Requirement Analysis and Cost Estimation

Content

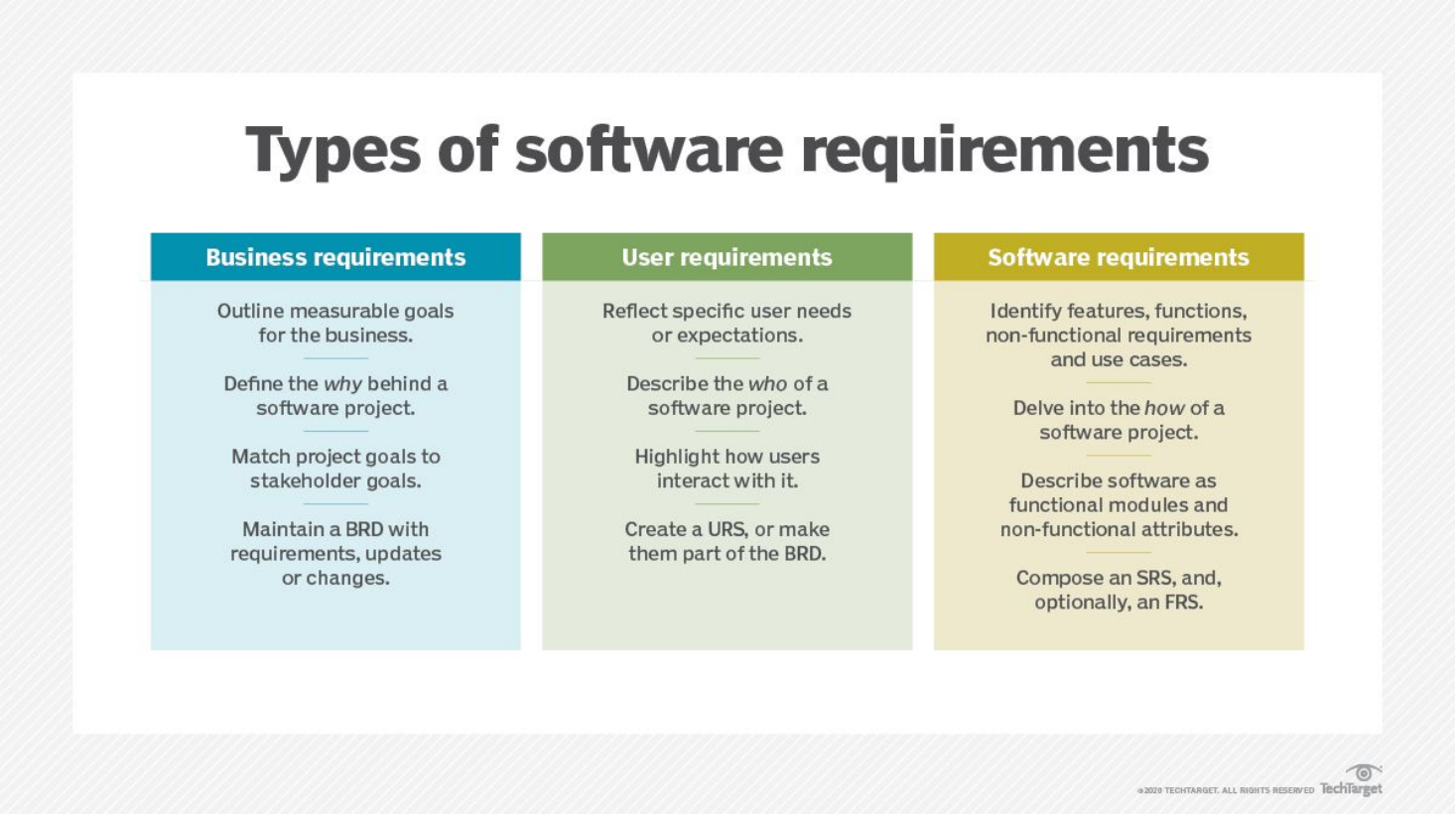
- Software Requirements: Functional & non-functional
- user-system requirement engineering process – feasibility studies – elicitation – validation & management – software prototyping – S/W documentation – Analysis and modelling Requirement Elicitation,
- Software requirement specification (SRS)
- 3Ps (people, product and process) Process and Project metrics Software
- Project Estimation: LOC, FP, Empirical Estimation Models - COCOMO II Model

Requirements

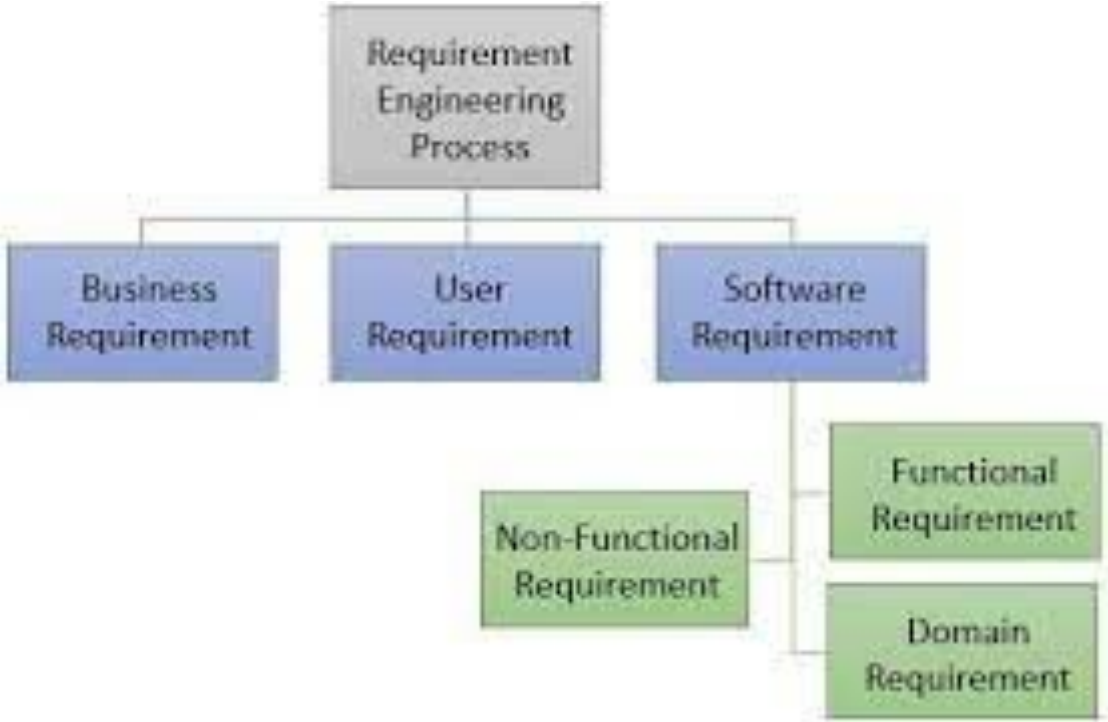
- The **information which describes the user's expectations about the system performance** is called as requirements.
- Characteristics:
  - Unambiguous
  - Verifiable
  - Clear
  - Understandable
  - Feasible
  - consistent

Requirement Engineering: the procedure which collects the software requirements from customer , analyze and document them is called requirement engg.

Types of Requirements



Types of Requirements



Functional Requirements

- These are the requirements that the end user specifically demands as basic facilities that the system should offer. It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely to perform.
- All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected.
- They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.
- **For example**, in a hospital management system, a doctor should be able to retrieve the information of his patients.
- Each high-level functional requirement may involve several interactions or dialogues between the system and the outside world.
- In order to accurately describe the functional requirements, all scenarios must be enumerated.
- There are many ways of expressing functional requirements e.g., natural language, a structured or formatted language with no rigorous syntax and formal specification language with proper syntax. Functional Requirements in Software Engineering are also called Functional Specification.

Non-Functional Requirement

- These are basically the quality constraints that the system must satisfy according to the project contract.
- Nonfunctional requirements, not related to the system functionality, rather define how the system should perform.
- The priority or extent to which these factors are implemented varies from one project to other. They are also called non-behavioral requirements.
- They basically deal with issues like:
  - Portability
  - Security
  - Maintainability
  - Reliability
  - Scalability
  - Performance
  - Reusability
  - Flexibility

Requirement Engineering Tasks

The software requirements engineering process includes the following steps of activities:

1. **Inception**
2. **Elicitation**
3. **Elaboration**
4. **Negotiation**
5. **Specification**
6. **Validation**
7. **Requirements Management**

Requirement Engineering Tasks

Inception: This is the first phase of the requirements analysis process. This phase gives an outline of how to get started on a project. In the inception phase, all the basic questions are asked on how to go about a task or the steps required to accomplish a task. A basic understanding of the problem is gained and the nature of the solution is addressed. Effective communication is very important in this stage, as this phase is the foundation as to what has to be done further. Overall in the inception phase, the following criteria have to be addressed by the software engineers:

- Understanding of the problem.
- The people who want a solution.
- Nature of the solution.
- Communication and collaboration between the customer and developer.

Requirement Engineering Tasks

Elicitation: This is the second phase of the requirements analysis process. This phase focuses on gathering the requirements from the stakeholders. One should be careful in this phase, as the requirements are what establishes the key purpose of a project. Understanding the kind of requirements needed from the customer is very crucial for a developer. In this process, mistakes can happen in regard to, not implementing the right requirements or forgetting a part. The right people must be involved in this phase. The following problems can occur in the elicitation phase:

- **Problem of Scope**: The requirements given are of unnecessary detail, ill-defined, or not possible to implement.
- **Problem of Understanding:** Not having a clear-cut understanding between the developer and customer when putting out the requirements needed. Sometimes the customer might not know what they want or the developer might misunderstand one requirement for another.
- **Problem of Volatility:** Requirements changing over time can cause difficulty in leading a project. It can lead to loss and wastage of resources and time.

Requirement Engineering Tasks

Elaboration: This is the third phase of the requirements analysis process. This phase is the result of the inception and elicitation phase. In the elaboration process, it takes the requirements that have been stated and gathered in the first two phases and **refines** them. **Expansion and looking into it** further are done as well. The main task in this phase is to indulge in modeling activities and develop a prototype that elaborates on the features and constraints using the necessary tools and functions.

Requirement Engineering Tasks

Negotiation: This is the fourth phase of the requirements analysis process. This phase emphasizes discussion and exchanging conversation on what is needed and what is to be eliminated. In the negotiation phase, negotiation is between the developer and the customer and they dwell on how to go about the project with limited business resources. Customers are asked to prioritize the requirements and make guesstimates on the conflicts that may arise along with it. Risks of all the requirements are taken into consideration and negotiated in a way where the customer and developer are both satisfied with reference to the further implementation. The following are discussed in the negotiation phase:

- **Availability of Resources.**
- **Delivery Time.**
- **Scope of requirements.**
- **Project Cost.**
- **Estimations on development.**

Requirement Engineering Tasks

Specification: This is the **fifth phase** of the requirements analysis process. This phase specifies the following:

- Written document.
- A set of models.
- A collection of use cases.
- A prototype.

In the specification phase, the requirements engineer gathers all the requirements and develops a working model. This final working product will be the basis of any functions, features or constraints to be observed. The models used in this phase include ER (Entity Relationship) diagrams, DFD (Data Flow Diagram), FDD (Function Decomposition Diagrams), and Data Dictionaries.

A software specification document is submitted to the customer in a language that he/she will understand, to give a glimpse of the working model.

Requirement Engineering Tasks

Validation: This is the **sixth phase** of the requirements analysis process. This phase focuses on checking for errors and debugging. In the validation phase, the developer scans the specification document and checks for the following:

- All the requirements have been stated and met correctly
- Errors have been debugged and corrected.
- Work product is built according to the standards.

This requirements validation mechanism is known as the formal technical review. The review team that works together and validates the requirements include software engineers, customers, users, and other stakeholders. Everyone in this team takes part in checking the specification by examining for any errors, missing information, or anything that has to be added or checking for any unrealistic and problematic errors. Some of the validation techniques are the following-

- Requirements reviews/inspections.
- Prototyping.
- Test-case generation.
- Automated consistency analysis.

Requirement Engineering Tasks

Requirements Management: This is the **last phase** of the requirements analysis process. Requirements management is a set of activities where the entire team takes part in identifying, controlling, tracking, and establishing the requirements for the successful and smooth implementation of the project.

In this phase, the team is responsible for managing any changes that may occur during the project. New requirements emerge, and it is in this phase, responsibility should be taken to manage and prioritize as to where its position is in the project and how this new change will affect the overall system, and how to address and deal with the change. Based on this phase, the working model will be analyzed carefully and ready to be delivered to the customer.

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.

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

Importance of Requirements Elicitation

1. **Compliance with Business Objectives**: The process of elicitation guarantees that the software development endeavours are in harmony with the wider company aims and objectives. Comprehending the business context facilitates the development of a solution that adds value for the company.
2. **User Satisfaction:** It is easier to create software that fulfils end users needs and expectations when they are involved in the requirements elicitation process. Higher user pleasure and acceptance of the finished product are the results of this.
3. **Time and Money Savings**: Having precise and well-defined specifications aids in preventing miscommunication and rework during the development phase. As a result, there will be cost savings and the project will be completed on time.
4. **Compliance and Regulation Requirements:** Requirements elicitation is crucial for projects in regulated industries to guarantee that the software conforms with applicable laws and norms. In industries like healthcare, finance, and aerospace, this is crucial.
5. **Traceability and Documentation:** Throughout the software development process, traceability is based on requirements that are well-documented. Traceability helps with testing, validation, and maintenance by ensuring that every part of the software can be linked to a particular requirement.

Requirements Elicitation Methods

1. Interviews

Objective of conducting an interview is to understand the customer's expectations from the software.

It is impossible to interview every stakeholder hence representatives from groups are selected based on their expertise and credibility. Interviews maybe be open-ended or structured.

In open-ended interviews there is no pre-set agenda. Context free questions may be asked to understand the problem.

In a structured interview, an agenda of fairly open questions is prepared. Sometimes a proper questionnaire is designed for the interview.

Requirements Elicitation Methods

2. Brainstorming Sessions

It is a group technique.

It is intended to generate lots of new ideas hence providing a platform to share views.

A highly trained facilitator is required to handle group bias and group conflicts.

Every idea is documented so that everyone can see it.

Finally, a document is prepared which consists of the list of requirements and their priority if possible.

Requirements Elicitation Methods

3. Facilitated Application Specification Technique

Its objective is to bridge the expectation gap – the difference between what the developers think they are supposed to build and what customers think they are going to get. A team-oriented approach is developed for requirements gathering. Each attendee is asked to make a list of objects that are:

- Part of the environment that surrounds the system.
- Produced by the system.
- Used by the system.

Each participant prepares his/her list, different lists are then combined, redundant entries are eliminated, team is divided into smaller sub-teams to develop mini-specifications and finally a draft of specifications is written down using all the inputs from the meeting.

Requirements Elicitation Methods

4. Quality Function Deployment

In this technique customer satisfaction is of prime concern, hence it emphasizes on the requirements which are valuable to the customer.

3 types of requirements are identified:

- Normal requirements: In this the objective and goals of the proposed software are discussed with the customer. Example – normal requirements for a result management system may be entry of marks, calculation of results, etc.
- Expected requirements: These requirements are so obvious that the customer need not explicitly state them. Example – protection from unauthorized access.
- Exciting requirements: It includes features that are beyond customer's expectations and prove to be very satisfying when present. Example – when unauthorized access is detected, it should backup and shutdown all processes.

Requirements Elicitation Methods

5. Use Case Approach

This technique combines text and pictures to provide a better understanding of the requirements.

The use cases describe the ‘what’, of a system and not ‘how’. Hence, they only give a functional view of the system.

The components of the use case design include three major things – Actor, use cases, use case diagram.

- Actor: It is the external agent that lies outside the system but interacts with it in some way. An actor maybe a person, machine etc. It is represented as a stick figure. Actors can be primary actors or secondary actors.
  - Primary actors: It requires assistance from the system to achieve a goal.
  - Secondary actor: It is an actor from which the system needs assistance.
- Use cases: They describe the sequence of interactions between actors and the system. They capture who(actors) do what(interaction) with the system. A complete set of use cases specifies all possible ways to use the system.
- Use case diagram: A use case diagram graphically represents what happens when an actor interacts with a system. It captures the functional aspect of the system.
  - A stick figure is used to represent an actor.
  - An oval is used to represent a use case.
  - A line is used to represent a relationship between an actor and a use case.

SRS

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

SRS

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

SRS Structure

SRS document comprises the following sections:

- Introduction
- General Description
- Functional Requirements
- Interface Requirements
- Performance Requirements
- Design Constraints
- Non Functional Attributes
- Preliminary Schedule and Budget
- Appendices

Introduction

- **Purpose of this Document** – At first, main aim of why this document is necessary and what's purpose of document is explained and described.
- **Scope of this document** – In this, overall working and main objective of document and what value it will provide to customer is described and explained. It also includes a description of development cost and time required.
- **Overview** – In this, description of product is explained. It's simply summary or overall review of product.

General Description

In this, general functions of product which **includes**

- objective of user,
- a user characteristic,
- features,
- benefits,
- its importance

Functional Requirements

- In this, possible outcome of software system which includes effects due to operation of program is fully explained.
- All functional requirements which may include calculations, data processing, etc. are placed in a ranked order.
- Functional requirements specify the expected behavior of the system-which outputs should be produced from the given inputs.
- They describe the relationship between the input and output of the system.
- For each functional requirement, detailed description all the data inputs and their source, the units of measure, and the range of valid inputs must be specified.

Interface Requirements

In this, software interfaces which mean how software program communicates with each other or users either in form of any language, code, or message are fully described and explained.

Performance Requirements

- In this, how a software system performs desired functions under specific condition is explained.
- It also explains required time, required memory, maximum error rate, etc.
- The performance requirements part of an SRS specifies the performance constraints on the software system.
- All the requirements relating to the performance characteristics of the system must be clearly specified.
- There are two types of performance requirements: static and dynamic.
- Static requirements are those that do not impose constraint on the execution characteristics of the system.
- Dynamic requirements specify constraints on the execution behaviour of the system.

Design Constraints

- In this, constraints which simply means limitation or restriction are specified and explained for design team.
- Examples may include use of a particular algorithm, hardware and software limitations, etc.
- There are a number of factors in the client's environment that may restrict the choices of a designer leading to design constraints such factors include standards that must be followed resource limits, operating environment, reliability and security requirements and policies that may have an impact on the design of the system.
- An SRS should identify and specify all such constraints.

Non Functional Attributes

- In this, non-functional attributes are explained that are required by software system for better performance.
- An example may include
  - Security,
  - Portability,
  - Reliability,
  - Reusability,
  - Application compatibility,
  - Data integrity,
  - Scalability
  - capacity, etc.

Preliminary Schedule and Budget

In this, initial version and budget of project plan are explained which include overall time duration required and overall cost required for development of project.

Appendices

In this, additional information like references from where information is gathered, definitions of some specific terms, acronyms, abbreviations, etc. are given and explained.

3P’s

For properly building a product, there’s a very important concept that we all should know in software project planning while developing a product. There are 3 critical components in software project planning which are known as the 3P’s namely:

- **Product**
- **Process**
- **People**

These components play a very important role in your project that can help your team meet its goals and objective.

People

- The most important component of a product and its successful implementation is human resources.
- In building a proper product, a well-managed team with clear-cut roles defined for each person/team will lead to the success of the product.
- We need to have a good team in order to save our time, cost, and effort.
- Some assigned roles in software project planning are project manager, team leaders, stakeholders, analysts, and other IT professionals.
- Managing people successfully is a tricky process which a good project manager can do.

Product

- As the name inferred, this is the deliverable or the result of the project.
- The project manager should clearly define the product scope to ensure a successful result, control the team members, as well technical hurdles that he or she may encounter during the building of a product.
- The product can consist of both tangible or intangible such as shifting the company to a new place or getting a new software in a company.

Process

- In every planning, a clearly defined process is the key to the success of any product.
- It regulates how the team will go about its development in the respective time period.
- The Process has several steps involved like, documentation phase, implementation phase, deployment phase, and interaction phase.

Process and Project metrics

a) Process Metrics

- These are the metrics pertaining to the Process Quality. They measure efficiency and effectiveness of various processes.

b) Project Metrics

- These are the metrics pertaining to the Project Quality. They measure defects, cost, schedule, productivity and estimation of various project resources and deliverables.

Project Metrics

- **Schedule Variance** : Any difference between the scheduled completion of an activity and the actual completion is known as Schedule Variance. Schedule variance = ((Actual calendar days – Planned calendar days) + Start variance)/ Planned calendar days x 100.

  The **Start Variance** field contains the amount of time that represents the difference between a baseline start date of a task or assignment and its currently scheduled start date

- **Effort Variance**: Difference between the planned outlined effort and the effort required to actually undertake the task is called Effort variance. Effort variance = (Actual Effort – Planned Effort)/ Planned Effort x 100.

Project Metrics

- **Size Variance**: Difference between the estimated size of the project and the actual size of the project (normally in KLOC or FP) Size variance = (Actual size – Estimated size)/ Estimated size x 100.

Project Metrics

- **Requirement Stability Index**: Provides visibility and understanding into the magnitude and impact of requirements changes. RSI = 1- ((No of changed + No of deleted + No of added) / Total no of Initial requirements) x100
- **Productivity (Project)**: It is a measure of output from a related process for a unit of input. Project Productivity = Actual Project Size / Actual Effort spent for the project
- **Productivity (for test case preparation):** Productivity in test case preparation = Actual no of test cases/ actual effort spent on test case preparation

Project Metrics

- **Productivity (for test case execution):** Productivity in test case execution = actual number of test cases / actual effort spend on testing.
- **Productivity (defect detection):** Productivity in defect detection = Actual number of defects (review + testing) / actual effort spent on (review + testing)
- **Productivity (defect fixation):** Productivity in defect fixation = actual no of defects fixed/ actual effort spent on defect fixation

Project Metrics

- **Schedule variance for a phase**: The deviation between planned and actual schedule for the phases within a project. Schedule variance for a phase = (Actual Calendar days for a phase – Planned calendar days for a phase + Start variance for a phase)/ (Planned calendar days for a phase) x 100
- **Effort variance for a phase**: The deviation between planned and actual effort for various phases within the project. Effort variance for a phase = (Actual effort for a phase – planned effort for a phase)/ (planned effort for a phase) x 100

Process Metrics

- **Cost of Quality**: It is a measure in terms of money for the quality performance within an organization. Cost of quality = (review + testing + verification review + verification testing + QA + configuration management + measurement + training + rework review + rework testing)/ total effort x 100
- **Cost of poor Quality**: It is the cost of implementing imperfect processes and products. Cost of poor quality = rework effort/ total effort x 100
- **Defect Density**: It is the number of defects detected in the software during the development divided by the size of the software (typically in KLOC or FP) Defect density for a project = Total number of defects/ project size in KLOC or FP

Process Metrics

- **Review Efficiency**: defined as the efficiency in harnessing/ detecting review defects in the verification stage. Review efficiency = (number of defects caught in review)/ total number of defects caught) x 100
- **Testing Efficiency**: Testing efficiency = 1 – ((defects found in acceptance)/ total no of testing defects) x 100
- **Defect removal efficiency**: Gives the efficiency with which defects were detected and minimum defects were filtered down to the customer. Defect removal efficiency = (1 – (total defects caught by customer/ total no of defects)) x 100

Software Project Estimation

- **Software measurement** is indicator of size, quantity amount or dimension of particular attribute of a product or process.
- It helps the project manager and entire software team to take decisions that lead to successful completion of the project by generating quantity result.
- Two categories of software measurements:
  - Direct Measures: It include software processes like Cost and Effort applied, Lines of code produced, Execution speed and total no. of errors that have been reported.
  - Indirect Measures: It includes products like functionality, quality, complexity, reliability, maintainability and many more.

Software Metrics

- Software metrics provide measures, functions or formulas for various aspects of software process and product.
- It including **measur**ing software performance, planning work items, measuring productivity and many other uses
- Software metrics of three categories:
  - Product metrics: it estimate size, complexity, quality and reliability of software.
  - Process metrics: it estimate fault rate during development, pattern of testing defect arrival, time it takes for a fixed operation.
  - Project metrics: it estimate number of software developers, cost, scheduling and productivity of software.

Size metrics

Size oriented metrics concentrates on the size of the program created.

- LOC(Lines of Code)
- FP(Functional Point)

LOC (Lines of Code)

- It is one of the earliest and simpler metrics for calculating the size of the computer program
- It is generally used in calculating and comparing the productivity of programmers.
- LOC not count comment or blank line in code
- In given example:
  - Total line of code=09

```
//Import header file
#include <iostream>
int main () {
  int num = 10;
//Logic of even number
        if (num % 2 == 0)
        {
           cout<<"It is even number";
        }
  return 0;
}
```

LOC

Advantages:

- Most used metric in cost estimation.
- It is very easy in estimating the efforts.

Disadvantages:

- It cannot measure the size of the specification.
- Bad software design may cause an excessive line of code
- It is language dependent
- Users cannot easily understand it.

LOC

Project	LOC	Cost $K	Efforts (Persons/ Month)	Documentation	Errors
A	10000	110	18	365	39
B	12000	115	20	370	45
C	15400	130	25	400	32

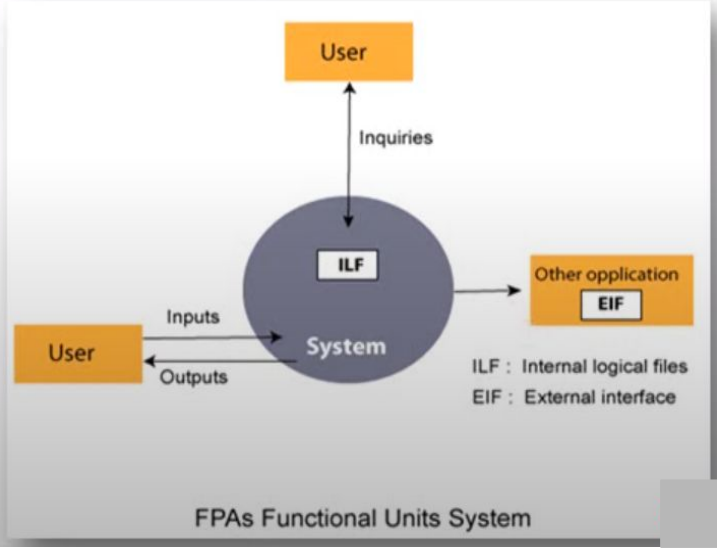
LOC Table

FP(Functional Point)

- Function oriented software metrics measure functionality, what the system performs, is the measure of the system size.
- It find out by counting the number and type of functions used in applications.

FP Attributes

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



FPAs Functional Units System

Functional Point

The steps in function point analysis are:

- Count the number of functions of each proposed type.
- Compute the Unadjusted Function Points(UFP).
- Find the Total Degree of Influence(TDI).
- Compute Value Adjustment Factor(VAF).
- Find the Function Point Count(FPC).

Functional Point

Count the number of functions of each proposed type: Find the number of functions belonging to the following types:

- External Inputs: Functions related to data entering the system.
- External outputs: Functions related to data exiting the system.
- External Inquiries: They lead to data retrieval from the system but don’t change the system.
- Internal Files: Logical files maintained within the system. Log files are not included here.
- External interface Files: These are logical files for other applications which are used by our system.

Functional Point

Compute the Unadjusted Function Points(UFP): Categorise each of the five function types like simple, average, or complex based on their complexity. Multiply the count of each function type with its weighting factor and find the weighted sum. The weighting factors for each type based on their complexity are as follows:

Function type	Simple	Average	Complex
External Inputs	3	4	6
External Output	4	5	7
External Inquiries	3	4	6
Internal Logical Files	7	10	15
External Interface Files	5	7	10

Functional Point

Find Total Degree of Influence: Use the '14 general characteristics' of a system to find the degree of influence of each of them. The sum of all 14 degrees of influence will give the TDI. The range of TDI is 0 to 70. The 14 general characteristics are: *Data Communications, Distributed Data Processing, Performance, Heavily Used Configuration, Transaction Rate, On-Line Data Entry, End-user Efficiency, Online Update, Complex Processing Reusability, Installation Ease, Operational Ease, Multiple Sites and Facilitate Change.*

Each of the above characteristics is evaluated on a scale of 0-5.

Functional Point

Compute Value Adjustment Factor(VAF): Use the following formula to calculate VAF

VAF = (TDI * 0.01) + 0.65

Find the Function Point Count: Use the following formula to calculate FPC

FPC = UFP * VAF

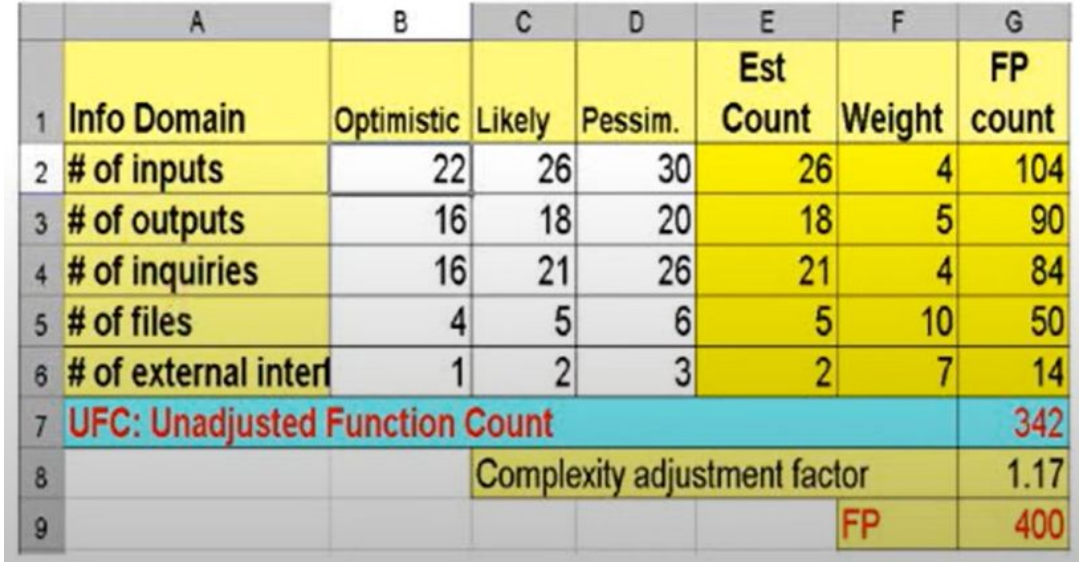
Functional Point

Advantages:

- Need detail specification
- Not restricted code
- Language interdependent

Disadvantages:

- Ignore quality issues



	A	B	C	D	E	F	G
1	Info Domain	Optimistic	Likely	Pessim.	Est Count	Weight	FP count
2	# of inputs	22	26	30	26	4	104
3	# of outputs	16	18	20	18	5	90
4	# of inquiries	16	21	26	21	4	84
5	# of files	4	5	6	5	10	50
6	# of external interf	1	2	3	2	7	14
7	UFC: Unadjusted Function Count						342
8			Complexity adjustment factor				1.17
9						FP	400

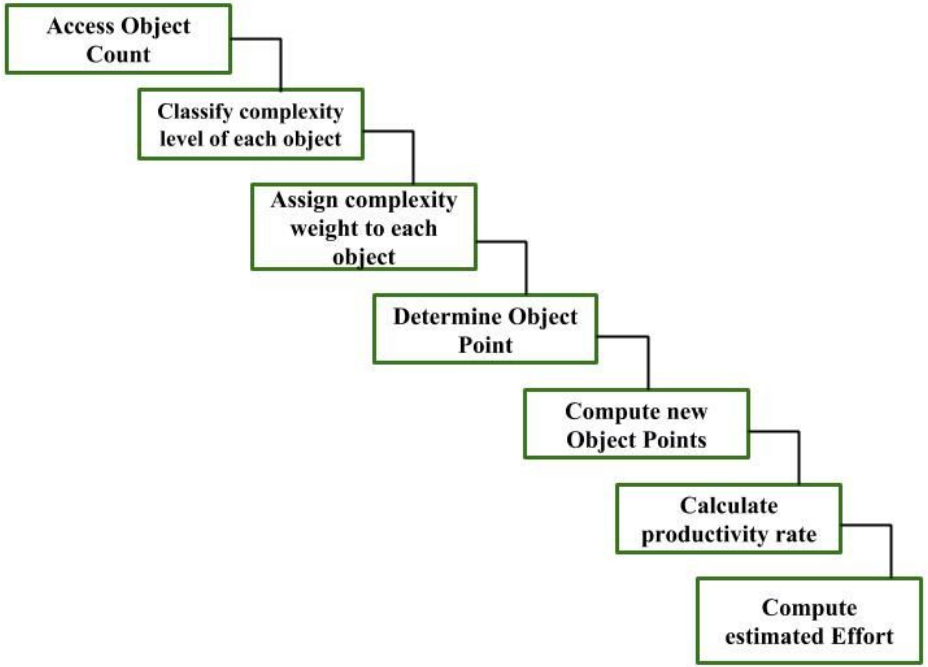
COCOMO II

Application Composition Estimation Model allows one to estimate the cost, and effort at stage 1 of the COCOMO II [Constructive Cost Model] Model. In this model, size is first estimated using Object Points. Object Points are easy to identify and count. Object Points define screen, reports, and third-generation (3GL) modules as objects. Object Point estimation is a new size estimation technique but it is well suited in Application Composition Sector.

COCOMO II

Estimation of Efforts

The following steps are taken to estimate the effort to develop a project:



COCOMO II

1. Access Object Counts

Estimate the number of screens, reports and 3GL components that will comprise this application.

2. Classify Complexity Levels of Each Object

We have to classify each object instance into simple, medium and difficult complexity level depending on values of its characteristics. Complexity levels are assigned according to the given table.

COCOMO II

No. of views contain	Sources of data tables		
	Total < 4 (< 2 servers < 3 clients)	Total < 8 (2 - 3 servers 3-5 clients)	Total 8 + (> 3 servers > 5 clients)
< 3	Simple	Simple	Medium
3 - 7	Simple	Medium	Difficult
> 8	Medium	Difficult	Difficult

For Screens

COCOMO II

No. of section contain	Sources of data tables		
	Total < 4 (< 2 servers < 3 clients)	Total < 8 (2 - 3 servers 3-5 clients)	Total 8 + (> 3 servers > 5 clients)
0 - 1	Simple	Simple	Medium
2 - 3	Simple	Medium	Difficult
4 +	Medium	Difficult	Difficult

For Reports

COCOMO II

3. Assign Complexity Weights to Each Object

The weights are used for three object types i.e, screens, reports and 3GL components. Complexity weight are assigned according to object's complexity level using following table.

Object Type	Complexity Weight		
	Simple	Medium	Difficult
Screen	1	2	3
Report	2	5	8
3GL Components	-	-	10

Complexity Weight

COCOMO II

4. Determine Object Points

Add all the weighted object instances to get one number and this is known as object point count.

Object Point = Sigma (number of object instances) * (Complexity weight of each object instance)

COCOMO II

5. Compute New Object Points (NOP)

We have to estimate the %reuse to be achieved in a project. Depending on %reuse

NOP = [(object points) * (100 – % reuse)] / 100

NOP are the object point that will need to be developed and differ from the object point count because there may be reuse of some object instance in project.

COCOMO II

6. Calculate Productivity rate (PROD)

Productivity rate is calculated on the basis of information given about developer's experience and capability. For calculating it, we use following table.

Developers experience & capability	Productivity (PROD)
Very Low	4
Low	7
Nominal	13
High	25
High	50

Productivity Rate

COCOMO II

7. Compute the estimated Effort

Effort to develop a project can be calculated as:

Effort = NOP / PROD

Effort is measured in person-month.

Example

Consider a database application project with

The application has four screens with four views each and seven data tables for three servers and four clients.

Application may generate two reports of six section each from seven data tables for two servers and three clients.

10% reuse of object points.

Developer’s experience and capability in similar environment is low.

Calculate the object point count, New object point and effort to develop such project.

Answer

Step 1: Number of screens = 4 and Number of records = 2

Step 2: For screens,

Number of views = 4

Number of data tables = 7

Number of servers = 3

Number of clients = 4

By using above given information and table (For Screens), Complexity level for each screen = medium

Answer

For reports,

Number of sections = 6

Number of data tables = 7

Number of servers = 2

Number of clients = 3

By using above given information and table (For Reports), Complexity level for each report = difficult.

Answer

Step 3: By using complexity weight table we can assign complexity weight to each object instance depending upon their complexity level.

Complexity weight for each screen = 2

Complexity weight for each report = 8

Answer

Step 4:

Object point count = sigma (Number of object instances) * (its Complexity weight)

= 4 * 2 + 2 * 8 = 24

Answer

Step 5:

%reuse of object points = 10% (given)

NOP = [object points * (100 – %reuse)] / 100

= [24 * (100 -10)]/100 = 21.6

Answer

Step 6:

Developer’s experience and capability is low (given) Using information given about developer and productivity rate table

Productivity rate (PROD) of given project = 7

Step 7:

Effort = NOP / PROD

= 21.6/7

= 3.086 person-month

Therefore, effort to develop the given project = 3.086 person-month.

2. Use the COCOMO II model to estimate the effort required to build a software for a Medium Reservation System that produces 18 screens, 24 reports, and will require approximately 110 software components.

Past projects show burdened labor rate 2500BD/PM. Reuse is at 25%.

Assume very high developer/environment maturity.

Find NOP, estimated efforts and estimated cost.

(5 Marks = 2 marks for NOP + 1.5 marks for estimation of efforts + 1.5 marks for cost)

Object type	**Complexity Weight**		
	Simple	**Medium**	**Difficult**
Screen	3	13	15
Report	6	17	17
3GL component	11	22	29

Developer's experience	**Very low**	**Low**	**Nominal**	**High**	**Very high**
Development environment maturity & capability	**Very low**	**Low**	**Nominal**	**High**	**Very high**
PROD	5	8	14	26	50

Answer

Step 1: Given: 18 screens, 24 reports, 25% reuse

As per the problem statement, since the system is of Medium size, thus the complexity weight comes out to be Medium.

From the given of object type and complexity weight we calculate the total object points.

Answer

Step 2: Total objects points are calculated using the following formula:

total object points = screens * complexity weight value + reports * complexity weight value + 3GL components * complexity weight value

For screens, the complexity weight value for Medium complexity is 13.

For reports, the complexity weight value for Medium complexity is 17.

total object points = 18 * 13 + 24* 17 = 642

Answer

NOP = ((total object points) * (100 - % reuse)) / 100

= 642 * (100 - 25) / 100

= 642 * 75 / 100

NOP = 481.5

Answer

Step 3: Steps to calculate estimated efforts:

Given: Very high developer / environment maturity

Estimated efforts are calculated by the following formula

efforts = NOP / PROD

Since there is very high developer / environment maturity, thus PROD value as per the given table is 50.

Substituting the value of NOP (calculated above) and PROD we get,

efforts = 481.5 / 50

efforts = 9.63 PM

Example

Calculate the function point for software application with multiple Processing Factors 5, 1, 0, 4, 3, 5, 4, 3, 4, 5, 2, 3, 4, 2 by using following given Date: The number of EI(Avg): 22,The number of EO(Low): 45,The number of EI(High): 06, The number of ILF(Avg): 05, The number of ELF(Low): 02.

Answer

Solution: Total Cost Factor (TC) = (22*4) + (45*4) + (06*6) + (05*10) + (02*5)

= 364

Function Point (FP) = TC *[0.65 + 0.01*∑ (Xi)]

= 364 *[0.65 + 0.01*(5+1+0+4+3+ 5+4+3+4+5+ 2+3+ 4+2)]

= 364* [0.65 + 0.01*45]

= 364 * [0.65 + 0.45]

= 364* 1.10

= 400.4