omission, etc. We describe requirements in terms of relationships and also resolve conflicts if any.

Requirement analysis starts with:

Requirement gathering which is also called as elicitation.

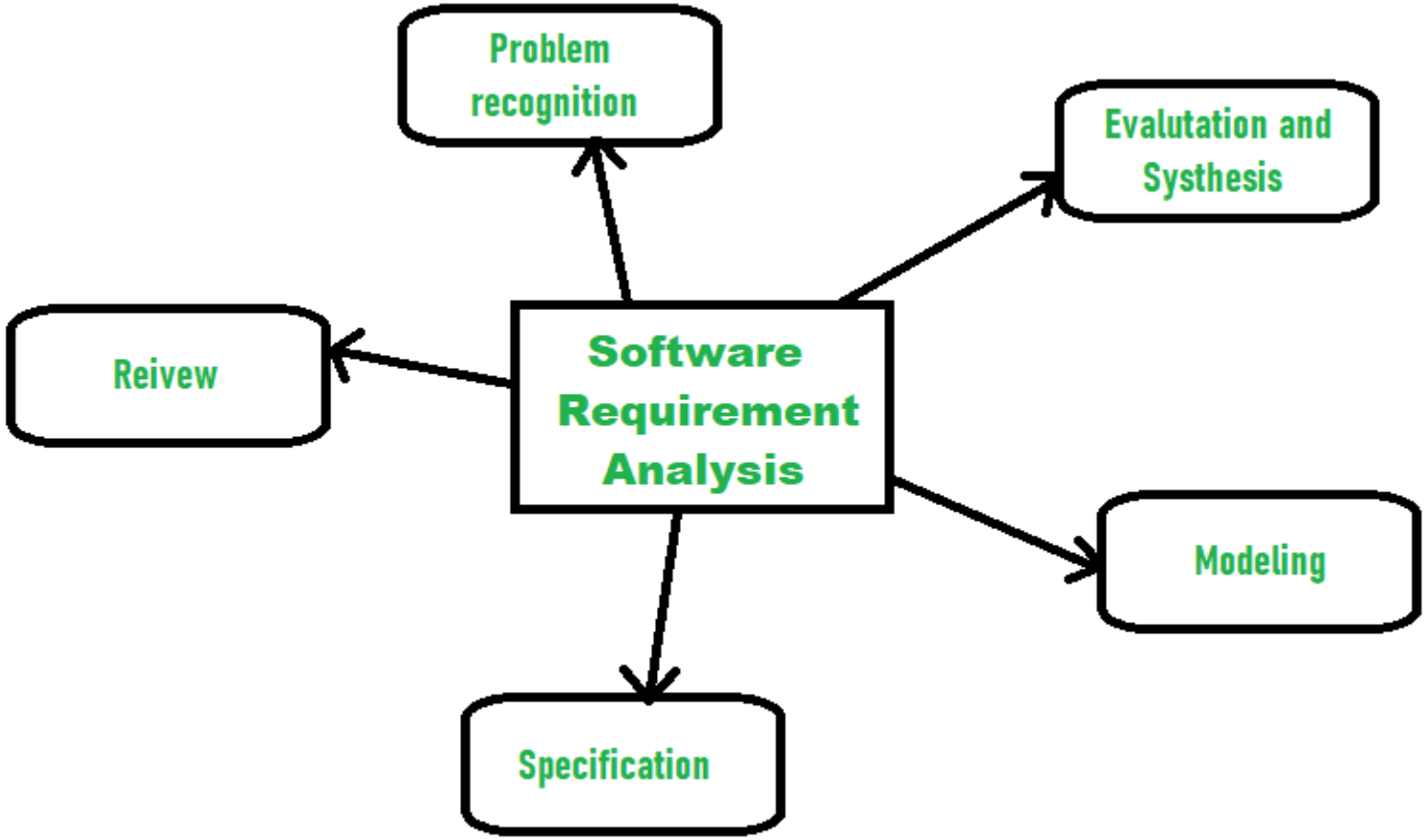
This is followed by **analyzing** the collected requirements to understand the correctness and feasibility of converting these requirements into a possible product.

And finally, **documenting** the requirements collected.

Requirements analysis is critical to the success or failure of a systems or software project

The requirements should be documented, actionable, measurable, testable, traceable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design.

Analysis


Problem recognition
Evalutation and Systhesis
Reivew
Software Requirement Analysis
Modeling
Specification

Types of requirement

In software engineering and systems engineering, a functional requirement defines a function of a system or its component, where a function is described as a specification of behavior between inputs and outputs.

Functional requirements may involve calculations, technical details, data manipulation and processing, and other specific functionality that define what a system is supposed to accomplish.

functional requirements specify particular results of a system.

These are the requirements that the end user specifically demands as basic facilities that the system should offer.

A product requirements document (PRD) is an artifact used in the product development process to communicate what capabilities must be included in a product release to the development and testing teams.

This document is typically used more in waterfall environments where product definition, design, and delivery happen sequentially, but may be used in an agile setting as well.

The PRD may follow on the heels of a marketing requirements document (MRD)—created by product marketing, marketing, or product management as well—that describes customer demand, market opportunity, and a business case for the overall product or a particular product release.

Some examples of **organizational requirements** are:

the organization's vision, goals, objectives and priorities. business and performance plans. ... legal requirements, for example, occupational health and safety and anti-discrimination legislation. standards (such as for ethical behavior) and protocols.

Organizational requirements. Requirements which are a consequence of organizational policies and procedures e.g. process standards used, implementation requirements, etc.

External requirements. Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements.

External requirements are those Requirement elements that have been connected to the current element using a Realization connector. By creating the connector from the element to the Requirement, you create an expectation that the element must implement the requirement as part of the system solution.

External requirements refer to the status of other elements in the model that are not sub-components of the given element. For example, assume that we define a computer as a system, and place inside the system several sub-components (e.g., motherboard, CPU, power supply, DVD drive, monitor).

The term elicitation is used in books and research to raise the fact that good requirements cannot just be collected from the customer, as would be indicated by the name requirements gathering.

Requirements elicitation is non-trivial because you can never be sure you get all requirements from the user and customer by just asking them what the system should do or not do (for Safety and Reliability).

Requirements elicitation practices include interviews, questionnaires, user observation, workshops, brainstorming, use cases, role playing and prototyping.

developing use cases

A use case is a description of how a person who actually uses that process or system will accomplish a goal. It's typically associated with software systems, but can be used in reference to any process.

A use case is a written description of how users will perform tasks on your website. It outlines, from a user's point of view, a system's behavior as it responds to a request. Each use case is represented as a sequence of simple steps, beginning with a user's goal and ending when that goal is fulfilled.

The three main parts of a use case scenario are the use case identifiers and initiators; the steps performed; and the conditions, assumptions, and questions.

Step 1: Identify who is going to be using the system directly. These are the Actors.

Step 2: Pick one of those Actors.

Step 3: Define what that Actor wants to do with the system. Each of these things that the actor wants to do with the system become a Use Case.

Step 4: For each of those Use Cases decide on the most usual course when that Actor is using the system. What normally happens.

Step 5: Describe that basic course in the description for the use case.

Step 6: Once you're happy with the basic course now consider the alternatives and add those as extending use cases.

building the requirements model

Requirements modeling in software engineering is essentially the planning stage of a software application or system.

Requirements modeling comprises several stages, or 'patterns': scenario-based modeling, data modeling, flow-oriented modeling, class-based modeling and behavioral modeling.

Requirements modeling is an approach used in projects where the requirements keep on evolving throughout the project.

Scenario-based elements :

Using a scenario-based approach, system is described from user's point of view. **For example**, basic use cases and their corresponding use-case diagrams evolve into more elaborate template-based use cases.

Class-based elements :

A collection of things that have similar attributes and common behaviors i.e., objects are categorized into classes.

In addition to class diagrams, other analysis modeling elements depict manner in which classes collaborate with one another and relationships and interactions between classes.

Behavioral elements :

Effect of behavior of computer-based system can be seen on design that is chosen and implementation approach that is applied. Modeling elements that depict behavior must be provided by requirements model.

Method for representing behavior of a system by depicting its states and events that cause system to change state is state diagram.

Flow-oriented elements :

As it flows through a computer-based system information is transformed. System accepts input, applies functions to transform it, and produces output in a various forms. Input may be a control signal transmitted by a transducer, a series of numbers typed by human operator, a packet of information transmitted on a network link retrieved from secondary storage.

requirements negotiation

Requirements engineering is a fundamental part of the software engineering process. When the stakeholders of the software project disagree on the requirements, requirements negotiation methods can be used to reach that agreement. This avoids rework and extra costs.

Requirements negotiation is an iterative. process through which stakeholders. make tradeoffs between. ▪ requested system functions. ▪ the capabilities of existing or envisioned.

There are five collaborative stages of the negotiation process: Prepare, Information Exchange, Bargain, Conclude, Execute.

There is no shortcut to negotiation preparation

Building trust in negotiations is key.

Communication skills are critical during bargaining.

Validation is the process of checking whether the software product is up to the mark or in other words product has high level requirements.

It is the process of checking the validation of product i.e. it checks what we are developing is the right product. Validation is the Dynamic Testing.

Validation activities are created and managed in the Business console, and are used to track and manage a test plan for the release and the results. ... When all validation activities are completed, the release can be approved and completed, at which point deployment can occur.