DIGITAL NOTES
ON
SOFTWARE REQUIREMENTS AND ESTIMATION

(R20A0561)

B.TECH III YEAR–II SEM

(2023-2024)

PREPARED BY

G.LAVARAJU

K. SHANTHI

DEPARTMENT OF INFORMATION TECHNOLOGY

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous Institution–UGC,Govt.of India)

(AffiliatedtoJNTUH, Hyderabad, Approved byAICTE-AccreditedbyNBA&NAAC–A'Grade-ISO9001:2015Certified)

Maisammaguda,Dhulapally(PostVia.Hakimpet),Secunderabad–500100,TelanganaState,India

MALLA REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY

B.TECH - III- YEAR II-SEM-IT **L/T/P/C**

3/-/-/3

(R20A0561) SOFTWARE REQUIREMENTS AND ESTIMATION

Course Objectives:

1. Learn the importance and scope of software requirements.
2. Understand the concepts of Requirements elicitation & Requirements modeling
3. Gain knowledge of principles and practices of Requirements Management.
4. Understand the process and methods of software estimation and size estimation.
5. Learn about Effort, Schedule and Cost Estimation techniques.

UNIT-I

Software Requirements: What and Why

Essential Software requirement, good practices for requirements engineering, Improving requirements processes, Software requirements and risk management

UNIT- II

Software Requirements Engineering

Requirements elicitation, requirements analysis documentation, review, elicitation techniques, analysis models, Software quality attributes, risk reduction through prototyping, setting requirements priorities, verifying requirements quality, **Software Requirements Modeling**- Use Case Modeling, Analysis Models, Dataflow diagram, state transition diagram, class diagrams, Object analysis, Problem Frame

UNIT-III

Software Requirements Management

Requirements management Principles and practices, Requirements attributes, Change Management Process, Requirements Traceability Matrix, Links in requirements chain Requirements ManagementTools: Benefits of using a requirements management tool, commercial requirements management tool, Rational Requisite pro, Caliber–RM, implementing requirements management automation,

UNIT- IV

Software Estimation

Components of Software Estimations, Estimation methods, Problems associated with estimation, Key project factors that influence estimation. Size Estimation-Two views of sizing, Function Point Analysis, MarkII FPA, Full Function Points, LOC Estimation, Conversion between size measures,

UNIT -V

Effort, Schedule and Cost Estimation

What is Productivity? Estimation Factors, Approaches to Effort and Schedule Estimation, COCOMO II, Putnam Estimation Model, Algorithmic models, Cost Estimation

Software Estimation Tools:

Desirable features in software estimation tools, IFPUG,USC'sCOCOMO II,
SLIM (Software Life CycleManagement) Tools

TEXT BOOKS:

1. Software Requirements and Estimation by Rajesh Naik and Swapna Kishore, Tata Mc Graw Hill

REFERENCES:

1. Software Requirements by Karl E. Weigers,Microsoft Press.
2. Managing Software Requirements, Dean Leffingwell & Don Widrig, Pearson Education,2003.
3. Estimating Software Costs, Second edition, Capers Jones, Tata McGraw-Hill, 2007.
4. Practical Software Estimation, M.A. Parthasarathy, Pearson Education, 2007.

Course Outcomes:
At the end of the course the students will be able to:

1. Develop an SRS for a business system.
2. Analyze and compare the various requirements elicitation techniques.
3. Construct the analysis models like DFD, UCD, ERD etc.
4. Apply Size Estimation methods to a specific data set.
5. Estimate the software in terms of size, cost, effort and schedule.

INDEX

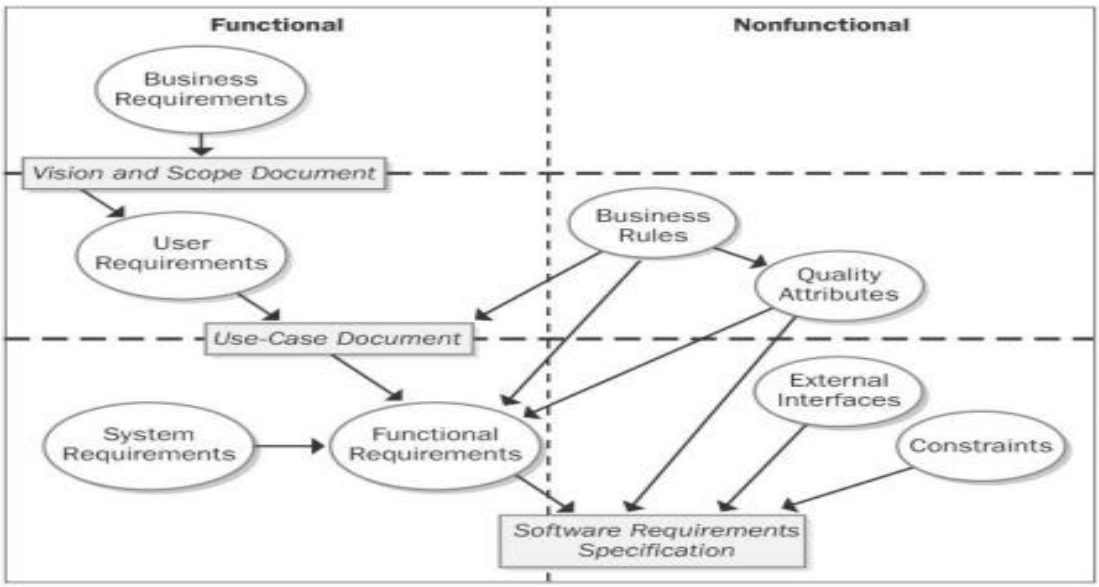
UNIT-I

Essential Software requirement:

Many software problems arise from short comings in the ways that people gather, document, agree on, and modify the product's requirements. The problem areas might include informal information gathering, implied functionality, erroneous or uncommunicated assumptions, inadequately defined requirements, and a casual change process.

Requirements *are a specification of what should be implemented. They are descriptions of how the system should behave, or of a system property or attribute. They may be a constraint on the development process of the system (Sommerville 1997).*

Requirements must be documented, they are the foundation for both the software development and the project management activities, all stakeholders must be committed to following an effective requirements process.



Software requirements include three distinct levels—business requirements,user requirements, and functional requirements. In addition, every system has an assortment of nonfunctional requirements.

- **Business requirements** describe why the organization is implementing the system—the objectives the organization hopes to achieve.
- **User requirements** describe user goals or tasks that the users must be able to perform with the product.
- **Functional requirements** specify the software functionality that the developers must build into the product to enable users to accomplish their tasks, thereby satisfying the business requirements. Functional requirements are documented in a
- **software requirements specification** (SRS). **System requirements** describes the toplevel requirements for a product that contains multiple subsystems—that is, a system.

A **feature** is a set of logically related functional requirements that provides a capability to the user and enables the satisfaction of a business objective.

Requirements specifications do **NOT** include design or implementation details (other than known constraints), project planning information, or testing information. These are project requirements but not product requirements;

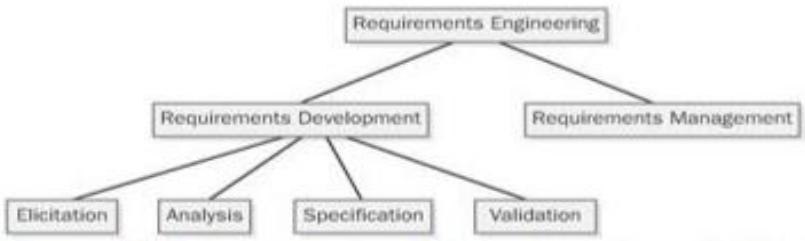


Figure 1-2: Subcomponents of the requirements engineering domain.

These sub disciplines encompass all the activities involved with gathering, evaluating,

and documenting the requirements for a software or software-containing product.

Requirements management entails "establishing and maintaining an agreement with the customer on the requirements for the software project".

It costs far more to correct a defect that's found late in the project than to fix it shortly after its creation. Preventing requirements errors and catching them early therefore has a huge leveraging effect on reducing rework.

Figure 1-3: The boundary between requirements development and requirements management.

When Bad Requirements Happen to Nice People

- Insufficient user involvement leads to late-breaking requirements that delay project completion.
- Creeping User Requirements
- Ambiguous Requirements
- Gold Platting, when a developer adds functionality that wasn't in the requirements specification
- Minimal Specification
- Overlooked User Classes
- Inaccurate Planning

Benefits from a High-Quality Requirements Process

- Fewer requirements defects
- Reduced development rework
- Fewer unnecessary features
- Lower enhancement costs
- Faster development
- Fewer miscommunications
- Reduced scope creep
- Reduced project chaos
- More accurate system-testing estimates
- Higher customer and team member satisfaction

Requirement Statement Characteristics

- Complete
- Correct
- Feasible
- Necessary
- Prioritized

- Unambiguous
- Verifiable

Requirements Specification Characteristics

- Complete
- Consistence
- Modifiable
- Traceable

Good practices for requirements engineering

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

Steps in Requirements Engineering Process

The requirements engineering process is an iterative process that involves several steps, including:

- **Requirements Elicitation**

This is the process of gathering information about the needs and expectations of stakeholders for the software system. This step involves interviews, surveys, focus groups, and other techniques to gather information from stakeholders.

- Requirements Analysis

This step involves analyzing the information gathered in the requirements elicitation step to identify the high-level goals and objectives of the software system. It also involves identifying any constraints or limitations that may affect the development of the software system.

- Requirements Specification

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

- Requirements Validation

This step involves checking that the requirements are complete, consistent, and accurate. It also involves checking that the requirements are testable and that they meet the needs and expectations of stakeholders.

- Requirements Management

This step involves managing the requirements throughout the software development life cycle, including tracking and controlling changes, and ensuring that the requirements are still valid and relevant.

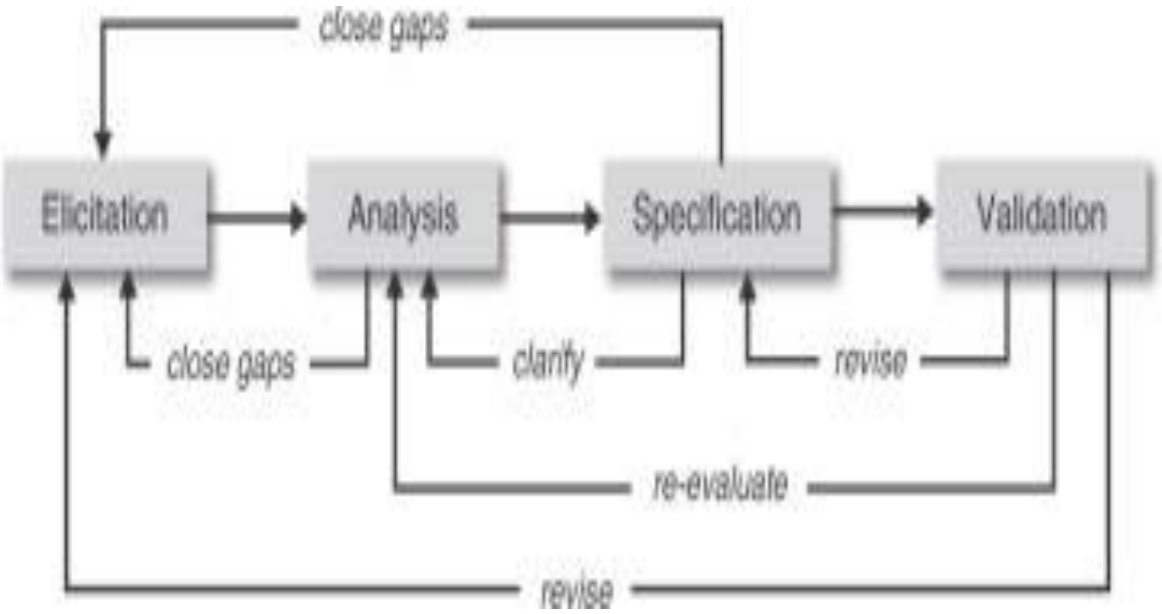
➢ Requirement Engineering

The Requirements Engineering process is a critical step in the software development life cycle as it helps to ensure that the software system being developed meets the needs and expectations of stakeholders, and that it is developed on time, within budget, and to the required quality.

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 the disciplined application of proven principle , methods ,tools and notations to describe a proposed system's intended behaviour and its associated constraints.

Elicitation	Activities to collect, discover, and invent requirements. Sometimes called gathering requirements, but elicitation is much more than a collection process.
Analysis	Activities to assess requirements for their details, value, interconnections, feasibility, and other properties to reach a sufficiently precise understanding to implement the requirements at low risk.
Specification	Activities to represent requirements knowledge in appropriate and persistent forms so that they can be communicated to others.
Validation	Activities to assess the extent to which requirements will satisfy a stakeholder need

- Requirements development is further partitioned into four subdomains:

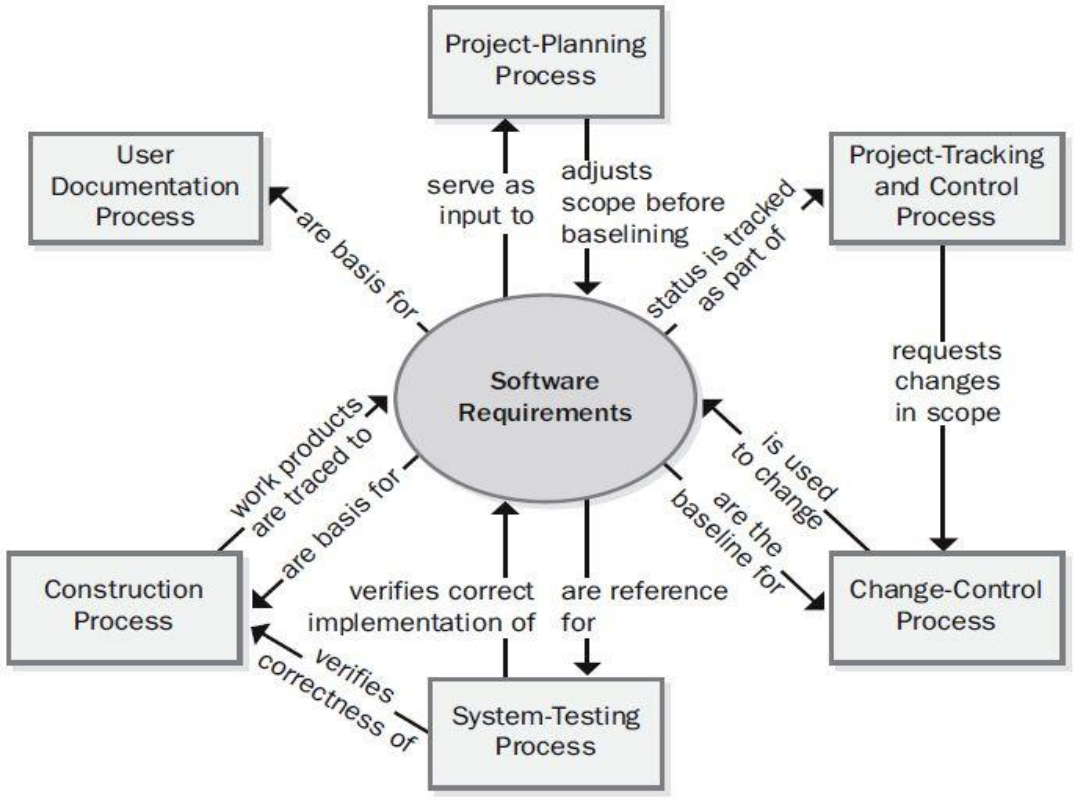


Improving requirements processes:

The ultimate objective of software process improvement is to reduce the cost of creating and maintaining software. There are several ways to accomplish this:

- Correcting problems encountered on previous or current projects.
- Anticipating and preventing problems that you might encounter on future projects.
- Adopting practices that are more efficient than the practices currently being used.

Requirements lie at the heart of every well-run software project, supporting and enabling the other technical and management activities. Figure 1 illustrates some connections between requirements and other processes; the sections that follow briefly describe these process interfaces.



Relationship of requirements to other project processes

- **Project planning**

Too often, project deadlines and staff allocations are determined before the requirements are well understood. It's no wonder then that so many projects overrun their schedules and budgets. A more realistic approach is to make requirements the foundation of the project planning process. The planners select an appropriate software development life cycle and develop resource and schedule estimates based on the requirements. Thoughtful planning might indicate that it's not possible to deliver the entire desired feature set within the available bounds of resources and time. The planning process can lead to reductions in the project scope or to selecting an incremental or iterative to deliver functionality in planned chunks.

- **Project tracking and control**.

Project tracking includes monitoring the status of each requirement so that the project manager can see whether construction and verification are proceeding as intended. If not, management might need to request a scope reduction through the change control process. If you find early on that your team isn't implementing requirements as quickly as planned, you'll need to adjust the expectations to reflect the reality of your team's productivity. Sometimes this means reallocating lower priority requirements from the backlog into later iterations than planned. It doesn't matter whether you, your managers, or your customers like this or not: that's just the way it is.

- **Change control**.

After a set of requirements has been baselined, all subsequent changes should be made through a defined change control process. The change control process helps ensure that:

- The impact of a proposed change is understood.
- All people who are affected by a change are made aware of it.
- The appropriate people make informed decisions to accept changes.
- Resources and commitments are adjusted as needed.
- The requirements documentation is kept current and accurate.

➢ **System testing**.

The testing and requirements processes are tightly coupled. User requirements and functional requirements are key inputs to system testing. If the expected behavior of the software under various conditions is not clearly specified, the testers will be hard-pressed to identify defects and to verify that all planned functionality has been implemented as intended. An excellent starting point is to start thinking about testing from the very beginning. Think of user acceptance tests for each requirement as you specify it. This is a great way to identify missing exceptions and ambiguous requirements.

➢ **Construction**.

Although executable software is the ultimate deliverable of a software project, requirements form the foundation for the design and implementation work, and they tie together the various construction work products. Use design reviews to ensure that the architecture and detailed designs correctly address all of the requirements, both functional and nonfunctional. Unit testing can determine whether the code satisfies the design specifications and the pertinent requirements. Requirements tracing lets you document the specific software design and code elements that were derived from each requirement.

➢ **User documentation**.

I once worked in an office area that also housed the technical writers who prepared user documentation for complex software-containing products. I asked one of the writers why they worked such long hours. “We’re at the end of the food chain,” she replied. “We have to respond to the final changes in user interface displays and the features that got dropped or added at the last minute.” The product’s requirements are an essential input to the user documentation process, so poorly written or late requirements will lead to documentation problems. The long-suffering people at the end of the requirements chain, such as technical writers and testers, are often enthusiastic supporters of improved requirements engineering practices.

Software requirements and risk management:

A risk is a probable problem- it might happen or it might not. There are main two characteristics of risk

Uncertainty- the risk may or may not happen which means there are no 100% risks.

loss – If the risk occurs in reality, undesirable results or losses will occur.

Risk management is a sequence of steps that help a software team to understand, analyze, and manage uncertainty.

Risk Assessment

The objective of risk assessment is to division the risks in the condition of their loss, causing potential. For risk assessment, first, every risk should be rated in two methods:

- The possibility of a risk coming true (denoted as r).
- The consequence of the issues relates to that risk (denoted as s).

Based on these two methods, the priority of each risk can be estimated:

$$p = r * s$$

Where p is the priority with which the risk must be controlled, r is the probability of the risk becoming true, and s is the severity of loss caused due to the risk becoming true. If all identified risks are set up, then the most likely and damaging risks can be controlled first, and more comprehensive risk abatement methods can be designed for these risks.

1. **Risk Identification:** The project organizer needs to anticipate the risk in the

project as early as possible so that the impact of risk can be reduced by making effective risk management planning.

A project can be of use by a large variety of risk. To identify the significant risk, this might affect a project. It is necessary to categories into the different risk of classes.

There are different types of risks which can affect a software project:

- **Technology risks:** Risks that assume from the software or hardware technologies that are used to develop the system.
- **People risks:** Risks that are connected with the person in the development team.
- **Organizational risks:** Risks that assume from the organizational environment where the software is being developed.
- **Tools risks:** Risks that assume from the software tools and other support software used to create the system.
- **Requirement risks:** Risks that assume from the changes to the customer requirement
  and the process of managing the requirements change.
- **Estimation risks:** Risks that assume from the management estimates of the resources required to build the system

2. **Risk Analysis:** During the risk analysis process, you have to consider every identified

   risk and make a perception of the probability and seriousness of that risk.

There is no simple way to do this. You have to rely on your perception and experience of previous projects and the problems that arise in them.

It is not possible to make an exact, the numerical estimate of the probability and seriousness of each risk. Instead, you should authorize the risk to one of several bands:

1. The probability of the risk might be determined as very low (0-10%), low (10-25%), moderate (25-50%), high (50-75%) or very high (+75%).

2. The effect of the risk might be determined as catastrophic (threaten the survival of the plan), serious (would cause significant delays), tolerable (delays are within allowed contingency),
   or insignificant.

Risk Control

It is the process of managing risks to achieve desired outcomes. After all, the identified risks of a plan are determined; the project must be made to include the most harmful and the most likely risks. Different risks need different containment methods. In fact, most risks need ingenuity on the part of the project manager in tackling the risk.

There are three main methods to plan for risk management:

1. **Avoid the risk:** This may take several ways such as discussing with the client to change the requirements to decrease the scope of the work, giving incentives to the engineers to avoid the risk of human resources turnover, etc.
2. **Transfer the risk:** This method involves getting the risky element developed by a third party, buying insurance cover, etc.
3. **Risk reduction:** This means planning method to include the loss due to risk. For instance, if there is a risk that some key personnel might leave, new recruitment can be planned.

Risk Leverage: To choose between the various methods of handling risk, the project plan must consider the amount of controlling the risk and the corresponding reduction of risk. For this, the risk leverage of the various risks can be estimated.

Risk leverage is the variation in risk exposure divided by the amount of reducing the risk.

Risk leverage = (risk exposure before reduction - risk exposure after reduction) / (cost of reduction)

1. **Risk planning:** The risk planning method considers each of the key risks that have been identified and develop ways to maintain these risks.

For each of the risks, you have to think of the behavior that you may take to minimize the disruption to the plan if the issue identified in the risk occurs.

You also should think about data that you might need to collect while monitoring the plan so that issues can be anticipated.

Again, there is no easy process that can be followed for contingency planning. It rely on the judgment and experience of the project manager.

2.Risk Monitoring: Risk monitoring is the method king that your assumption about the product, process, and business risks has not changed.

UNIT-II

Software Requirements Engineering:

What does “software requirements engineering” mean exactly? While it is an integral phase in software engineering, it is not well understood. Software requirements engineering refers to the first phase, before any of the actual designing, coding, testing, or maintenance takes place.

The goal is to create an important early document and process in the software design. Often referred to as software requirements specification, or SRS, it determines what software is produced. It is basically the gathering of information of a customer's or potential customer/target audience's requirements for a system, before any actual design takes place. Software requirements specifications give feedback to the potential customer, break down any problems into smaller parts, and also provide input toward the actual design.

The customer will have clear information with the help of charts, flow charts, diagrams and tables to be certain that the software will serve the intended purpose. It will also have the system requirements entailed in the document. Sometimes these are presented together with the customer requirements, and sometimes they are separate.

Recording the information in an organized manner helps cement ideas and isolates any potential problems that need to be worked on, before they can interrupt the process or create any problems. The requirements will act as a “parent document” to later plans, such as design specifications or testing and verification. It includes functional requirements, performance requirements, user requirements, input requirements, design requirements, operational and principal requirements, as well as constraints. Only when the

requirements are defined, can the actual designing begin.

Requirements Elicitation:

Requirements elicitation is the process of gathering and defining the requirements for a software system. The goal of requirements elicitation is to ensure that the software development process is based on a clear and comprehensive understanding of the customer's needs and requirements. This article focuses on discussing Requirement Elicitation in detail.

What is Requirement Elicitation?

Requirements elicitation is perhaps the most difficult, most error-prone, and most communication-intensive software development.

1. It can be successful only through an effective customer-developer partnership. It is needed to know what the users require.
2. Requirements elicitation involves the identification, collection, analysis, and refinement of the requirements for a software system.
3. It is a critical part of the software development life cycle and is typically performed at the beginning of the project.
4. Requirements elicitation involves stakeholders from different areas of the organization including business owners, end-users, and technical experts.
5. The output of the requirements elicitation process is a set of clear, concise, and well-defined requirements that serve as the basis for the design and development of the software system.

Requirements Elicitation Activities

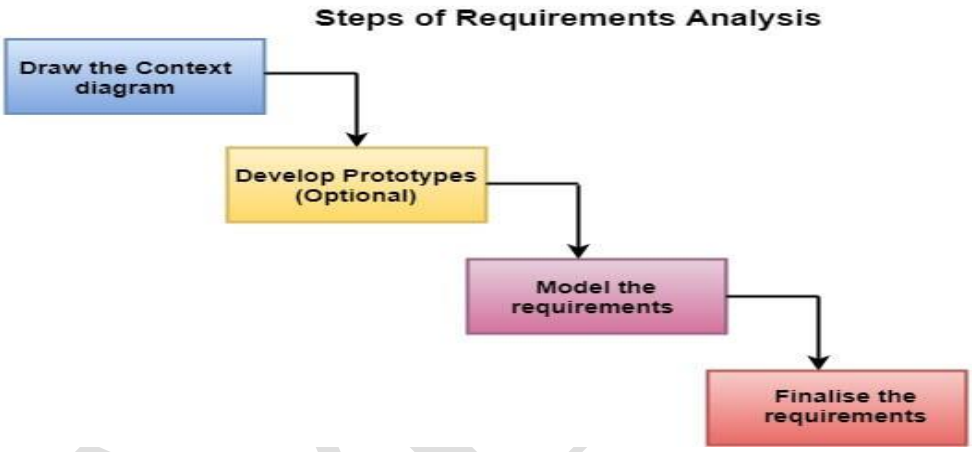
Requirements elicitation includes the subsequent activities. A few of them are listed below:

1. Knowledge of the overall area where the systems are applied.
2. The details of the precise customer problem where the system is going to be applied must be understood.
3. Interaction of system with external requirements.
4. Detailed investigation of user needs.
5. Define the constraints for system development.

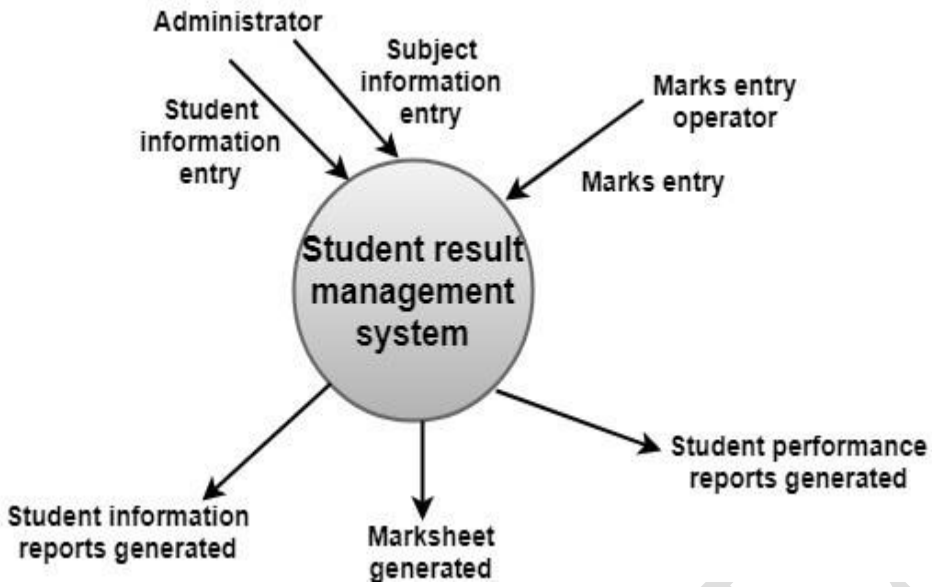
Requirements Analysis:

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 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) 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.

Review:

What are requirement reviews?

In short, Requirement review is the practice of scanning the software errors to make the industry user-friendly for all.

Why is requirement review performed?

Software in the current time is so Advanced that the act of requirement review holds greater importance in software development. The pursuance of requirement reviews helps to have a clear peek into the space of the software industry. The requirement reviews call attention to the only chance of finding quality reviews. So keeping in mind that there are problems and solutions to everything, a requirement review needs a good performance.

Importance of performing requirement review :

- The performance of requirement review helps to radiate the precise and correct data to the consumers and users.
- It helps to have a quick tour of the Existing project to see whether or not it is going in the right direction.
- It helps to provide practical instructions and helps make decisions accordingly.

Methods of performing requirement review :

1. Team consultation:

Suggestion matters also matter the way of performance. Teamwork goes hand in hand. When there are people to offer suggestions, give appropriate guidelines, and supervise in a team. There is no doubt about the project getting mismanaged. Reaching out to the team/individual who has better insights into requirement review works the best way.

2.Understanding the user's requirement:

Recognize the user's needs and go all out in understanding them. Requirements keep on changing with time. So, When you have collected a list of things that a user requires in the current time. There you found a way to go about it. To get exact information on their requirements, Asking for feedback is Important.

2. Finding measures to software problem:

The occurrence of software problems is predictable. Errors and defects are bound to take place in software development. In this context, Rather than making a fuss about the Problems, developers should find solutions to satisfy the requirements. The requirement review not only meets the expectations of users but also the standard of the entire industry.

Advantages of performing requirement reviews :

- Requirement reviews accord the developers a motive and structure to carry out the project further.
- Group collaboration is the highlight. Group work saves time.
- Therefore, the developers can utilize the saved time in rechecking and reconfirming the processing work to take it ahead.

Disadvantages of performing requirement reviews :

- Lack of attention acts as a hindrance. When a team does not listen to each other in a meeting room because of disagreement on matters, it emerges as a sign of unprofessional and uncoordinated work.
- At times, the Review cannot be accurate. So, If you fail in assembling the precise information, it can be an obstacle for the developers and the industry.

Elicitation Techniques:

What Is Requirements Elicitation?

It is all about obtaining information from stakeholders. In other words, once the business analysis has communicated with stakeholders for understanding their requirements, it can be described as elicitation. It can also be described as a requirement gathering.

Requirement elicitation can be done by communicating with stakeholders directly or by doing

some research, experiments. The activities can be planned, unplanned, or both.

- **Planned activities** include workshops, experiments.
- **Unplanned activities** happen randomly. Prior notice is not required for such activities. **For example**, you directly go to the client site and start discussing the requirements however there was no specific agenda published in advance.

Following tasks are the part of elicitation:

- **Prepare for Elicitation:** The purpose here is to understand the elicitation activity scope, select the right techniques, and plan for appropriate resources.
- **Conduct Elicitation:** The purpose here is to explore and identify information related to change.
- **Confirm Elicitation Results:** In this step, the information gathered in the elicitation session is checked for accuracy.

We hope, you have got an idea about requirement elicitation by now. Let's move on to the requirements elicitation techniques.

Requirements Elicitation Techniques

1) Stakeholder Analysis

Stakeholders can include team members, customers, any individual who is impacted by the project or it can be a supplier. Stakeholder analysis is done to identify the stakeholders who will be impacted by the system.

2) Brainstorming

This technique is used to generate new ideas and find a solution for a specific issue. The members included for brainstorming can be domain experts, subject matter experts. Multiple ideas and information give you a repository of knowledge and you can choose from different ideas.

This session is generally conducted around the table discussion. All participants should be given an equal amount of time to express their ideas.

3) Interview

This is the most common technique used for requirement elicitation. Interview techniques should be used for building strong relationships between business analysts and stakeholders. In this technique, the interviewer directs the question to stakeholders to obtain information. One to one interview is the most commonly used technique.

If the interviewer has a predefined set of questions then it's called **a structured interview.** If the interviewer is not having any particular format or any specific questions then it's called an unstructured interview.

4) Document Analysis/Review

This technique is used to gather business information by reviewing/examining the available materials that describe the business environment. This analysis is helpful to validate the implementation of current solutions and is also helpful in understanding the business need.

Document analysis includes reviewing the business plans, technical documents, problem reports, existing requirement documents, etc. This is useful when the plan is to update an existing system. This technique is useful for migration projects.

5) Focus Group

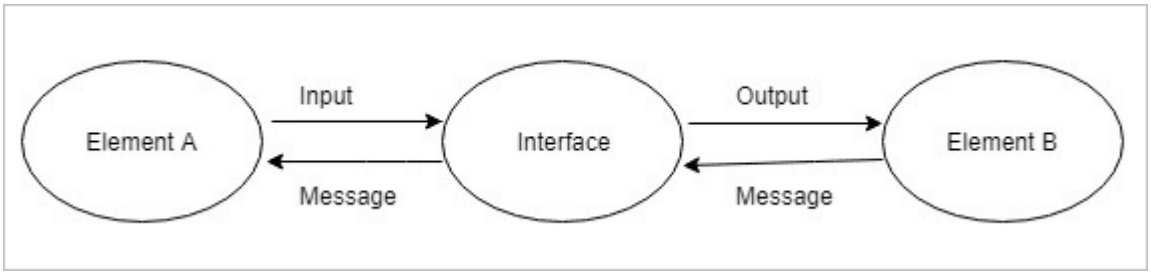
By using a focus group, you can get information about a product, service from a group. The Focus group includes subject matter experts. The objective of this group is to discuss the topic and provide information. A moderator manages this session.

The moderator should work with business analysts to analyze the results and provide findings to the stakeholders.

6) Interface Analysis

Interface analysis is used to review the system, people, and processes. This analysis is used to identify how the information is exchanged between the components. An Interface can be described as a connection between two components.

This is described in the below image:



7) Observation

The main objective of the observation session is to understand the activity, task, tools used, and events performed by others.

The plan for observation ensures that all stakeholders are aware of the purpose of the observation session, they agree on the expected outcomes, and that the session meets their expectations. You need to inform the participants that their performance is not judged.

8) Prototyping

Prototyping is used to identify missing or unspecified requirements. In this technique, frequent demos are given to the client by creating the prototypes so that client can get an idea of how the product will look like. Prototypes can be used to create a mock-up of sites, and describe the process using diagrams.

9) Joint Application Development (JAD)/ Requirement Workshops

This technique is more process-oriented and formal as compared to other techniques. These are structured meetings involving end-users, PMs, SMEs. This is used to define, clarify, and complete requirements.

This technique can be divided into the following categories:

- **Formal Workshops:** These workshops are highly structured and are usually conducted with the selected group of stakeholders. The main focus of this workshop is to define, create, refine, and reach closure on business requirements.
- **Business Process Improvement Workshops:** These are less formal as compared to the above one. Here, existing business processes are analyzed and process improvements are identified.

10) Survey/Questionnaire

For Survey/Questionnaire, a set of questions is given to stakeholders to quantify their thoughts. After collecting the responses from stakeholders, data is analyzed to identify the area of interest of stakeholders.

Questions should be based on high priority risks. Questions should be direct and unambiguous. Once the survey is ready, notify the participants and remind them to participate.

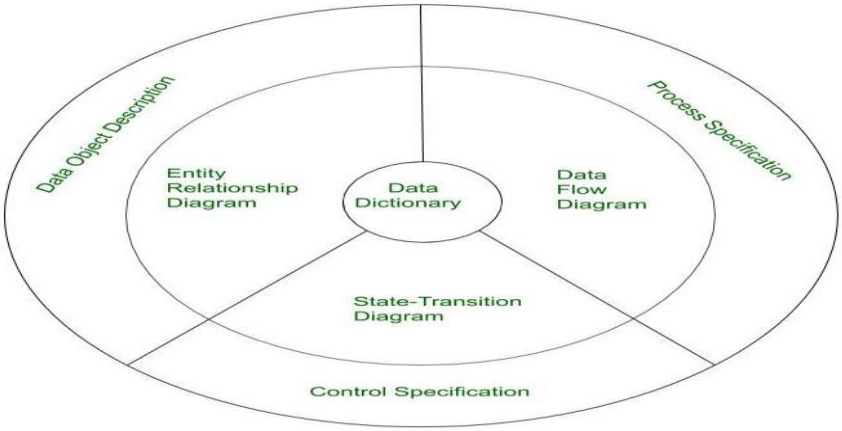
Analysis Model :

Analysis Model is a technical representation of the system. It acts as a link between system description and design model. In Analysis Modelling, information, behavior, and functions of the system are defined and translated into the architecture, component, and interface level design in the design modeling.

Objectives of Analysis Modelling:

- It must establish a way of creating software design.
- It must describe the requirements of the customer.
- It must define a set of requirements that can be validated, once the software is built.

Elements of Analysis Model:



- **DataDictionary:**
  It is a repository that consists of a description of all data objects used or produced by the software. It stores the collection of data present in the software. It is a very crucial element of the analysis model. It acts as a centralized repository and also helps in modeling data objects defined duringsoftwarerequirements.

- **EntityRelationshipDiagra(ERD):**
  It depicts the relationship between data objects and is used in conducting data modeling activities. The attributes of each object in the Entity-Relationship Diagram can be described using Data object description. It provides the basis for activity related to data design.

- **DataFlowDiagram(DFD):**
  It depicts the functions that transform data flow and it also shows how data is
- transformed when moving from input to output. It provides the additional information which is used during the analysis of the information domain and serves as a basis for the modeling of function. It also enables the engineer to develop models of functional and information domainsat the same time.

- **StateTransitionDiagram:**
  It shows various modes of behavior (states) of the system and also shows the transitions from one state to another state in the system. It also provides the details of how the system behaves due to the consequences of external events. It represents the behavior of a system by presenting its states and the events that cause the system to change state. It also describes what actions are taken due to the occurrence of a particular
  event.

- **Process Specification:**
  It stores the description of each function present in the data flow diagram. It describes the input to a function, the algorithm that is applied for the transformation of input, and the output that is produced. It also shows regulations and barriers imposed on the performance characteristics that are applicable to the process and layout constraints that could influence the way in which the process will be implemented.

- **ControlSpecification:**
  It stores additional information about the control aspects of the software. It is used to indicate how the software behaves when an event occurs and which processes are invoked due to the occurrence of the event. It also provides the details of the processes which are executed to manage events.

- **DataObjectDescription:**
  It stores and provides complete knowledge about a data object present and used in the software. It also gives us the details of attributes of the data object present in the Entity Relationship Diagram. Hence, it incorporates all the data objects and their attributes.

Software Quality Attributes:

Software Quality shows how good and reliable a product is. To convey an associate degree example, think about functionally correct software. It performs all functions as laid out in the SRS document. But, it has an associate degree virtually unusable program. Even though it should be functionally correct, we tend not to think about it to be a high-quality product.

Another example is also that of a product that will have everything that the users need but has an associate degree virtually incomprehensible and not maintainable code. Therefore, the normal construct of quality as “fitness of purpose” for code merchandise isn’t satisfactory.

Factors of Software Quality

The modern read of high-quality associates with software many quality factors like the following:

1. **Portability:** A software is claimed to be transportable, if it may be simply created to figure in several package environments, in several machines, with alternative code merchandise, etc.
2. **Usability:** A software has smart usability if completely different classes of users (i.e. knowledgeable and novice users) will simply invoke the functions of the merchandise.
3. **Reusability:** A software has smart reusability if completely different modules of the merchandise will simply be reused to develop new merchandise.

4. **Correctness:** Software is correct if completely different needs as laid out in the SRS document are properly enforced.
5. **Maintainability:** A software is reparable, if errors may be simply corrected as and once they show up, new functions may be simply added to the merchandise, and therefore
   the functionalities of the merchandise may be simply changed, etc
6. **Reliability.** Software is more reliable if it has fewer failures. Since software engineers do not deliberately plan for their software to fail, reliability depends on the number and type of mistakes they make.
7. **Efficiency.** The more efficient software is, the less it uses of CPU-time, memory, disk space, network bandwidth, and other resources. This is important to customers in order to reduce their costs of running the software, although with today's powerful computers, CPU time, memory and disk usage are less of a concern than in years gone

Risk Reduction Through Proto typing:

Prototyping in software development plays a significant role in reducing risks associated with software requirements. Here's how it helps in mitigating such risks:

1. **Clarification of Requirements:** Often, clients or stakeholders may not have a clear and comprehensive understanding of their requirements at the project's outset. Prototyping
2. allows for the visualization of the proposed software, making it easier for stakeholders to
3. grasp functionalities, interfaces, and workflows. This helps in eliciting and refining requirements more accurately, reducing misunderstandings and ambiguities.
4. **Validation of Requirements:** Prototyping enables stakeholders to interact with a tangible representation of the software early in the development process. By demonstrating the prototype, it becomes easier to validate whether the requirements align with the
5. stakeholders' expectations. Any discrepancies or misunderstandings can be identified and rectified in the early stages, reducing the risk of developing a software product that does not meet user needs.
6. **Gathering User Feedback:** Prototypes can be used to gather feedback from end-users, stakeholders, and potential customers. By involving these parties in the evaluation of the prototype, developers can understand user preferences, identify missing or conflicting requirements, and make necessary adjustments. This user involvement minimizes the risk of building software that doesn't meet user expectations.
7. **Reducing Change Requests:** Prototyping allows for early exploration of different design and functionality options. Through iterative development and feedback cycles, changes can be incorporated into the prototype, reducing the likelihood of major change requests later in the development process. This helps in managing scope creep and controlling

additional costs or delays caused by late-stage requirement modifications.

8. **Managing Uncertainty:** Software projects often face uncertainty due to evolving or unclear requirements. Prototyping provides a means to manage this uncertainty by allowing flexibility in accommodating changes. As requirements become clearer through the prototyping iterations, the risk associated with changing or evolving requirements diminishes.
9. **Enhancing Communication:** Prototypes serve as a communication tool between stakeholders, including developers, designers, clients, and end-users. They facilitate discussions, ensuring everyone has a shared understanding of the requirements and functionalities. Improved communication reduces the risk of misunderstandings or misinterpretations that could impact the final product.

In summary, prototyping in software development contributes significantly to requirement risk reduction by clarifying and validating requirements, gathering early feedback, managing uncertainty, reducing change requests, and enhancing communication among stakeholders. This approach helps in creating software that better aligns with user needs, reducing the likelihood of costly rework and improving the overall success of the project.

Setting Requirements Priorities:

Most stakeholders view all requirements as equally important. They don't always understand that development teams are working with limited time and resources. This can cause friction between the different parties. By performing requirements prioritization techniques, you can ease
that tension and help stakeholders understand the limitations of development teams. Developers and stakeholders should collaborate when creating requirements prioritization strategies. This allows you to explore various alternatives when conflict arises.

Why is requirements prioritization important?

In an ideal world, dev teams will have all the resources, time, and money they want to put into

every project. Unfortunately, the reality is that teams often battle against limitations that force them to compromise on various aspects of development.

Without prioritization, teams have much less wiggle room when it comes to changing requirements. They need to know how to prioritize requirements to avoid missing deadlines, going over budget, or dropping requirements altogether. You can deliver a good product that satisfies customers and performs below business expectations with prioritization.

Requirements prioritization techniques allow development teams to make compromises that don't detract from the product's value. It helps them better manage resources and requirements and prepare for unknowns. Say your requirements prioritization shows that some requirements are unfeasible given the allocated resources. The development team can collaborate with the stakeholders to identify the best way to proceed with the project.

The more you go through requirement prioritization, the better your stakeholders will understand the limitations the dev team faces. This will help to make collaboration much more streamlined and manage expectations for the future.

What are 3 ways to prioritize requirements?

With so many prioritization techniques available, it can be difficult to know how to prioritize requirements. Thankfully, you can use some familiar prioritization frameworks that you're likely familiar with already.

MoSCoW method

MoSCoW prioritization is a great tool for establishing a hierarchy of priorities during a project. It solves one of the biggest issues of less robust prioritization tools by laying out specific definitions for each priority level.

MoSCoW is an acronym that gives us four prioritization categories: **Must**-have,

Should- have, **Could**-have, and **Won't**-have.

This is a great choice for requirements prioritization because it clearly labels each requirement This helps stakeholders quickly understand how requirement priorities are affected by resource limitations.

ICE scoring

ICE scoring prioritizes requirements using three set measurements: Impact, Confidence, and Ease. This requirement prioritization technique differs from weighted prioritization methods by using just these three parameters and assigning each a relative score.

ICE is one of the quicker and easier requirement prioritization strategies, though it does suffer from being quite subjective.

Kano analysis

The Kano Model looks at which features will be most important to customer satisfaction levels. It looks purely from an outside perspective to identify which requirements will truly add value for the user.

This is highly useful for requirements prioritization, as it helps manage limited resources without detracting from the user experience.

What is the agile method for prioritizing requirements?

As with anything in the agile world, there are several different methods for prioritizing requirements in an agile setting.

Priority poker

There's no reason that planning and prioritization need to be dull. Priority poker allows teams to prioritize requirements *and* have a little fun in the process. It's also one of the best requirements prioritization techniques for eliminating subconscious bias like the HiPPO effect.

Cost of Delay (CoD)

Cost of Delay is a crucial metric that emphasizes development time for each requirement. It places a monetary value on any delays and tasks the team with prioritizing requirements with time frames in mind.

Opportunity Scoring

Opportunity scoring is another requirements prioritization technique that focuses on the user. The results of opportunity scoring allow teams to dedicate more time and resources to requirements that customers really need.

Verifying RequirementsQuality:

The main goals of requirements verification are to ensure completeness, correctness, and consistency of the system requirements.

This phase can uncover missing requirements, ambiguous, or invalid ones, reducing rework and cost overruns. It's far more effective to resolve a little problem upfront than it is in the future when hundreds of lines of code or a completely manufactured complex product must be tracked down and fixed.

Requirements verification is necessary because it helps ensure that the system to be built will meet its objectives and functions as intended. Incomplete, incorrect, or inconsistent requirements can lead to problems during system development, testing, and deployment.

Techniques Used in Requirements Verification:

There are several techniques that can be used for requirements verification to ensure that the requirements meet the necessary quality criteria. Some of the commonly used

techniques include:

1. **Inspection:** This technique involves a systematic review of the requirements by a team of experts to identify any errors, omissions, or inconsistencies. It can be conducted manually or using automated tools.
2. **Testing:** Testing involves designing and executing tests to verify that the requirements meet the desired functionality and quality criteria. It can be conducted at different levels, such as unit testing, integration testing, and acceptance testing.
3. **Walkthrough:** In a walkthrough, the requirements are reviewed by a group of stakeholders who provide feedback and identify any issues or concerns. It is typically less formal than an inspection.
4. **Prototyping:** Prototyping involves creating a simplified version of the software to validate the requirements and identify any issues or limitations. It can help stakeholders visualize and understand the system better.
5. **Simulation:** Simulation involves creating a model of the system and testing its behaviour under different scenarios. It can help identify issues with the requirements that may not be apparent in static documentation.
6. **Traceability Analysis:** Traceability analysis involves tracking the relationships between the requirements and other artifacts, such as design documents and test cases, to ensure that the requirements are complete, consistent, and verifiable.

Software Requirements Modeling:

Major styles of requirement modeling currently in practice are as follows – scenario-based model, data or flow model, class model, and behavioral model.

Scenario-based model - This model is prepared from an end-user perspective. Techniques such as use cases, user stories, and activity diagrams would be a major contributor to this model.

Flow-based model – When a requirement deals with a specific data flow that affects various modules of a System data-based modeling is the best method that can accommodate the system flow and the data flow within the system. This could be a tedious model but it is a helpful model to do impact analysis. Data Flow diagrams, activity diagrams are types of this model.

Class-based model – Class diagrams are the most popular UML diagrams used for the construction of software applications. They are a graphical representation of the static view of the system and represent different aspects of the application. This involves identifying the classes such as the event occurrences, roles, places, structures, and other artifacts involved.

Behavioral model – This modeling happens from a user perspective. Mainly state diagram and sequence diagram builds up this model.

In short, the 5 common types that make up a requirement model are use case, user stories, activity diagram, flow diagram, state diagram, and sequence diagram.

The different modeling technique is individually understandable. However, it is highly important to know how to use these models and when to use them. In some cases, certain models may be more appropriate than the others may, whereas in other cases, almost all the models may be applicable with a difference in weightage.

There are 5 major aspects to consider when it comes to deciding on the usage of requirement models.

The five aspects are:

- Nature of the requirement
- Type of requirement
- Impact of the requirement in the system
- Users related to the requirement
- The phase of the process that would deal with the requirement

Nature of the requirement – This aspect deals with identifying if a given requirement is an enhancement requirement or a foundation requirement for a new system or a process

Type of requirement – Three major types of requirements that could be considered for deciding the usage of requirements models are a functional requirement, performance requirement, and technical

requirement. The functional requirement deals with the operation processes, where the change of state of various parameters would result in the required end-point. Performance requirements are add-ons of an existing process or for a functional requirement, to enhance the process efficiency. Technical requirements can either be allocated requirement that automatically flows from a functional or performance requirement to an element of the system or is a derived requirement that deals with the interfaces between the various stages within a system.

Impact of the requirement in the system – An impact of a requirement in a system could be considered in three different fashions – minimum, medium, and maximum. A minimum impact could be defined as an impact that changes the user experience or the system parameter for a specific instance. An impact would be considered medium if the requirement is indirectly affecting the other related process or system parameters. A maximum impact would be considered where the majority of the parameters of the system or the related processes appear transformed.

Users related to the requirement – Three major types of users would be business users, functional users, and technical users. Business users would have the objective of knowing the business value proposition that would be obtained if the endpoint of the requirement is reached. Functional users would be oriented towards understanding the

changes or the impact that could be expected in the direct and indirect process related to the requirement where the technical concern is not a major aspect. Technical users would be aligned towards the development estimation of effort and risk for the given requirement initially and later during the development, they would need the detail related to the nuances to achieve the required endpoint.

Phases of a process – The common phases of a process would be the following-
Initial understanding phase Work estimation & Overall impact analyzing phase, Work initiation, Work in progress, Work in last phase, Quality analysis, End-user acceptance, Delivery.The judiciousness of choosing the modeling technique depends on choosing the priority of having those five elements of modeling based on the variation in the five aspects of a requirement discussed above.or each of the modeling types, here's how the above five aspects are considered:

Use Case: This technique would be suitable when the requirement is functional in-nature. Use cases depic the high-level functionalities that one would want the system to perform. This technique would be useful to get a foundation level understanding or a holistic style of the requirement. More than a business user
his would be handy for a functional or technical user. This would play a major role in defining the scope of work, risk, and effort estimation for a given requirement.

Activity diagram – This technique deals with the overall business process or system

process – which is suitable for all types of users in line with the nature of requirement being functional and the type being foundational. This technique can aid in impact analysis by just defining the system or process scope but this technique can't be of assistance for detailed impact analysis.

Flow Diagram– This technique is helpful for a specific functional segment of a system or a process. So enhancement requirements of any nature would fit in this technique that would be easy for functional and technical users. After having defined the scope of the system or the process, this flow chart will guide in analyzing the impact in detail. On this front, technical and functional users are the ones who use the flow diagram for their work more than the business users. As a process, the requirement represented as a flow diagram would be highly utilized until the last phase of the development.

State diagram – This technique is more specific compared to a flow chart. Regarding the objects of a system or the process, the various states of an object that happens during a process flow are depicted only in a state diagram. In line with this, only technical and functional users would find this useful especially during the development phase more than the initial and final phases. This element can't be a direct contributor to the impact analysis parameter.

Sequence diagram – This is more relevant for a technical user, especially when many processes happen in parallel. It provides a visual representation of how processes or objects

interact during a scenario.

This technique adds more value for technical users, as this will help in drilling down to detailed technical specifications. On the other side, this is the most preferred methodology for requirement reference majorly during the development phase. Here's a summary of the above-stated aspects in the form of 'Requirements modeling techniques' vs 'Usage aspects' matrix –

Requirement modeling techniques	**Nature of requirement**	**Requirement impact**
User Stories	All category	Doesn't reflect any impact
Use case	Preferred for functional requirement	Reflects the scope/outline for the impact sections
Activity diagram	Preferred for functional requirement	Reflects the scope/outline for the impact sections
Flow diagram	Preferred for functional & technical requirement	Preferred for high impact requirement

State diagram	Preferred for functional & technical requirement	All levels of impact would be reflected. Mandatory for high impact requirements
Sequence diagram	Preferred for functional & technical requirement	All levels of impact would be reflected. Mandatory for high impact requirements

ObjectAnalysis:

Object-Oriented Analysis and Design (OOAD) is a software engineering methodology that involves using object-oriented concepts to design and implement software systems. OOAD involves a number of techniques and practices, including object-oriented programming, design patterns, UML diagrams, and use cases. Here are some important aspects of OOAD:

1. Object-Oriented Programming: Object-oriented programming involves modeling real-world objects as software objects, with properties and methods that represent the behavior of those objects. OOAD uses this approach to design and implement software systems.
2. Design Patterns: Design patterns are reusable solutions to common problems in software design. OOAD uses design patterns to help developers create more maintainable and efficient software systems.
3. UML Diagrams: Unified Modeling Language (UML) is a standardized notation for creating diagrams that represent different aspects of a software system. OOAD uses UML diagrams to represent the different components and interactions of a software system.
4. Use Cases: Use cases are a way of describing the different ways in which users interact with a software system. OOAD uses use cases to help developers understand the requirements of a system and to design software systems that meet those requirements.

Object-Oriented Analysis (OOA) is the first technical activity performed as part of object-oriented software engineering. OOA introduces new concepts to investigate a problem. It is based on a set of basic principles, which are as follows-

1. The information domain is modeled.
2. Behavior is represented.
3. The function is described.
4. Data, functional, and behavioral models are divided to uncover greater detail.
5. Early models represent the essence of the problem, while later ones
6. provide implementation details.

the above notes principles form the foundation for the OOA approach.

Problem Frames:

Problem Frames is a methodology used in software engineering for understanding and solving complex problems. It was developed by Michael Jackson, not the musician but a British computer scientist, in the 1990s.

The main idea behind Problem Frames is to analyze problems in a systematic way by breaking them down into smaller, manageable components called "frames." Each frame represents a particular aspect or viewpoint of the problem, allowing engineers to focus on specific areas and understand the problem from different perspectives.

Some key components and concepts within Problem Frames include:

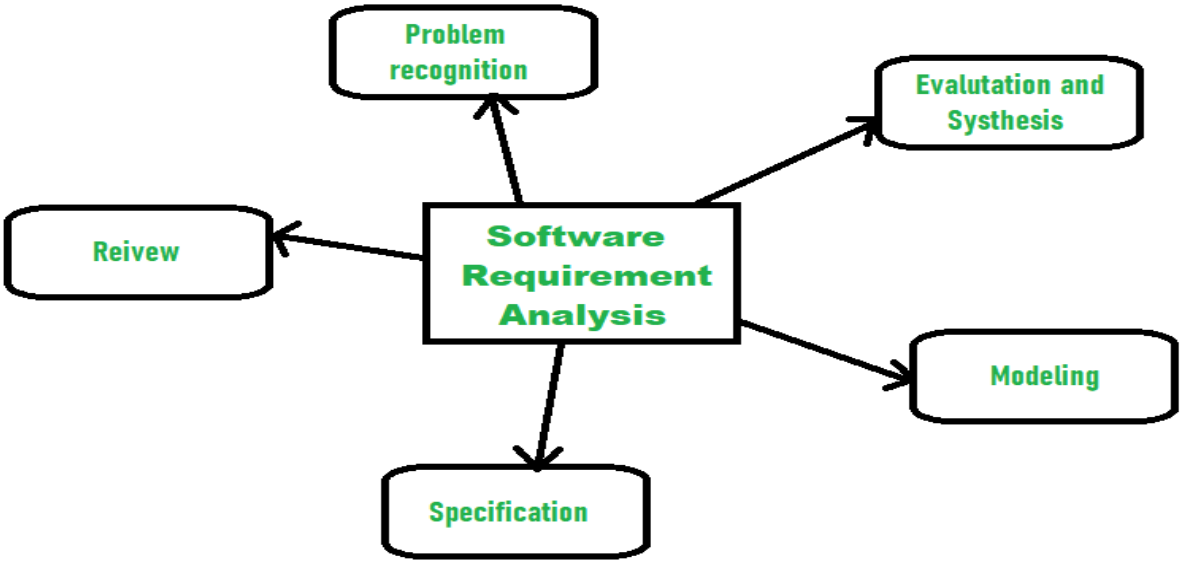
1. **Context**: Understanding the environment or context in which the problem exists. This involves identifying stakeholders, constraints, and relevant external factors.
2. **Goals**: Defining the objectives or goals that need to be achieved. This step involves understanding what success looks like for the system or project being developed.
3. **Decomposition**: Breaking down the problem into smaller, more manageable frames. Each frame focuses on a specific aspect of the problem, making it easier to analyze and solve.
4. **Frames**: These are the individual components that represent different aspects or perspectives of the problem. Each frame provides a structured way of looking at a particula issue within the broader problem.
5. **Relations between Frames**: Understanding how the different frames interact with each other and influence one another is crucial. Identifying dependencies and relationships helps in developing a holistic solution.
6. **Solution Context**: Considering the context in which the solution will exist. This involves understanding the implications, constraints, and requirements of implementing the solution.

Problem Frames provide a structured approach to problem-solving in software engineering, helping teams to systematically analyze complex problems and design effective solutions. The methodology emphasizes breaking down the problem into manageable parts, ensuring a comprehensive understanding before attempting to solve it.

Unit-III

Software Requirement Management

The requirement management process is the process of managing changing requirements during the requirements engineering process and system development where the new requirements emerge as a system is being developed and after it has gone into use. During this process, one must keep track of individual requirements and maintain links between dependent requirements so that one can assess the impact of requirements changes along with establishing a formal process for making change proposals and linking these to system requirements



It belongs to one of the phases of the Requirement Engineering Process.

Now during this phase, there needs to be a certain level of requirement management details which will help to make Requirement Management decisions. To accumulate the details for taking that decision one can follow the following processes:

- **Requirements Identification:** In this, the requirement must be uniquely identified so that it can be cross-referenced with other requirements. Here, one can learn what is important and required and what is not and it also helps to establish a foundation for product vision, scope, cost, and schedule.
- **Requirement change management process:** This is the set of activities that assess the impact and cost of changes.\

- **Traceability policies:** The main purpose of this policy is to keep a record of the defined relationships between each requirement and the system designs which will help to minimize the risks.

Tool support: Tools like MS Excel, spreadsheets,or a simple database system can be used.

Advantages of the Requirement Management Process:

1. Recognizing the need for change in the requirements.
2. Improved team communication.
3. It helps to minimize errors at the early stage of the development cycle.

Requirement Engineering Principals:

1. **Value Orientation:** The act of writing requirements based on assumptions has no value, neither for the users nor for your business. The requirements outcome need to add value for the users. Remember, users use products to do their job better. The requirements should be something that adds value to the outcome and the benefit of using your product in end-users life.
2. **Stakeholders:** Using Requirement Engineering helps you do your best to satisfy the stakeholders' needs and desires. When we speak about developing a digital product, many product people think they should just deliver a product based on end-user needs.
3. **Critical:** Most important ones. If you don't consider engaging them, you are going to develop a useless product, and they can make the product fail.
4. **Major:** These are the ones that have an important role in product success. They have a great impact on a product's success, but by not considering them, the product won't fail.
5. **Minor:** Not considering these stakeholders will not have an impact or have a minor impact on product success.

Requirement Engineering Practices:

1. **Eliciting requirements: collaborate & focus on value:**

The gathering of requirements should be a highly collaborative effort. Therefore, even before you start thinking about the product itself, you'll have to map all the relevant stakeholders who have interest in and influence over the project**.**

2. **Requirements quality translates to software quality:**

When defining requirements, focus on delivering value, and make sure that everyone involved shares an understanding of 'value' in your project. This, in turn, helps avoid scope creep: you'll want to make sure that the scope of the project is clearly defined and documented so that it doesn't spiral out of control.

3. **Prioritize requirements and set expectations:**

Requirements definition should be an iterative process, which greatly helps the next step. Set up a priority list of your requirements based on their value, and make sure all stakeholders agree with the final list. Having this prioritized list of requirements provides clarity, making it easier for your team to set realistic expectations with the customer/end user on what is to be delivered.

4. **Trace requirements through the lifecycle:**

Follow the progress of requirements along the development lifecycle. Tie requirements in with tasks, source code, risks, and test cases so that you have airtight traceability from end to end.

5. **Use a dedicated tool for managing requirements:**

No high-performing engineering team relies on inadequate tooling, and more and more companies consider updating their toolchains a strategic step that will increase their

profitability.

Requirements Attributes:

Requirements Attributes: A *requirement attribute* is a descriptive property associated with a requirement. Requirement attributes are either user-defined attributes (defined by the project owner) or system attributes To avoid causing errors in requirement records or in integrations with other products, do not modify system attributes.

Examples of Attributes:

Priority - Statement of relative importance of the requirement to stakeholders (high, medium, low).

Assigned to - Who in the organization is responsible for making sure the requirement is met (person's name or organizational name).

Target Iteration – The iteration in which the requirement is planned to be implemented (number or text).

Estimation of Size - Gives you a high-level estimate for the effort required to implement and verify the requirement, typically measured using a neutral unit such as points.

Effort Remaining – An estimate of the remaining effort to implement and verify the requirement (hours).

Completion Status – The progress of implementing a requirement. This may be captured as an enumerated list (Complete, Partially Completed, Not Started) or can be inferred from the Effort Remaining attribute.

Source - Person, document or other origin of a given requirement. This is useful for determining whom to call for questions or for grouping requirements according to the person making the demands.

Comments - Reviewer's or writer's comments on a requirement.

Difficulty - An indication of the level of effort needed or how hard it will be to implement the requirement (high, medium, low).

Risk - Confidence measure on the likelihood of meeting (or not meeting) a requirement. Could be high, medium, low or the integers one through ten.

Test ID - Identification of a specific test or other method of verification.

Change Mangement Process:

What is change management?

Change management is the process that businesses and organizations use to implement changes through building and delivering effective change strategies. It includes reviewing reasons for change, implementing changes, and helping people adapt to these changes. This could be staff structure, introducing new technology, reducing costs, increasing profits, or a combination of these to reach a desired goal.

What is the change management process?

The change management process refers to the stages involved in any change management strategy and its implementation. implementing new technology into a business will not just involve the technology change itself. It may affect staffing levels, require structural changes, new recruitment drives, or even redundancies.

The change management process breaks down into the following five steps:

1. **Prepare for change:** It's an important part of the process, ensuring the change manager supports staff through any concerns and manages resistance by communicating the process and getting buy-in from employees.
2. **Create a vision for change:** This stage is about creating the strategy to reach transformation once stakeholders have agreed for a change. Those involved set goals, delegating key performance indicators (KPIs) and tasks to the relevant parties.
3. **Implement changes:** This step puts the change plans into action. Excellent management and communication are key here, and change managers need to make sure everyone is doing their duties and that employees are still happy and empowered, to ensure everything runs smoothly**.**
4. **Review and analyze:** The final stage of the process is important to make sure that changes continue and are beneficial. Change managers review what worked and what didn't work to make adjustments accordingly**.**

Types of change management

1.Anticipatory

Anticipatory change is when an organization makes changes in response to something expected to happen. For example, environmental concerns or new trends the organization wants to capitalize on can cause stakeholders to anticipate the need for change**.**

3. Reactive

Reactive change happens in response to an event that impacts the business. This could be new industry regulations or changes to deal with a pandemic like Covid-19.

3. Incremental

Incremental change is a series of changes, usually at a micro level, that adds up to wider overall changes. Examples include implementing a reward system, introducing new flexible working policies, or changing office hours.

4. Strategic

Strategic changes are made at and filtered down from a higher level and impact the whole organization.

Requirements Traceability Matrix

What is Traceability Matrix (TM)?

A Traceability Matrix is a document that co-relates any two-baseline documents that require a many-to-many relationship to check the completeness of the relationship.

What is Requirement Traceability Matrix?

Requirement traceability Matrix is a document that maps and traces user requirement with test cases. It captures all requirements proposed by the client and requirement traceability in a single document, delivered at the conclusion of the Software development life cycle.

RTM captures all requirements proposed by the client and their traceability in just one document delivered at the end of the life-cycle.

RTM usually helps to evaluate the impact of project requirements. When requirements shift in the middle of a project, a traceability matrix lets you see the impacted workflows, test cases, training materials, software code, etc.

Benefits of RTM

Versioning is Easier and More Effective: As a project manager, it's not uncommon for the requirements of your project to undergo modification at some point. RTM helps you trace these shifts and how it impacts every part of your project.

Tackling Defects: A traceability matrix can aid you in filtering defects linked to

crucial requirements, along with defect severity, priority, and more. Finally, RTM establishes complete test coverage.

How to Create Traceability Matrix?

Steps for creating a requirements traceability matrix.

1.Establish your RTM goals by laying out your reason for creating the RTM.

2.Gather all accessible requirement documentation, such as the technical requirement document (TRD) or functional requirement document (FRD), and business requirement document (BRD). You'll also need testing documentation, like test cases, results, and bugs.

3.To make a simple RTM document, you can use an Excel spreadsheet. Create columns for business requirements, functional requirements, test cases, test results, and bugs. Then, record each requirement from BRD with a requirement ID number.

4.Take the FRD and record all corresponding functional requirements for every business requirement.

5.Connect test case IDs to the corresponding functional requirements.

Types of Traceability Matrix

There are three types of RTM

1. **Forward Traceability**

Forward traceability is used to map the requirements to the test cases. Not only will this establish that every requirement is being tested from top to bottom, but it will also assist in confirming that a project's trajectory is sound.

2. **Backward Traceability Matrix**

You can make a backward traceability matrix by mapping test cases with the requirements. Doing so aids you in avoiding "scope creep" and going beyond the initial requirements unnecessarily.

3. **Bidirectional Traceability**

Bidirectional traceability essentially combines forward and backward traceability into one document. This type is useful because it establishes that each requirement has relating test cases.

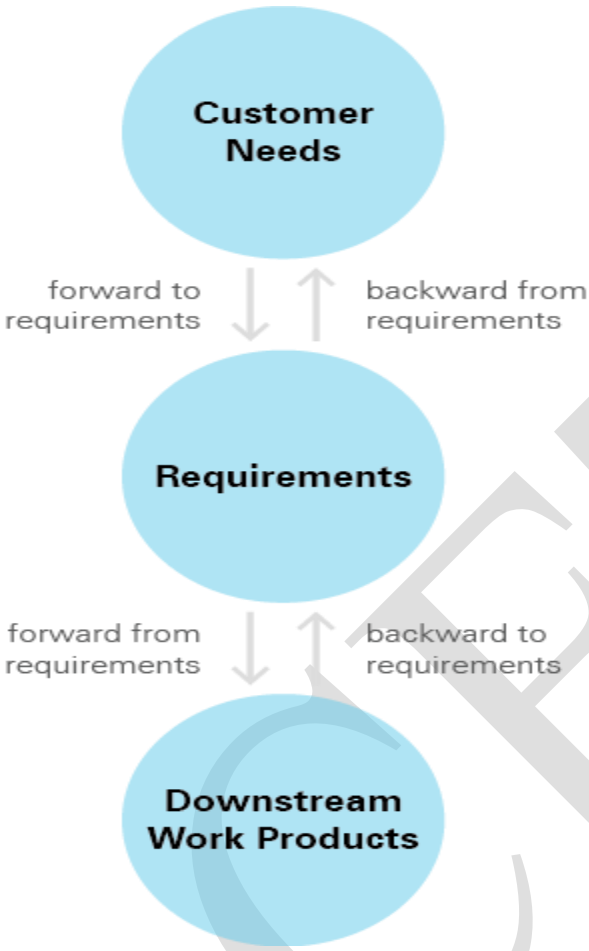
Requirements Traceability Matrix (RTM) Tools

consider these requirements management tools:

- **Visure Requirements:** This tool is provided by Visure Solutions, which is focused on business-critical and safety-critical industries. Its Visure Requirements tool provides complete traceability.
- **Modern Requirements4DevOps**: This tool is integrated with Microsoft's Azure DevOps, TFS, and VSTS, and gives project managers traceability through every stage of the process.
- **ReQtest**: Providing traceability from project start to finish; this tool is based in the cloud. It has a very customizable requirement module that assists project managers in quickly evaluating and tracing changes.

Links in requirements chain Requirements Management Tools:

Derived requirements traceability is a form of requirements management focused on tracing requirements that aren't explicitly defined in higher-level requirements



Four Types of Derived Requirements Traceability

1.Forward to Requirements

When customer needs evolve, requirements may have to be adjusted in response. By making these adjustments, project teams can keep pace with changes in customers priorities, introductions of new business rules, and modifications of existing rules, among other events.

2. Backward From Requirements

Tracking backward from requirements can provide clarity into the origin of each derived requirement. For instance, a requirements management tool could show the link between the derived requirement, the requirement it came from, and the customer use case being addressed.

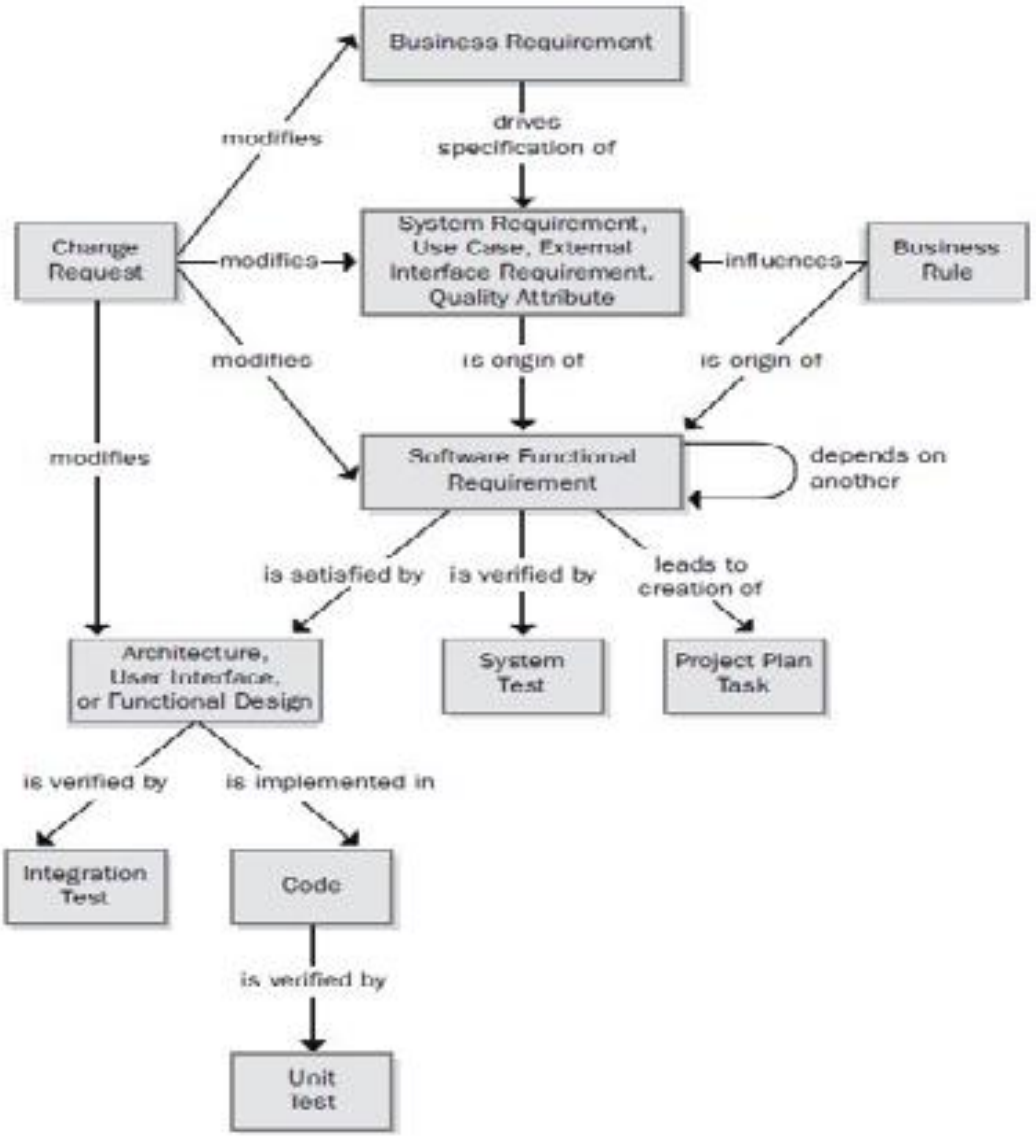
3. Forward From Requirements

Once derived requirements begin flowing into downstream deliverables during product development, it's possible to draw trace relationships between requirements and their corresponding elements. This type of link provides assurance that every requirement is satisfied by a particular component.

4. Backward to Requirements

Finally, this type of link allows for visibility into why certain features were created. Consider how most applications include lines of code that don't directly relate to stakeholder requirements. Even so, it is important to know why a software engineer wrote that code in the first place.

Traceability links create clarity in such situations, shining a light on how the different pieces of a system all fit together. Conversely, test cases derived from – and traced back to – individual requirements offer a mechanism for detecting unimplemented requirements, because the tester won't find the expected functionality.



Benefits of using a requirements management tool

Benefit of implementing a requirements management tool

Minimises defects

A requirements management tool can help you identify the causes of defects, allowing you to plan effective solutions. It can also help you implement procedures that preemptively address potential causes for defects.

Mitigates risk

Some products might involve elements of risk that can require effective management to avoid or mitigate. If the product is a physical product, a requirements management tool can help you identify appropriate safety requirements that protect the product's users.

Improves product delivery time

A requirements management tool can effectively reduce the product's development time. A management tool can help you organise product requirements, avoid delays, reduce defects and identify product dependencies.

Reduces costs

Implementing a requirements management tool usually reduces the overall cost of product development processes. This is because a requirements management tool can reduce delays and shorten the development phase. Businesses usually incur more costs if a product is in the development phase for an extended duration.

Provides traceability

One of the important features that many requirement management tools might offer is the ability to trace errors and defects to an original source. Depending on the software, a requirements management tool can record processes and display project information in clear visualisations. This can help you identify which processes in the product's development are causing defects or issues.

commercial requirements management tool

Commercial requirements management tools are essential for businesses to effectively capture, track, and manage requirements throughout the project lifecycle. These tools offer features like requirement documentation, traceability, collaboration, and version control. Here are a few popular ones:

1. **IBM Engineering Requirements Management DOORS**: This tool is known for its robustness in managing complex requirements. It allows traceability, collaboration, and integrates well with other IBM tools.
2. **Jama Connect**: It provides a collaborative platform for requirements, risk, and test management. It's user-friendly and offers customization options.
3. **Polarion Requirements**: Part of the Siemens PLM Software suite, Polarion offers requirements management along with capabilities for ALM(Application Lifecycle Management).
4. **Helix RM (formerly IBM Rational DOORS Next Generation)**: A modern requirements management tool offering traceability, collaboration, and integration with other tools in the Helix suite.
5. **Visure Requirements**: Known for its flexibility and scalability, Visure Requirements offers features like traceability, impact analysis, and customizable reporting.

Rational Requisite pro:

Rational RequisitePro, previously known as Requisite, was an IBM Rational software tool specifically designed for managing and tracing software project requirements. It was widely used for requirement management in software development projects, particularly in the early stages of the software development lifecycle.

RequisitePro offered a structured approach to capturing, organizing, and managing requirements. Some of its key features included:

1. **Requirements Management**: Capturing and documenting requirements in a structured manner, allowing for easy organization and categorization.
2. **Traceability**: Establishing and managing traceability relationships between different requirements, ensuring that changes to one requirement are reflected and tracked across related requirements.
3. **Collaboration and Communication**: Facilitating collaboration among team members by providing a centralized platform for discussions, comments, and feedback on requirements.
4. **Version Control**: Tracking changes to requirements and managing different versions, allowing teams to revert to previous versions if needed.

Caliber – RM:

Caliber Requirements Management (Caliber-RM) is a software tool developed by Micro Focus. It's designed to facilitate the management of software and system requirements throughout the software development lifecycle. Caliber-RM provides features that aid in capturing, analyzing, and tracing requirements, ensuring a streamlined and organized approach to requirement management.

Some key features of Caliber-RM include:

1. **Requirement Capture**: It allows users to capture and document requirements in a structured format, making it easier to organize and manage them.
2. **Traceability**: Caliber-RM enables the establishment of traceability links between different requirements, ensuring that changes in one requirement are tracked across related requirements.
1. **Collaboration and Communication**: The tool supports collaboration among team members by providing a platform for discussions, comments, and feedback on requirements.
2. **Version Control**: Caliber-RM allows for versioning of requirements, enabling teams to track changes and manage different versions.
3. **Reporting and Analysis**: It offers reporting and analysis capabilities, allowing stakeholders to gain insights into the status and progress of requirements.

Caliber-RM aims to streamline the requirements management process, improve communication among team members, and ensure that the software being developed aligns with the specified requirements.

Implementing requirements management automation

Implementing requirements management automation involves leveraging tools and processes to streamline and optimize the handling of requirements throughout a project's lifecycle. Here's a step-by-step guide to implementing this:

1.Assessment and Planning:

- **Assess Current Practices:** Evaluate existing manual processes for managing requirements.
- **Identify Needs:** Determine the specific needs, challenges, and goals for automation.

2.Selecting the Right Tool:

- **Research Tools:** Explore various requirements management tools available in the market.
- **Consider Requirements:** Ensure the chosen tool aligns with your project's requirements, scalability, integration capabilities, and budget.

3.Implementation Strategy:

- **Define Processes:** Map out how requirements will be captured, documented, reviewed, and managed using the tool.
- **Training and Onboarding:** Provide training to the team members on how to use the
- selected tool effectively.

4.Migration and Data Import:

- **Transfer Existing Data:** If applicable, migrate existing requirements data into the new automated system.

5.Customization and Configuration:

- **Customize Workflows:** Tailor the tool's workflows to match the specific requirements of your organization and project.
- **Configure Settings:** Set up permissions, access controls, and notifications as needed.

6. Integration with Other Tools:

- **Integrate with Existing Systems:** Ensure seamless integration with other tools used in your project management or development environment (e.g., issue tracking, version control

- systems).

7.Testing:

- **Test Functionality:** Verify that the automated system meets the requirements and functions as expected.
- **User Acceptance Testing (UAT):** Involve stakeholders and end-users to validate the system's usability and effectiveness.

8.Rollout and Adoption:

- **Gradual Implementation:** Consider a phased rollout to allow for smooth adoption by teams.
- **Encourage Adoption:** Provide ongoing support and encouragement for team members to embrace the new automated system.

9.Monitoring and Improvement:

- **Performance Metrics:** Establish metrics to measure the effectiveness and efficiency of the automated requirements management system.
- **Feedback and Iteration:** Gather feedback from users and stakeholders to identify areas for improvement and make necessary adjustments.

10.Documentation and Maintenance:

- **Document Processes:** Maintain documentation outlining the automated requirements management processes and procedures.
- **Regular Updates:** Keep the system updated with the latest versions and patches.

UNIT-IV
Software Estimation

Components of Software Estimations:

Software estimation involves predicting the effort, time, cost, and resources required to develop a software product. Estimating accurately is crucial for project planning, resource allocation, and meeting client expectations. Several components are considered when making software estimations:

1. **Size of the Project:** The size of the software, often measured in lines of code, function points, or other metrics, serves as a fundamental aspect of estimation. Larger projects typically require more effort and time to develop.
2. **Complexity:** The complexity of the software, including its architecture, design intricacy, and technical challenges, significantly impacts the estimation. Complex systems generally take more time and effort to develop.
3. **Requirements:** Clear and well-defined requirements are essential for accurate estimation. Changes or uncertainties in requirements throughout the project can affect estimates. Requirements volatility should be considered when estimating.
4. **Experience and Expertise:** The skills and experience of the development team play a crucial role. A more experienced team might be able to handle tasks more efficiently and accurately, impacting the estimation.
5. **Historical Data:** Past project data and historical information can be valuable for estimation. Analyzing similar past projects can provide insights into effort, time, and resources required for the current project.
6. **Risk Assessment:** Identifying and assessing risks associated with the project is important. Factors such as technology uncertainties, dependencies on third-party components, or changes in the market can impact estimates.
7. **Tools and Technology:** The tools, frameworks, and technologies used in the project can influence estimations. Some technologies might speed up development, while others might require additional time for learning or troubleshooting.
8. **Project Management Approach:** The chosen project management methodology (e.g., Agile, Waterfall, etc.) can impact estimations. Agile methodologies, for instance, might require frequent re-estimations due to their adaptive nature.
9. **Communication and Collaboration:** Effective communication among team members and stakeholders helps in understanding requirements and potential challenges, which can affect estimations.

10. **Buffer for Contingencies:** Including a buffer or contingency in estimates to account for unforeseen events, delays, or changes is a common practice to mitigate risks.

Estimation methods:

Estimating software projects accurately is challenging due to the dynamic nature of software development. Often, a combination of expert judgment, historical data analysis, and continuous refinement throughout the project lifecycle helps in improving the accuracy of estimations.

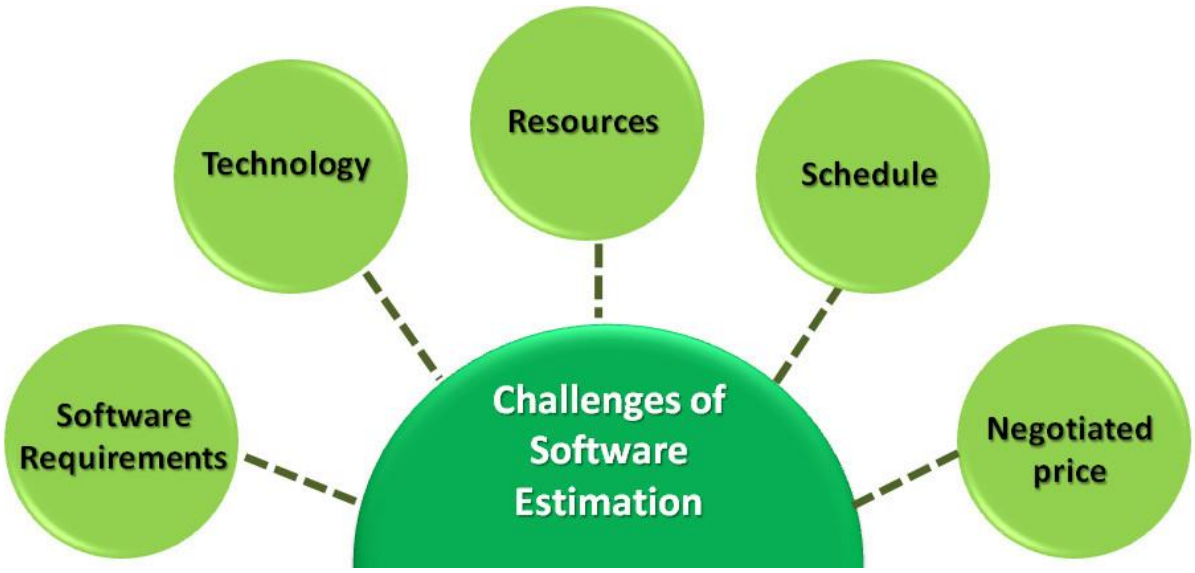
Software estimation methods are techniques used to predict the effort, time, cost, and resources required for software development. Several methods and approaches exist, each with its strengths and weaknesses. Here are some common software estimation methods:

1. **Expert Judgment:** This method relies on the expertise and experience of individuals or a group of experts. It involves consulting with experienced professionals who have worked on similar projects to gather insights and make informed estimates. While subjective, expert judgment is often used in combination with other methods for more accurate estimates.
2. **Algorithmic Estimation Models:** These models use mathematical algorithms based on historical data to predict future project parameters. Examples include COCOMO (Constructive Cost Model) and its variants, which use various formulas and factors to estimate effort, cost, and schedule based on project size and complexity.
3. **Use Case Points:** This method estimates software size based on the number and complexity of use cases. Use case points involve assigning weights to use cases based on complexity factors to estimate effort and resources required.
4. **Function Points:** Function Point Analysis assesses software by quantifying its functionality based on user input, output, inquiries, data files, and interfaces. Function points serve as a measure of software size and can be used to estimate effort and resources.
5. **Comparative Estimation:** This method involves comparing the current project with similar past projects to derive estimates. The key is to identify similarities and differences between projects and adjust estimates accordingly.
6. **Three-Point Estimation:** This technique considers an optimistic, pessimistic, and most likely scenario for each task or requirement and calculates a weighted average or uses a formula (like PERT - Program Evaluation and Review Technique) to derive the final estimate.

7. **Expert Estimation with Delphi Technique:** In this method, experts anonymously provide individual estimates, and these estimates are aggregated and discussed iteratively until a consensus is reached, helping to reduce bias and influence from dominant opinions.
8. **Top-down and Bottom-up Estimation:** Top-down estimation involves deriving estimates for the entire project and then breaking them down into smaller components. Bottom-up estimation starts with individual tasks or components and aggregates them to create an overall estimate.
9. **Parametric Estimation:** This method uses parameters and mathematical models to estimate based on historical data, project characteristics, and other factors. It involves establishing relationships between variables to derive estimates.

Each estimation method has its strengths and weaknesses, and the choice of method often depends on project size, complexity, available data, and the preferences of the development team. Combining multiple estimation methods or refining estimates iteratively as the project progresses often leads to more accurate predictions.

Problems associated with software estimation



Anybody who has been in the field of software development will immediately relate to what I am going to say in this blog. Software Engineering has given us estimation models like

Function point estimation, NESMA and most recent Planning Poker extensively used in agile Development. But still the challenge continues and it starts before the beginning of Software Development Life Cycle..

1. **Software Requirements** - To build a product or customized software, the requirements are never clear. They keep evolving. Even if, they are well-documented in RFP (Request for Proposal) or a Product Description Document, they are never synchronously understood by all the stakeholders. Additionally, because of the new Web or Cloud platform, there are various non-functional requirements like load, multiple browsers, multiple devices, dependency on computer, network speed etc. When you are quoting during software sales, your estimate can go off by more than 50 %.
2. **Technology** – Software Technology is continuously changing and there are patches and versions of each technologies which you need to keep an eye on. Some patches are needed for some operating system, or some patches have fixed a recent vulnerability. There is a new service pack released. OMG (Oh My God !), you feel exhausted keeping pace with turbulence. By the time, you have start development, some new technology or an upgrade is beckoning you and it is tough to decide whether you opt for the current stable version and compromise the new features or try the new version with the risk of being the guinea pig..
3. **Resources** – As a direct consequence of Technology turbulence, Resource productivity gets affected. Resource Productivity is continuously changing and they

need to be trained continuously. Again, you may train them, but they may not be available by the time the project starts- either they have left the company or allotted to the project which has got delayed. Again, resources, being human, their productivity varies due to their moods and personal problems. Exactly at the time of critical software delivery, the critical resource may fall ill or have a fight with his wife or may need to attend a religious/social function.

4. **Schedule** – Customer wants delivery as per his schedule because of his business priorities. This is sometimes not possible due to technical or business process limitations. Certain modules need to be developed before certain others. Some software deliveries need to be sequential and cannot be parallel like you cannot deploy two mothers to get a child delivered in 4 months, instead of 9 months.
5. **Negotiated Price** – It is very difficult to justify the price which you quote for the software. The resources needed to develop the software are human and their efficiency or experience is not directly visible. Sometimes, the software sales have a tough time justifying the cost. To win the contract and hope to recover in future sales, the negotiated price is lower than cost of software development.

Key project factors that influence estimation:

Estimating a software project involves considering numerous factors that can significantly

impact the accuracy of estimations. Key project factors that influence software estimation include:

1. **Project Scope:** The defined boundaries and extent of the project greatly affect estimation. A clear, well-defined scope helps in more accurate estimation, while a vague or changing scope can introduce uncertainties.
2. **Requirements Clarity and Stability:** The clarity, completeness, and stability of project requirements play a crucial role. Ambiguous, volatile, or evolving requirements make estimation challenging and may lead to inaccurate predictions.
3. **Project Complexity:** The complexity of the software solution, including its architecture, functionality, technical intricacies, and integration requirements, affects the estimation process. More complex projects generally require more effort and time.
4. **Project Size:** The size of the project, often measured in terms of lines of code, function points, or other metrics, is a fundamental factor. Larger projects typically require more resources and time for development.
5. **Team Expertise and Experience:** The skills, expertise, and experience of the development team impact estimation. A highly skilled and experienced team might handle tasks more efficiently, potentially affecting estimated effort and timeframes.
6. **Technological Factors:** The choice of technology stack, frameworks, tools, and third-party integrations can influence estimation. Working with unfamiliar or emerging technologies may require additional time for research and implementation.
7. **Dependencies and Constraints:** External dependencies, such as dependencies on third-party components or services, availability of resources, or regulatory constraints, can affect estimation accuracy.
8. **Risks and Uncertainties:** Identifying and assessing project risks is crucial. Uncertainties like market changes, technological uncertainties, or unexpected events, can significantly impact estimations.
9. **Project Management Approach:** The chosen project management methodology, whether Agile, Waterfall, or others, can impact estimation. Different methodologies have varying approaches to planning and estimation.
10. **Communication and Collaboration:** Effective communication among team members, stakeholders, and clients is vital. Lack of communication or collaboration issues can lead to misunderstandings, impacting estimation accuracy.
11. **Quality Standards and Testing Requirements:** High-quality standards and rigorous testing requirements affect the overall development time and effort. Estimations need to account for adequate time for testing and quality assurance activities.

Two views of sizing:

In software size estimation, there are two primary approaches or perspectives often used to

determine and quantify the size of a software project:

1. **Functional Size Measurement:** This approach focuses on measuring the size of software based on its functionalities and user requirements rather than technical aspects. Function Point Analysis (FPA) is a prominent method within this perspective. FPA quantifies the software's functional size by evaluating the number of inputs, outputs, inquiries, logical files, and external interfaces. It aims to measure what the software does from a user's perspective, abstracting away from the underlying technical implementation.
2. **Physical Size Measurement (Lines of Code - LOC):** This approach measures software size based on the actual lines of code written for the software. Unlike the functional view that emphasizes what the software does, this method provides a more direct and technical measure of size, focusing on the volume of code written. However, the number of lines of code can vary significantly based on programming languages, coding practices, and the efficiency of the code.

These two perspectives offer distinct viewpoints on how to measure software size during estimation. The functional approach, exemplified by methods like Function Point Analysis, assesses size based on the software's functionalities, while the physical approach, represented by Lines of Code estimation, evaluates size based on the lines of code written, providing a more implementation-centric view.

Choosing between these approaches or using them in combination depends on various factors, including the nature of the project, the level of abstraction required, and the available information or expertise within the development team. Integrating both perspectives can provide a more comprehensive understanding of the software's size and complexity.

Function Point Analysis was initially developed by Allan J. Albrecht in 1979 at IBM and it has been further modified by the **International Function Point Users Group (IFPUG)**. The initial definition is given by **Allan J. Albrecht.**
Functional Point Analysis gives a dimensionless number defined in function points which we have found to be an effective relative measure of function value delivered to our customer.

Objectives of Functional Point Analysis

- The objective of FPA is to measure the functionality that the user requests and receives.
- The objective of FPA is to measure software development and maintenance
- independently of the technology used for implementation.
- It should be simple enough to minimize the overhead of the measurement process.
- It should be a consistent measure among various projects and organizations.

Types of Functional Point Analysis

There are basically two types of Functional Point Analysis, that are listed below.

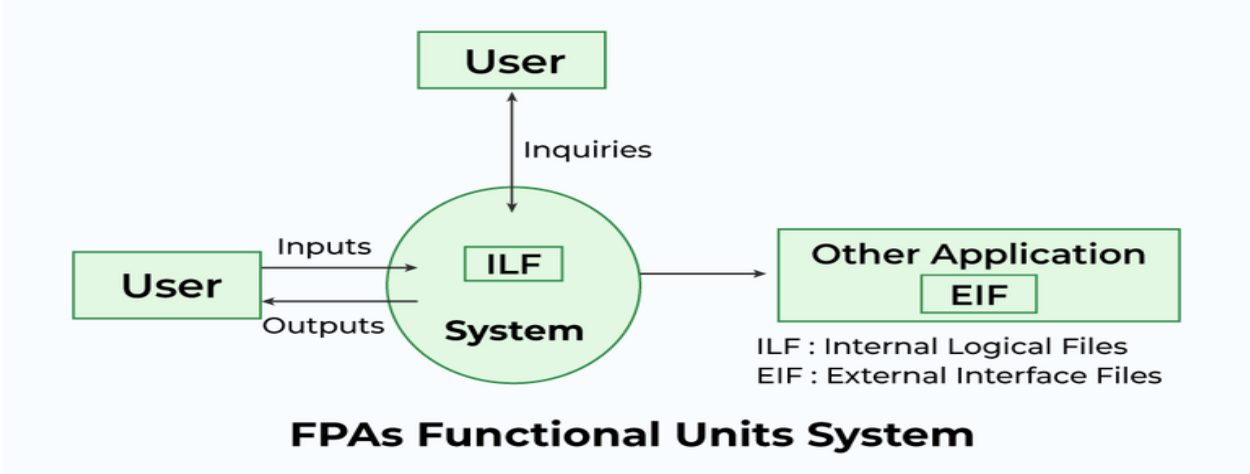
- Transactional Functional Type
- Data Functional Type

Transactional Functional Type

- **External Input (EI):** EI processes data or control information that comes from
- outside the application's boundary. The EI is an elementary process.
- **External Output (EO):** EO is an elementary process that generates data or control information sent outside the application's boundary.
- **External Inquiries (EQ):** EQ is an elementary process made up of an input-output combination that results in data retrieval.

Data Functional Type

- **Internal Logical File (ILF):** A user-identifiable group of logically related data or
- control information maintained within the boundary of the application.
- **External Interface File (EIF):** A group of users recognizable logically related data allusion to the software but maintained within the boundary of another software.



FPAs Functional Units System

Benefits of Functional Point Analysis

- FPA is a tool to determine the size of a purchased application package by counting all the functions included in the package.
- It is a tool to help users discover the benefit of an application package to their

- organization by counting functions that specifically match their requirements.
- It is a tool to measure the units of a software product to support quality and productivity analysis.
- It is a vehicle to estimate the cost and resources required for software development and maintenance.
- It is a normalization factor for software comparison.

Characteristics of Functional Point Analysis

We calculate the functional point with the help of the number of functions and types of functions used in applications. These are classified into five types.

Measurement Parameters	Examples
Number of External Inputs (EI)	Input screen and tables
Number of External Output (EO)	Output screens and reports
Number of external inquiries (EQ)	Prompts and interrupts
Number of internal files (ILF)	Databases and directories
Number of external interfaces (EIF)	Shared databases and shared routines

Functional Point helps in describing system complexity and also shows project timelines. It is majorly used for business systems like information systems.

Weights of 5 Functional Point Attributes

Measurement Parameter	Low	Average	High
Number of external inputs (EI)	3	4	6
Number of external outputs (EO)	4	5	7
Number of external inquiries (EQ)	3	4	6
Number of internal files (ILF)	7	10	15
Number of External Interfaces (EIF)	5	7	10

Functional Complexities help us in finding the corresponding weights, which results in finding the Unadjusted Functional point (UFp) of the Subsystem. Consider the complexity as average for all cases. Below mentioned is the way how to compute FP.

			Weighing Factor		
Measurement Parameter	**Count**	Total_Count	**Simple**	**Average**	**Complex**
Number of external inputs (EI)	32	32*4=128	3	4	6
Number of external outputs (EO)	60	60*5=300	4	5	7
Number of external inquiries (EQ)	24	24*4=96	3	4	6
Number of internal files (ILF)	8	8*10=80	7	10	15
Number of external interfaces (EIF)	2	2*7=14	5	7	10
Algorithms used Count total →		618			

From the above tables, Functional Point is calculated with the following formula

*FP = Count-Total * [0.65 + 0.01 * ∑**(fi)**]*

*= Count * CAF*

Here, the **count-total** is taken from the chart.

*CAF = [0.65 + 0.01 * ∑**(fi)**]*

- ∑**(fi) =** sum of all 14 questions and it also shows the complexity factor – CAF.
- CAF varies from 0.65 to 1.35 and ∑**(fi)** ranges from 0 to 70.
- When ∑**(fi)** = 0, CAF = 0.65 and when ∑**(fi)** = 70, CAF = 0.65 + (0.01*70) = 0.65 + 0.7 = 1.35

Questions on Functional Point

1. **Consider a software project with the following information domain characteristic for the calculation of function point metric.**

Number of external inputs (I) = 30

Number of external output (O) = 60

Number of external inquiries (E) = 23

Number of files (F) = 08
Number of external interfaces (N) = 02

Mark IIFPA:

The Mark II Function Point Analysis (Mark IIFPA) is an enhanced or refined version of the traditional Function Point Analysis (FPA) method used in software size estimation. FPA, developed by Allan Albrecht in the 1970s, is a well-known technique for quantifying the functional size of software based on user interactions and functionalities.

Mark II Function Point Analysis was introduced as an improvement upon the original FPA method, aiming to address certain limitations and enhance the accuracy and applicability of function point counting.

Some of the key features or improvements in Mark II Function Point Analysis might include:

1. **Refined Counting Rules:** Mark II FPA might have updated or refined counting rules compared to the original FPA method, which could lead to more precise and consistent function point counts.
2. **Adaptability:** It could be more adaptable to modern software development practices, taking into account changes and advancements in technology, architecture, and software functionalities.
3. **Enhanced Guidelines:** Mark II FPA might offer more detailed guidelines or methodologies for counting function points, providing clearer instructions on how to evaluate and measure different types of functionalities.
4. **Increased Accuracy:** The improvements in Mark II FPA aim to enhance accuracy in estimating software size, which is crucial for project planning, resource allocation, and decision-making.

However, specific details and nuances of the Mark II Function Point Analysis might vary based on the version or adaptation of the methodology used by different practitioners or organizations. It's essential to refer to authoritative sources or documentation specific to Mark II FPA for a more comprehensive understanding of its features, guidelines, and

applications in software size estimation.

Full Function Points:

The Full Function Point (FFP) method in software estimation is an extended and more comprehensive version of the traditional Function Point Analysis (FPA) technique. FFP aims to provide a more detailed and nuanced measurement of the functional size of software by including additional categories or aspects that might not be covered by the standard FPA.

Here are the key components and characteristics of the Full Function Point method:

1. **Extension of Function Point Analysis (FPA):** FFP builds upon the foundation of FPA, which quantifies software functions based on five primary categories: External Inputs (EIs), External Outputs (EOs), External Inquiries (EQs), Internal Logical Files (ILFs), and External Interface Files (EIFs).
2. **Additional Categories or Aspects:** FFP extends the scope of function point counting by incorporating supplementary categories or aspects. These additional aspects could encompass more complex functionalities, modern technology components, transaction types, elements of data communications, distributed architectures, or other elements that impact the software's functional size.
3. **Comprehensive Measurement:** The purpose of Full Function Points is to offer a more comprehensive measurement of the software's functional size. By encompassing a broader range of functionalities and interactions, FFP aims to capture a more detailed view of the software's complexity, intricacies, and scope.
4. **Enhanced Accuracy and Detail:** The inclusion of additional categories or aspects in FFP allows for a more detailed assessment of the software's functional size. This increased granularity enhances the accuracy of estimation, aiding in better project planning, resource allocation, and decision-making.
5. **Detailed Analysis and Counting Rules:** Similar to FPA, Full Function Points require a thorough analysis of the software's functionalities and their complexity. Counting rules and guidelines specific to FFP need to be followed, ensuring consistent and standardized measurement across different software projects.
6. **Improved Project Management:** The comprehensive assessment provided by Full

Function Points supports project managers and stakeholders in understanding the software's size, complexity, and functional intricacies. This information is valuable for estimating project timelines, resource requirements, and overall project scope.

7. **Application Flexibility:** The specific categories or aspects included in Full Function Points might vary based on the needs of the project, advancements in technology, industry standards, or organization-specific considerations.

LOC Estimation:

A **line of code (LOC)** is any line of text in a code that is not a comment or blank line, and also header lines, in any case of the number of statements or fragments of statements on the line. LOC clearly consists of all lines containing the declaration of any variable, and executable and non-executable statements. As Lines of Code (LOC) only counts the volume of code, you can only use it to compare or estimate projects that use the same language and are coded using the same coding standards.

Features :

- Variations such as "source lines of code", are used to set out a codebase.
- LOC is frequently used in some kinds of arguments.
- They are used in assessing a project's performance or efficiency.

Advantages :

- Most used metric in cost estimation.
- Its alternates have many problems as compared to this metric.
- It is very easy in estimating the efforts.

Disadvantages :

- Very difficult to estimate the LOC of the final program from the problem specification
- It correlates poorly with quality and efficiency of code.
- It doesn't consider complexity.

Research has shown a rough correlation between LOC and the overall cost and length of developing a project/ product in Software Development, and between LOC and the number of defects. This means the lower your LOC measurement is, the better off you probably are in the development of your product.

Let's take an example and check how does the Line of code work in the simple sorting program
given below:

C++

```
void selSort(int x[], int n) {
  //Below function sorts an array in ascending
```

```
order
    int i, j, min, temp;
    for (i = 0; i < n - 1; i++) {
        min = i;
        for (j = i + 1; j < n; j++)
        if (x[j] < x[min])
        min = j;
        temp = x[i];
        x[i] = x[min];
        x[min] = temp;
    }
}
```

So, now If LOC is simply a count of the number of lines then the above function shown contains **13 lines of code (LOC).** But when comments and blank lines are ignored, the function shown above contains **12 lines of code (LOC)**.

Let's take another example and check how does the Line of code work the given below:

C++

```
void main()
 {
    int fN, sN, tN;
    cout << "Enter the 2 integers: ";
    cin >> fN >> sN;
    // sum of two numbers in stored in variable sum
    sum = fN + sN;
    // Prints sum
    cout << fN << " + " <<  sN << " = " << sum;
    return 0;
}
```

Here also, If LOC is simply a count of the numbers of lines then the above function shown contains 11 lines of code (LOC). But when comments and blank lines are ignored, the function shown above contains 9 lines of code (LOC).

Conversion between Size measures:

Effort and Schedule

Sizing the project by using function points, SLOC, or other methods is a job only half done. Transforming the size to a deliverable effort within a comfortable schedule makes the

project planning a complete success story. Further, the total project effort (for example, in person months) that needs to be consumed in a given schedule provides the guidance to do a proper resource loading.

Once the phase-wise resource loading details are available, you can apply the resource rate to each category of resource—such as project manager, architect, analyst, and developer—for the duration of the assignment. Thus the total base cost for the project is calculated. You can then add project management, configuration management, and other overheads as appropriate to get the gross cost. Figure 7.3 shows the broad parameters that are to be taken into account during different lifecycle stages of the project execution.

Deriving Effort

The overall project effort (typically measured in person months) is directly dependent on two critical inputs: application size and project team/programmer productivity. The steps to calculate each of these items are as follows:

- From the given specification for the application, calculate the size of the application.
- The size can be estimated by using one of the popular estimation methods, such as
    - *Function points method:* Output will be in FP count.
    - *Object points method:* Output will be a list of classes of simple/medium/
    - complex categories.
    - *SLOC method:* Output will be a "gut feel" of lines of code.
- Make sure that you have the productivity (delivery rate) available for the technology platform on which the application is being developed. For every language there are available average productivity figures that should be adjusted by the historic project productivity data for your own IT organization. Productivity of your project team:
    - Is based on competency of programmers
    - Is specific to a given technology
    - Is dependent on the software development environment
- Convert application size to effort (in person months):
    - Effort = Application size x productivity
- The effort thus derived is the total project effort that would be spent for all the lifecycle stages of the project, from requirements creation through user acceptance. Add project

management and configuration management effort as applicable. The effort is also the aggregate of the individual effort spent by each of the resources assigned to the project.

Scheduling

Transforming the overall project effort into a delivery schedule (elapsed time) is somewhat tricky. If the right approach is not applied, the risks of project failure are high. There are three alternatives to calculate the schedule:

- Use popular scheduling methods like COCOMO II.
- "Gut feel" scheduling based on past experience.
- Schedule driven by business user need.

The schedule data that can be obtained by one of these methods is in the form of duration required to deliver the project itself. For example, the schedule could span 10 months from the start date of the project. The schedule thus encompasses all the lifecycle stages of the entire project. From the total duration given to the project team, the project manager must divide the time into lifecycle-based segments. The lifecycle phase percentage is also to be based on historical delivery information of the IT organization. For example, with 10 months of elapsed time, the schedule can be split as follows:

- *Requirements:* 2 months (20 percent)
- *Detailed design:* 1.5 months (15 percent)
- *Build and unit test:* 4 months (40 percent)
- *System and integration test:* 2.5 months (25 percent)

Resource Loading

Resource loading is a complex activity and has to be worked on with extreme care. Improper assignment of resources will have an impact on project delivery schedules as well as the quality of outputs. Resource loading requires two critical mapping considerations:

- The right resource role for the appropriate lifecycle stage. For example, you need to
- know when to assign a project manager, an architect, or a programmer.
- The right duration of assignment. This includes when to assign and when to release.
- The effort spent by each resource is determined by tactful resource allocation method.

For Figure 7.3 shown earlier in this section, the resource loading patterns are displayed illustratively in Table 7.2. For your project, you can prepare a table showing resource role assignments for the appropriate durations. For example, assume a total project effort of 100 person months. This effort includes project management and configuration management effort. Table 7.2 illustrates the typical resource loading based on the percentage breakup of elapsed time, as given in the example in this chapter.

Table 7.2. Resource Loading Chart

Resource	***M1***	***M2***	***M3***	***M4***	***M5***	***M6***	***M7***	***M8***	***M9***	***M10***	***Total PM***
Project manager	1	1	1	1	1	1	1	1	1	1	10
Technical analyst		1	1	1					1	1	5
Business analyst	2	3	3	3	3	3	3	3	2	2	27
Programmer			4	4	6	8	10	8	6	4	50
Configuration controller			1	1	1	1	1	1	1	1	8
Total effort	3	5	10	10	11	13	15	14	11	8	100

[M1 = Month 1] [Total PM = Total Person Months].

Costing

Once the resource loading chart (as shown in Table 7.2) is complete, it is fairly easy to attach the rate per hour (or rate per week/month) for each of the resource roles, such as project manager, architect, analyst, developer, etc. The steps are

- Arrive at the rate per time unit for each of the resources.
- From the resource loading chart, obtain the duration of assignment for each category

of resource (project manager, architect, analyst, developer).

- Multiply the individual resource allocation duration by the rate to obtain individual
- resource costs.
- Aggregate the individual costs to get the overall project cost for resources.
- Add overheads and buffers as applicable.

UNIT-V

What is Productivity:

Software productivity tools help developers **create and manage software projects more effectively**. These tools can help developers track the progress of their projects, manage software dependencies, and automate software builds. Some software productivity tools are specific to a particular programming language, while others are more general. Many software productivity tools are available as open source software, and many are also available as commercial software.

There are various software productivity tools available for compiler design. Some of the most popular ones are:

1. **Code generation tools:** These tools help in automatically generating code for the compiler. This can be extremely helpful in reducing the development time and effort required for the compiler.
2. **Debugging tools:** These tools help in debugging the compiler code. This can be
3. extremely helpful in finding and correcting errors in the compiler code.
4. **Performance analysis tools:** These tools help in analyzing the performance of the compiler. This can help optimize the compiler code for better performance.

Estimation Factors:

The necessity of cost estimation stems from the requirements of scheduling and cost planning. For lack of more precise methods, cost estimation for software development is almost always based on a comparison of the current project with previous ones. Due to the uniqueness of software systems, the number of comparable projects is usually quite small, and empirical data are seldom available. But even if there are exact cost records of comparable projects, these data are based on the technical and organizational conditions under which the comparable project was carried out at that time

The technical and organizational conditions are variable parameters, which makes empirical data

from comparable projects only an unreliable basis for estimates. Relationship between the best and worst programming experience (referring to the same task, [Schnupp 1976]):

The time requirement for each task handled in a team consists of two basic components ([Brooks 1975]): (1) Productive work

(2) Communication and mutual agreement of team members

If no communication were necessary among team members, then the time requirement t for a project would decline with the number n of team members

$t \approx 1/n$

If each team member must exchange information with one other and that the average time for such communication is k, then the development time follows the formula:

$t \approx 1/n + k. n2/2$ "

Adding manpower to a late software project makes it later." ([Brooks 1975])

Most empirical values for cost estimation are in-house and unpublished. The literature gives few specifications on empirical data, and these often deviate pronouncedly. The values also depend greatly on the techniques and tools used.

Distribution of the time invested in the individual phases of software development (including the documentation effort by share) according to the selected approach model and implementation technique ([Pomberger 1996]):

Approach model: classical sequential software life cycle

Implementation technique: module-oriented problem analysis and system specification....

25% design.....

25% implementation.....

15% testing..... 35%

Approach model: prototyping-oriented software life cycle

Implementation technique: module-oriented

problem analysis and system specification..... 40%

design..... 25% implementation.....
10% testing..... 25%

Approach model: object- and prototyping-oriented software life cycle Implementation technique: object-oriented

problem analysis and system specification..... 45%

design..... 20%implementation.....

18% testing……27%.....

The following options are useful to achieve reliable cost and effort estimates:

Delay estimation until late in the project. The longer we wait, the less likely we are to make errors in our estimates. However this is not practical. Cost estimates must be provided "up-front". 2. Base estimates on similar projects that have already been completed. This works well if the current project is quite similar to past efforts. Unfortunately, past experience has not always been a good indicator of future results.

2. Use "decomposition techniques" to generate project cost and effort estimates. These techniques use a "divide and conquer" approach to estimation. By decomposing a project into major functions and related software engineering activities, cost and effort estimation can be performed in a step-wise fashion.

Use one or more empirical models for software cost and effort estimation. A model is

based on experience (historical data) and takes the form d = f(vi), where d is one of a number of estimated variables (eg. effort, cost, project duration) and vi are selected independent parameters (eg. Estimated LOC or FP)

Approaches to Effort and Schedule Estimation:

The cost and schedule estimation process helps in determining number of resources to complete all project activities. It generally involves approximation and development of costing alternatives to plan, perform or work, deliver, or give project. A good estimation is very much essential for keeping a project under budget.

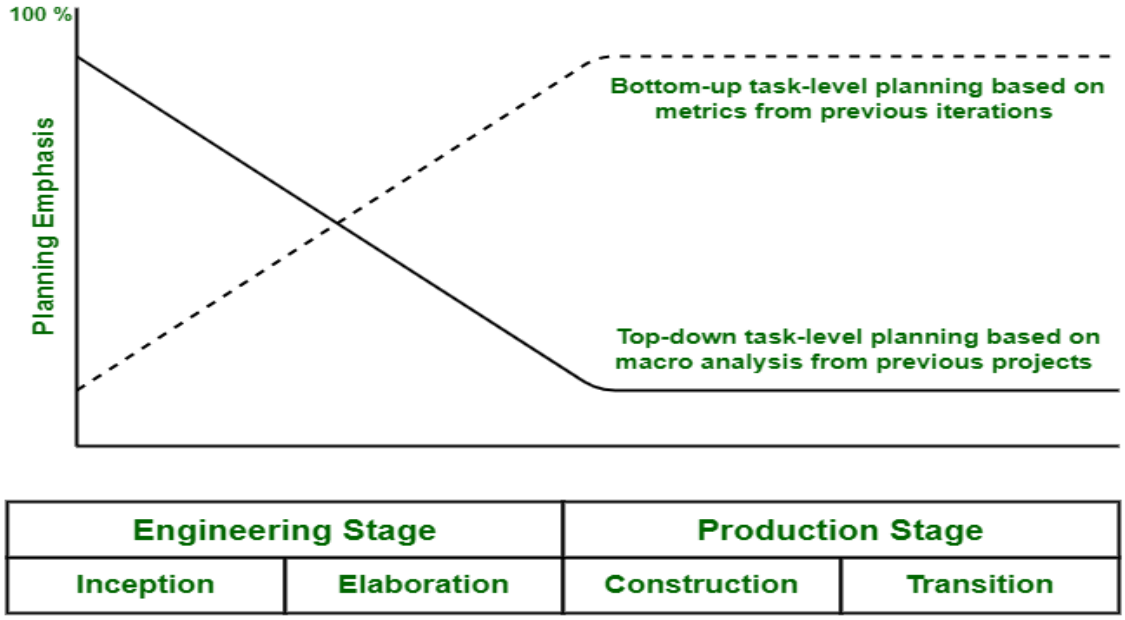
Two perspectives are generally required to derive project plans. These perspectives are given below :

1. **Forward-Looking :**
    - The Forward-Looking approach is also known as Top-Down approach. This approach generally starts with describing and explaining various project
    - tasks that involve starting with project aim or end deliverable and breaking it all down into smaller planning chunks.
    - Top-down budgeting also refers to a method of budgeting where project managers prepare a high-level budget for organization.
    - These project managers or senior management develops and creates a characterization of overall size, process, environment, people, and quality that is essential for software project. In this approach, duration of deliverable's is estimated.
    - It generally takes less time and effort than bottom-up estimate. With help of software cost estimation model, an estimation of overall effort and schedule
    - is done. The project manager generally divides estimation of overall effort into a top-level of WBS (Work Breakdown Structure).
    - They also divide schedule into major milestones dates. At this stage, sub-project managers are simply given responsibility for decomposing every element of WBS into lower levels with help of various allocations of top-level, staffing profile, and, major milestones dates as constraints.
    - The main benefit of this approach is use of holistic data from earlier projects or products, along with unmitigated risks, and scope creeps. This also helps in reducing risk of overlooked work activities or costs.
2. **Backward-Looking :**
    - Backward-Looking approach is also known as Bottom-up approach.

- In this approach, project team breaks requirements of clients down,
- determining lowest level appropriate to develop a range of estimates,
- covering overall scope of project based on available definition of task.
- Overall elements of lowest level WBS are generally explained into detailed tasks, for which WBS element manager is responsible for estimating budge and schedule.
- All of these estimates are joined and integrated into higher-level WBS budgets and milestones.

Milestone scheduling also called budget allocation with help of top-down approach results in a highly optimistic plan. Whereas, bottom-up approach results in a highly pessimistic plan. Iteration is very much needed and important, using results of one approach to validate and even check results of other approach. Both of approaches should be used together, in balance, throughout life-cycle of project as shown below.

Below is diagram showing planning balance through life cycle.



Engineering stage planning emphasis on following points :

- Macro-level task estimation for engineering artifacts.
- Macro-level task estimation for production stage artifacts.

- Stakeholder concurrence.
- Coarse-grained variance analysis of actual vs. planned expenditures.
- Tuning top-down project-independent planning guidelines into project-specific planning guidelines.
- WBS definition and elaboration.

Production stage planning emphasis on following points :

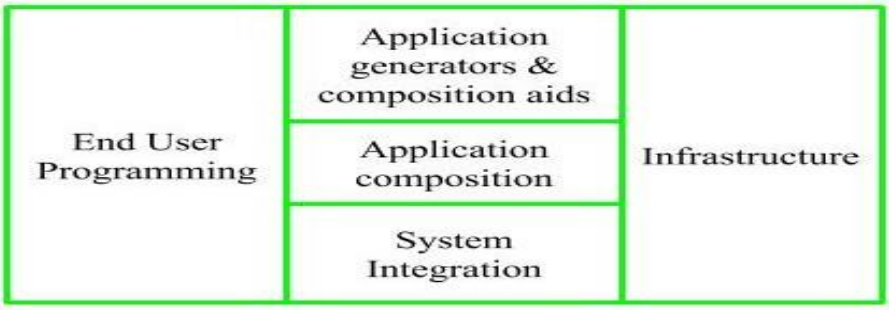
- Macro-level task estimation for production stage artifacts.
- Macro-level task estimation for maintenance of engineering artifacts.
- Stakeholder concurrence.
- Coarse-grained variance analysis of actual vs. planned expenditures.

Top-down perspective generally dominates during engineering stage. This is because there is no enough depth or details of understanding not even stability in sequences of detailed task to perform planning of bottom-up approach. On other hand, there is enough prior experience and planning fidelity that bottom-up planning perspective dominates during production stage.

COCOMO II

COCOMO-II is the revised version of the original Cocomo (Constructive Cost Model) and was developed at the University of Southern California. It is the model that allows one to estimate the cost, effort, and schedule when planning a new software development activity.

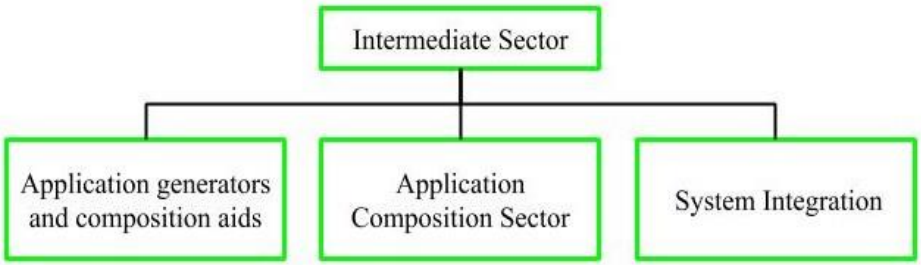
Sub-Models of COCOMO Model



End User Programming

Application generators are used in this sub-model. End user write the code by using these application generators. For Example, Spreadsheets, report generator, etc.

2. Intermediate Sector

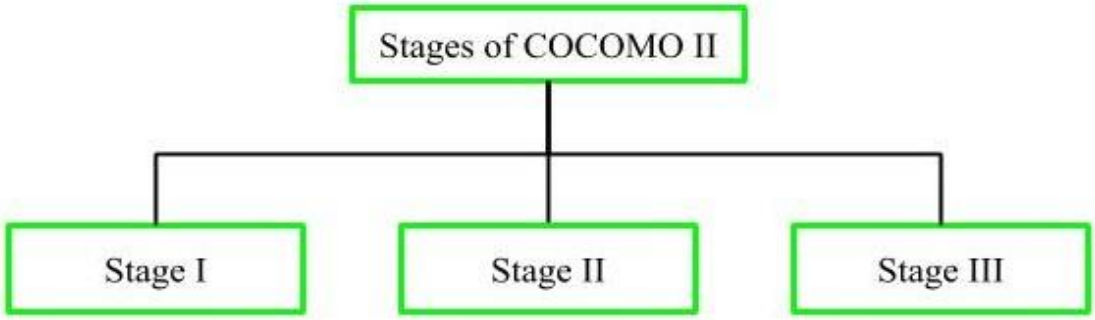


- **Application Generators and Composition Aids:** This category will create largely prepackaged capabilities for user programming. Their product will have many
- reusable components. Typical firms operating in this sector are Microsoft, Lotus,
- Oracle, IBM, Borland, Novell.
- **Application Composition Sector:** This category is too diversified and to be handled by prepackaged solutions. It includes GUI, Databases, domain specific components such as financial, medical or industrial process control packages.
- **System Integration:** This category deals with large scale and highly embedded systems.

3. Infrastructure Sector

This category provides infrastructure for the software development like Operating System, Database Management System, User Interface Management System, Networking System, etc.

Stages of COCOMO II



1. Stage-I

It supports estimation of prototyping. For this it uses Application Composition Estimation Model

This model is used for the prototyping stage of application generator and system integration.

2. Stage-II

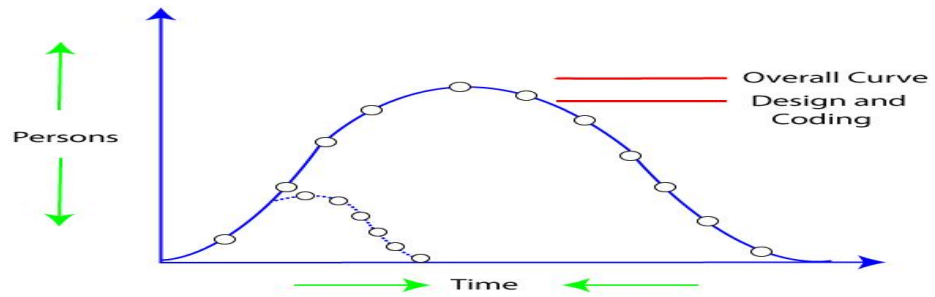
It supports estimation in the early design stage of the project, when we less know about it. For this it uses Early Design Estimation Model. This model is used in early design stage of application generators, infrastructure, system integration.

3. Stage-III

It supports estimation in the post architecture stage of a project. For this it uses Post Architecture Estimation Model. This model is used after the completion of the detailed architecture of application generator, infrastructure, system integration.

Putnam Estimation Model:

he Lawrence Putnam model describes the time and effort requires finishing a software project of a specified size. Putnam makes a use of a so-called The Norden/Rayleigh Curve to estimate project effort, schedule & defect rate as shown in fig:



The Rayleigh manpower loading Curve

Putnam noticed that software staffing profiles followed the well known Rayleigh distribution. Putnam used his observation about productivity levels to derive the software equation:

$$L = C_k K^{1/3} t_d^{4/3}$$

The various terms of this expression are as follows:

K is the total effort expended (in PM) in product development, and L is the product estimate in **KLOC** .

t_d correlate to the time of system and integration testing. Therefore, t_d can be relatively considered as the time required for developing the product.

C_k Is the state of technology constant and reflects requirements that impede the development of the program.

Typical values of C_k = 2 for poor development environment

C_k= 8 for good software development environment

C_k = 11 for an excellent environment (in addition to following software engineering principles, automated tools and techniques are used).

The exact value of C_k for a specific task can be computed from the historical data of the organization developing it.

Putnam proposed that optimal staff develop on a project should follow the Rayleigh curve. Only a small number of engineers are required at the beginning of a plan to carry out planning and specification tasks. As the project progresses and more detailed work are necessary, the number of engineers reaches a peak. After implementation and unit testing, the number of project staff falls.

Effect of a Schedule change on Cost

Putnam derived the following expression:

$$L = C_k K^{1/3} t_d^{4/3}$$

Where, **K** is the total effort expended (in PM) in the product development

L is the product size in KLOC

t_d corresponds to the time of system and integration testing

C_k Is the state of technology constant and reflects constraints that impede the progress of the Program Now by using the above expression, it is obtained that,

$$K = L^3 / C_k^3 t_d^4$$

Or $$K = C / t_d^4$$

Or $\frac{K_1}{K_2} = t_{d2}^4 / t_{d1}^4$

Or $K \propto 1/t_d^4$

Or, cost $\propto 1/t_d$

(As project development effort is equally proportional to project development cost)

From the above expression, it can be easily observed that when the schedule of a project is compressed, the required development effort as well as project development cost increases in proportion to the fourth power of the degree of compression. It means that a relatively small compression in delivery schedule can result in a substantial penalty of human effort as well as development cost.

Algorithmic models:

Algorithmic modeling is a powerful computational design methodology that allows designers to create complex geometries and shapes using mathematical algorithms. With the growing demand for optimization and customization across industries, algorithmic modeling has become an essential tool for architects, engineers, product designers, and digital artists. This article provides an academic overview of algorithmic modeling, including its concepts, historical overview, benefits, and applications.

Algorithmic Modeling Concepts

Algorithmic modeling is based on the use of mathematical algorithms to create complex geometries and shapes. The process includes defining a set of parameters that are used to create a design. These parameters can be adjusted and optimized to create different design

variations. Algorithmic models are created using parametric modeling software. The software allows designers to create and modify algorithmic models in a visual programming environment

Benefits of Algorithmic Modeling

Algorithmic modeling offers several advantages over traditional design methods, including

Flexibility: Algorithmic models can be customized and optimized according to specific project parameters, making it easy to change and adapt projects.

Efficiency: Algorithmic modeling enables faster and more efficient design iterations, reducing the time required for manual adjustments and iterations.

Complexity: Algorithmic modeling allows designers to create complex shapes and figures that would be difficult or impossible to create by hand.

Accuracy. Algorithmic models provide a high degree of precision and accuracy, which is very important in industries such as engineering and manufacturing.

Applications of algorithmic modeling

Architecture. Architects use algorithmic modeling to create complex building designs and explore different design options.

Engineering: Engineers use algorithmic modeling to optimize product performance and improve manufacturing efficiency.

Product Design: Algorithmic modeling is used in product design to create complex shapes and optimize product performance.

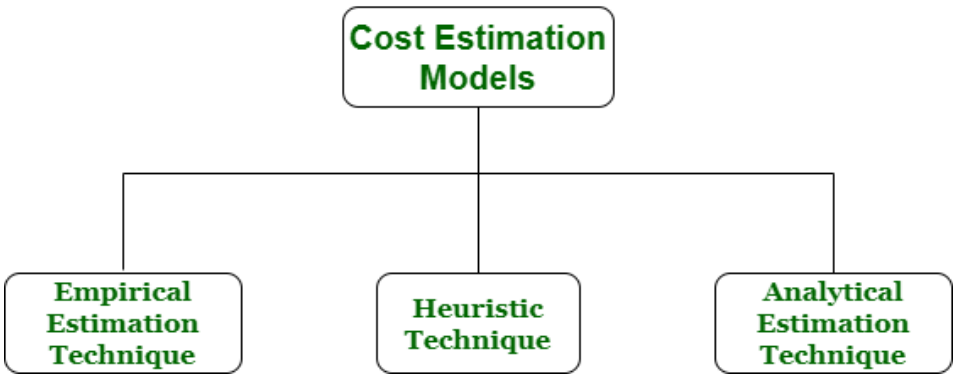
Digital art and animation. Algorithmic modeling is used in digital art and animation to

create complex visual effects and realistic 3D models.

Cost estimation

simply means a technique that is used to find out the cost estimates. The cost estimate is the financial spend that is done on the efforts to develop and test software in Software Engineering

Cost estimation models are some mathematical algorithms or parametric equations that are used to estimate the cost of a product or a project. Various techniques or models are available for cost estimation, also known as Cost Estimation Models as shown below :



1. **Empirical Estimation Technique –** Empirical estimation is a technique or model in which empirically derived formulas are used for predicting the data that are a required and essential part of the software project planning step. These techniques are usually based on the data that is collected previously from a project and also based on some guesses, prior experience with the development of similar types of projects, and assumptions. It uses the size of the software to estimate the effort. In this technique, an educated guess of project parameters is made. Hence, these models are based on common sense. However, as there are many activities involved in empirical estimation techniques, this technique is formalized. For example Delphi technique and Expert Judgement technique.
2. **Heuristic Technique –** Heuristic word is derived from a Greek word that means "to discover". The heuristic technique is a technique or model that is used for solving problems, learning, or discovery in the practical methods which are used for achieving immediate goals. These techniques are flexible and simple for taking quick decisions through shortcuts and good enough calculations, most probably when working with complex data. But the decisions that are made using this technique are necessary to be optimal. In this technique, the relationship among different project parameters is expressed using mathematical equations. The popular heuristic technique is given by Constructive Cost Model (COCOMO). This technique is also used to increase or speed up the analysis and investment decisions.

3. **Analytical Estimation Technique –** Analytical estimation is a type of technique that is used to measure work. In this technique, firstly the task is divided or broken down into its basic component operations or elements for analyzing. Second, if the standard time is available from some other source, then these sources are applied to each element or component of work. Third, if there is no such time available, then the work is estimated based on the experience of the work. In this technique, results are derived by making certain basic assumptions about the project. Hence, the analytical estimation technique has some scientific basis. Halstead's software science is based on an analytical estimation model.

Other Cost Estimation Models are:

1. **Function Point Analysis (FPA):** This technique counts the number and complexity of functions that a piece of software can perform to determine how functional and sophisticated it is. The effort needed for development, testing and maintenance can be estimated using this model.
2. **Putnam Model:** This model is a parametric estimation model that estimates effort, time and faults by taking into account the size of the the programme, the expertise of the development team and other project-specific characteristics.
3. **Price-to-Win Estimation:** Often utilized in competitive bidding, this model is concerned with projecting the expenses associated with developing a particular software project in order to secure a contract. It involves looking at market dynamics and competitors.
4. **Models Based on Machine Learning:** Custom cost estimating models can be built using machine learning techniques including neural networks, regression analysis and decision trees. These models are based on past project data. These models are flexible enough to adjust to changing data and project-specific features.
5. **Function Points Model (IFPUG):** A standardized technique for gauging the functionality of software using function points is offered by the International Function Point Users Group (IFPUG). It is employed to calculate the effort required for software development and maintenance.

Desirable features in software estimation tools:

The decomposition technique and empirical estimation model are available as part of a range of software tools. Such automated estimation tools are helpful in estimating cast and effort and conducting "what-if" analysis for important project variables, such as delivery data or staffing. All automated estimation tools display the same general characteristics, and all perform the following generic functions-

Sizing of Project Deliverable : Estimated the size of one or more work products i.e., external representation of software, software itself, distributed functionality, descriptive information, all are approximate first.

Selecting Project Activities : The required process framework is selected and the software engineering project is specified.

Predicting Staffing Levels : The number of people available is specified. This is an important task, because the relationship between the people available and work is highly inauspicious.

Predicting Software Effort : The estimation tool related to the use of some models from the size of project deliverable to the effort required (from producing them).

Predicting Software Cost : Software costs can be calculated by assigning labor rates to project activities.

Predicting Software Schedules : Having knowledge of effort, staffing level and project activities, a draft schedule can be produced by allocating lober in software engineering activities based on the recommended model for effort distribution.

Here are the few automation estimation tools:

1. **Time monitoring tools:** Programmes such as Harvest or Toggl assist keep track of how much time is spent on activities, but they also offer insights into previous information, which helps make future estimations more accurate.
2. **Tools for Test Automation:** Tools such as Selenium or Appium automate the testing process during the testing phase.
3. **Tools for Continuous Integration/Continuous Deployment (CI/CD):** These tools facilitate a more efficient and error-free release process while also accelerating development.
4. **Planning and Estimation Tools:** Together estimating the amount of work needed for projects or user stories is made easier by tools like Planning Poker.
5. **Requirements Management Tools:** Software such as IBM Engineering connects the process of gathering, monitoring and maintaining project requirements is automated with requirements management systems.
6. **Machine Learning-Based Estimation Tools:** Based on previous information, team performance and other project criteria, these tools use machine learning algorithms to generate more precise and based on fact estimations.
7. **Tools for Resource Management:** Applications such as Resource Guru facilitate effective team resource scheduling and management.
8. **Code Review Tools:** By evaluating code for quality, security and maintainability, tool such as Code Climate can automate certain steps in the code review process.

The International Function Point Users Group (IFPUG)

Critical to the ongoing success of function points as a viable software metric is the work that is being accomplished by the International Function Point Users Group (IFPUG). Since 1986, IFPUG has continued to grow in members and in its importance to the software measurement community. Today IFPUG enjoys a membership of thousands of individual, corporate, educational, and institutional members from more than 30 countries.

IFPUG is a not-for-profit, member-run user group. IFPUG's mission is to be a recognized leader in promoting and encouraging the effective management of application software development and maintenance activities through the use of function point analysis and other software-measurement techniques. For more information, contact IFPUG at www.ifpug.org/, email ifpug@ifpug.org, phone (USA) 609-799-4900, or fax 609-799-7032. IFPUG serves to facilitate the exchange of knowledge and ideas for improved software-measurement techniques and seeks to provide a composite environment that stimulates the personal and professional development of its members. IFPUG typically meets twice a year, once in the spring and then again in the fall. The spring conference is dedicated to training programs and committee meetings. The fall program is devoted to both committee work and training programs, but it also typically includes a two-and-a-half-day user conference and vendor showcase.

Committee work is the core of IFPUG. The two most visible committees are the Counting Practices Committee and the Certification Committee. The Counting Practices Committee is responsible for maintaining the current counting guidelines, *Counting Practices Manual*. The Certification Committee is responsible for establishing and enforcing the certification guidelines. Other critical committees that generate guidelines for either counting or using function points include New Environments, IT Performance and Management Reporting. IFPUG also has an Academic Affairs Committee, a Communications and Marketing Committee, a Conference Committee, and an Education Committee to assist the members with academic studies, information dissemination, conferences, and workshops.

USC's cocomo II:

he USC COCOMO II model, also known as COCOMO 2000, is an advancement of the COCOMO (Constructive Cost Model) developed by Barry Boehm in the 1980s. COCOMO II was developed at the University of Southern California's Center for Systems and Software Engineering as an enhancement and extension of the original COCOMO model. COCOMO II, or COCOMO 2000, aims to provide a more detailed and comprehensive framework for estimating software development effort, cost, and duration. It considers a broader range of project attributes and characteristics compared to its predecessor, offering more accuracy and flexibility in estimation.

Key features and improvements in COCOMO II (2000) include:

1. **Three Sub-models:** COCOMO II consists of three different sub-models to cater to different stages of software development:
    a. **Application Composition Model:** Used for estimating costs in projects involving integrating existing software components.
    b. **Early Design Model:** Estimates costs based on early design decisions before detailed specifications are available.
    c. **Post-Architecture Model:** Suitable for estimating costs after the software architecture has been defined.
2. **Expanded and Refined Cost Drivers:** COCOMO II includes a more extensive set of cost drivers compared to the original COCOMO. These cost drivers consider various project attributes, team factors, process characteristics, and product attributes to provide more accurate estimations.
3. **Enhanced Estimation Granularity:** COCOMO II allows for more detailed estimation by breaking down different components, phases, and attributes of the software project, providing a more fine-grained view of the estimation process.
4. **Improved Sensitivity Analysis:** COCOMO II enables better sensitivity analysis, allowing project managers to understand how changes in different parameters affect the overall estimates.
5. **Better Reusability Modeling:** COCOMO II provides improved models for estimating costs when reusing components or leveraging existing software assets, considering their impact on project effort and cost.

6. **Updated Estimation Equations:** COCOMO II includes updated estimation equations and algorithms, increasing the accuracy of project cost, effort, and schedule predictions.

COCOMO II (2000) is widely recognized and used in the software industry due to its more sophisticated and customizable nature compared to the original COCOMO model. It offers software development teams a structured framework for estimation, aiding in better planning, resource allocation, and decision-making throughout the software development lifecycle.

Definition of Software Development Lifecycle Management

Software Development Life Cycle (SDLC) Management is a process that aims to develop software with the lowest cost, highest quality, and in the shortest time. It also includes detailed documentation for how to develop, extend, and maintain the software system.

A **Software Development Life Cycle** involves several different stages, including requirements gathering, planning/designing, building, testing, and finally deployment.

Description of Software Development Life Cycle Management

Some of the most popular **Software Development Life Cycle Management** methodologies include Spiral Development, Agile, and SCRUM.

Teams following the best practices of the Software Development Life Cycle Management process see more success and have an easier time developing software in stages. The SDLC process applies equally well for Minimum Viable Products which are part of the Lean Startup approach, as well as full blown projects.

The main stages of the Software Development Life Cycle Management process include:

1. **Identify Requirements** – The first stage is about understanding the problems you are trying to solve, and what the software needs to do.
2. **Plan & Design** – Taking into account the requirements, the next step is to plan and prioritize the features that need to be created. This phase can and should include some research to validate technologies and approaches.
3. **Build / Code** – This stage is one of the longest, and where much of the work is done. It should overlap to some degree with the following Documentation and Test phase.
4. **Test / Debug / Document** – Documenting and testing should happen during the build stage as well, but this stage is where the focus turns from developing features to bug fixing and stabilizing the software for launch to customers.
   Often Alpha and Beta versions are released in this phase to ensure configuration and

other rare or hard to spot bugs are found and fixed.

5. **Deploy** – The final deployment stage is where the software is released for customers to use and goes live.

Tools & Templates

Software Development Life Cycle Management tools and templates include many kinds of charting software, spreadsheets, or simply a long, horizontal paper that can be drawn on and updated for each of the stages.

upBOARD's Software Development Life Cycle (SDLC) Management Tools & Templates upBOARD provides a strategic view of your software development life cycle (SDLC), a line of sight to your software development strategy, and a complete dashboard for tracking activities and results. Using upBOARD, your software development process becomes a living "board" that's always on, current, and available on the cloud for everyone to see. Our experts In agile software development have assembled online tools and templates to help guide any software developer, software development manager, team or organization across the software development lifecycle. Use these best practices as standalone processes or mix and match tools and templates to create your own leading-edge collaborative environment, including integrating data from Google Drive, JIRA and other tools.