SOFTWARE ENGINEERING

UNIT 2

Software Project Management: Software project management complexities, Responsibilities of a software project manager, Metrics for project size estimation, Project estimation techniques, Empirical Estimation techniques, COCOMO, Halstead's software science, risk management.

Requirements Analysis and Specification: Requirements gathering and analysis, Software Requirements Specification (SRS), Formal system specification, Axiomatic specification, Algebraic specification, Executable specification and 4GL

SOFTWARE PROJECT MANAGEMENT

Effective project management is crucial to the success of any software development project.

In the past, several projects have failed not for want of competent technical professionals neither for lack of resources, but due to the use of faulty project management practices.

The main goal of software project management is to enable a group of developers to work effectively towards the successful completion of a project.

As can be inferred from the above definition, project management involves use of a set of techniques and skills to steer a project to success.

For large projects, a different member of the team (other than the project manager) assumes the responsibility of technical leadership. The responsibilities of the technical leader includes addressing issues such as which tools and techniques to use in the project, high-level solution to the problem, specific algorithms to use, etc.

SOFTWARE PROJECT MANAGEMENT COMPLEXITIES

Management of software projects is much more complex than management of many other types of projects. The main factors contributing to the complexity of managing a software project are the following:

Invisibility: Software remains invisible, until its development is complete and it is operational. Anything that is invisible, is difficult to manage and control.

Invisibility of software makes it difficult to assess the progress of a project and is a major cause for the complexity of managing a software project.

Changeability: Because the software part of any system is easier to change as compared to the hardware part, the software part is the one that gets most frequently changed.

Frequent changes to the requirements and the invisibility of software are possibly the two major factors making software project management a complex task.

Complexity: Even a moderate sized software has millions of parts (functions) that interact with each other in many ways—data coupling, serial and concurrent runs, state transitions, control dependency, file sharing, etc.

Uniqueness: Every software project is usually associated with many unique features or situations. This makes every project much different from the others

Exactness of the solution: the parameters of a function call in a program are required to be in complete conformity with the function definition.

Team-oriented and intellect-intensive work: Software development projects are akin to research projects in the sense that they both involve team-oriented, intellect-intensive work.

RESPONSIBILITIES OF A SOFTWARE PROJECT MANAGER

i. Job Responsibilities for Managing Software Projects

- A software project manager takes the overall responsibility of steering a project to success.
- This surely is a very hazy job description. In fact, it is very difficult to objectively describe the precise job responsibilities of a project manager.
- The job responsibilities of a project manager ranges from invisible activities like building up of team morale to highly visible customer presentations.
- Most managers take the responsibilities for project proposal writing, project cost estimation, scheduling, project staffing, software process tailoring, project monitoring and control, software configuration management, risk management, managerial report writing and presentation, and interfacing with clients.

We can broadly classify a project manager's varied responsibilities into the following two major categories:
• Project planning, and
• Project monitoring and control.

In the following subsections, we give an overview of these two classes of responsibilities.

Project planning: Project planning is undertaken immediately after the feasibility study phase and before the starting of the requirements analysis and specification phase.

Project planning involves estimating several characteristics of a project and then planning the project activities based on these estimates made.

The initial project plans are revised from time to time as the project progresses and more project data become available.

Project monitoring and control: Project monitoring and control activities are undertaken once the development activities start.

The focus of project monitoring and control activities is to ensure that the software development proceeds as per plan.

While carrying out project monitoring and control activities, a project manager may sometimes find it necessary to change the plan to cope up with specific situations at hand.

ii. Skills Necessary for Managing Software Projects

- Effective software project management calls for good qualitative judgment and decision taking capabilities.
- In addition to having a good grasp of the latest software project management techniques such as cost estimation, risk management, and configuration management, etc., project managers need good communication skills and the ability to get work done.

- Some skills such as tracking and controlling the progress of the project, customer interaction, managerial presentations, and team building are largely acquired through experience.
- Never the less, the importance of a sound knowledge of the prevalent project management techniques cannot be overemphasized.

Three skills that are most critical to successful project management are the following:
- Knowledge of project management techniques.
- Decision taking capabilities.
- Previous experience in managing similar projects.

METRICS FOR PROJECT SIZE ESTIMATION

The project size is a measure of the problem complexity in terms of the effort and time required to develop the product. Currently, two metrics are popularly being used to measure size—lines of code (LOC) and function point (FP).

Lines of Code (LOC)

- LOC is possibly the simplest among all metrics available to measure project size. Consequently, this metric is extremely popular.
- This metric measures the size of a project by counting the number of source instructions in the developed program.
- Obviously, while counting the number of source instructions, comment lines, and header lines are ignored.
- Determining the LOC count at the end of a project is very simple. However, accurate estimation of LOC count at the beginning of a project is a very difficult task
- One can possibly estimate the LOC count at the starting of a project, only by using some form of systematic guess work.
- Systematic guessing typically involves the following. The project manager divides the problem into modules, and each module into sub-modules and so on, until the LOC of the leaf-level modules are small enough to be predicted.

shortcomings of the LOC metrics

1. LOC is a measure of coding activity alone. A good problem size measure should consider the total effort needed to carry out various life cycle activities (i.e. specification, design, code, test, etc.) and not just the coding effort.

2. LOC count depends on the choice of specific instructions: LOC gives a numerical value of problem size that can vary widely with coding styles of individual programmers.

3. LOC measure correlates poorly with the quality and efficiency of the code: Larger code size does not necessarily imply better quality of code or higher efficiency.

4. It is very difficult to accurately estimate LOC of the final program from problem specification: As already discussed, at the project initiation time, it is a very difficult task to accurately estimate the number of lines of code (LOC) that the program would have after development.

Function Point (FP) Metric

- Function point metric was proposed by Albrecht in 1983. This metric overcomes many of the shortcomings of the LOC metric.
- Function point metric has several advantages over LOC metric. One of the important advantages of the function point metric over the LOC metric is that it can easily be computed from the problem specification itself. Using the LOC metric, on the other hand, the size can accurately be determined only after the product has fully been developed.

Conceptually, the function point metric is based on the idea that a software product supporting many features would certainly be of larger size than a product with less number of features.

Though each feature takes some effort to develop, different features may take very different amounts of efforts to develop.

Function point (FP) metric computation

The size of a software product (in units of function points or FPs) is computed using different characteristics of the product identified in its requirements specification. It is computed using the following three steps:

Step 1: Compute the unadjusted function point (UFP) using a heuristic expression.

Step 2: Refine UFP to reflect the actual complexities of the different parameters used in UFP computation.

Step 3: Compute FP by further refining UFP to account for the specific characteristics of the project that can influence the entire development effort.

Step 1: UFP computation

The unadjusted function points (UFP) is computed as the weighted sum of five characteristics of a product as shown in the following expression. The weights associated with the five characteristics were determined empirically by Albrecht through data gathered from many projects.

```
UFP = (Number of inputs)*4 + (Number of outputs)*5 +
 (Number of inquiries)*4 + (Number of files)*10 +
 (Number of interfaces)*10                  (3.1)
```

1. **Number of inputs:** Each data item input by the user is counted. It needs to be further noted that individual data items input by the user are not simply added up to compute the number of inputs, but related inputs are grouped and considered as a single input.
2. **Number of outputs:** The outputs considered include reports printed, screen outputs, error messages produced, etc. While computing the number of outputs, the individual data items within a report are not considered; but a set of related data items is counted as just a single output.
3. **Number of inquiries:** An inquiry is a user command (without any data input) and only requires some actions to be performed by the system. Examples of such inquiries are print account balance, print all student grades, display rank holders' names, etc.

4. **Number of files:** The files referred to here are logical files. A logical file represents a group of logically related data. Logical files include data structures as well as physical files.
5. **Number of interfaces:** Here the interfaces denote the different mechanisms that are used to exchange information with other systems. Examples of such interfaces are data files on tapes, disks, communication links with other systems, etc

Step 2: Refine parameters

UFP computed at the end of step 1 is a gross indicator of the problem size. This UFP needs to be refined. This is possible, since each parameter (input, output, etc.) has been implicitly assumed to be of average complexity.

For example, some input values may be extremely complex, some very simple, etc. In order to take this issue into account, UFP is refined by taking into account the complexities of the parameters of UFP computation (Eq. 3.1).

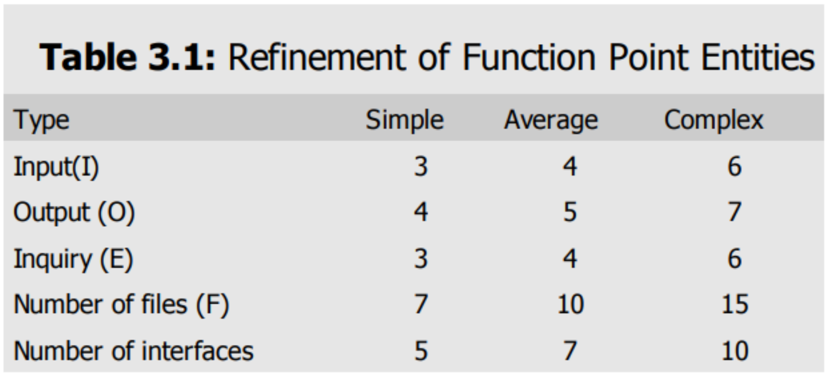

Table 3.1: Refinement of Function Point Entities

Type	Simple	Average	Complex
Input(I)	3	4	6
Output (O)	4	5	7
Inquiry (E)	3	4	6
Number of files (F)	7	10	15
Number of interfaces	5	7	10

Step 3: Refine UFP based on complexity of the overall project

- In the final step, several factors that can impact the overall project size are considered to refine the UFP computed in step 2.
- Examples of such project parameters that can influence the project sizes include high transaction rates, response time requirements, scope for reuse, etc.
- Albrecht identified 14 parameters that can influence the development effort. The list of these parameters have been shown in Table 3.2.

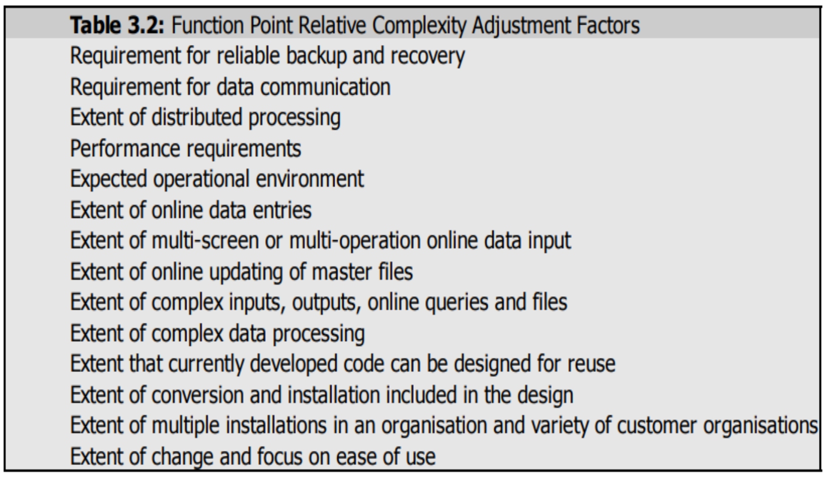

Table 3.2: Function Point Relative Complexity Adjustment Factors
Requirement for reliable backup and recovery
Requirement for data communication
Extent of distributed processing
Performance requirements
Expected operational environment
Extent of online data entries
Extent of multi-screen or multi-operation online data input
Extent of online updating of master files
Extent of complex inputs, outputs, online queries and files
Extent of complex data processing
Extent that currently developed code can be designed for reuse
Extent of conversion and installation included in the design
Extent of multiple installations in an organisation and variety of customer organisations
Extent of change and focus on ease of use

- The resulting numbers are summed, yielding the total **degree of influence (DI).**

- A **technical complexity factor (TCF)** for the project is computed and the TCF is multiplied with UFP to yield FP.

FP=UFP*TCF

Empirical Estimation Techniques

- Empirical estimation techniques are essentially based on making an educated guess of the project parameters.
- While using this technique, prior experience with development of similar products is helpful.
- Although empirical estimation techniques are based on common sense and subjective decisions, over the years, the different activities involved in estimation have been formalised to a large extent.

RISK MANAGEMENT

Every project is susceptible to a large number of risks. Without effective management of the risks, even the most meticulously planned project may go hay ware.

A risk is any anticipated unfavourable event or circumstance that can occur while a project is underway.

In this context, risk management aims at reducing the chances of a risk becoming real as well as reducing the impact of a risks that becomes real. Risk management consists of three essential activities—risk identification, risk assessment, and risk mitigation.

(i) Risk Identification

- The project manager needs to anticipate the risks in a project as early as possible.
- As soon as a risk is identified, effective risk management plans are made, so that the possible impacts of the risks is minimised.
- So, early risk identification is important. Risk identification is somewhat similar to the project manager listing down his nightmares.
  A project can be subject to a large variety of risks. In order to be able to systematically identify the important risks which might affect a project, it is necessary to categorise risks into different classes.

  The project manager can then examine which risks from each class are relevant to the project. There are three main categories of risks which can affect a software project: project risks, technical risks, and business risks.

  Project risks: Project risks concern various forms of budgetary, schedule, personnel, resource, and customer-related problems. An important project risk is schedule slippage.

  Technical risks: Technical risks concern potential design, implementation, interfacing, testing, and maintenance problems. Technical risks also include ambiguous specification, incomplete specification, changing specification, technical uncertainty, and technical obsolescence.

  Business risks: This type of risks includes the risk of building an excellent product that no one wants, losing budgetary commitments, etc.

(ii) Risk Assessment

The objective of risk assessment is to rank the risks in terms of their damage causing potential. For risk assessment, first each risk should be rated in two ways:

- The likelihood of a risk becoming real (r).
- The consequence of the problems associated with that risk (s).

Based on these two factors, the priority of each risk can be computed as follows:

$$p = r * s$$

where, p is the priority with which the risk must be handled, r is the probability of the risk becoming real, and s is the severity of damage caused due to the risk becoming real.

If all identified risks are prioritised, then the most likely and damaging risks can be handled firs

(iii) Risk Mitigation

After all the identified risks of a project have been assessed, plans are made to contain the most damaging and the most likely risks first.
Different types of risks require different containment procedures

There are three main strategies for risk containment:

Avoid the risk: Risks can be avoided in several ways. Risks often arise due to project constraints and can be avoided by suitably modifying the constraints.

Transfer the risk: This strategy involves getting the risky components developed by a third party, buying insurance cover, etc.

Risk reduction: This involves planning ways to contain the damage due to a risk.

- The most important risk reduction techniques for technical risks is to build a prototype that tries out the technology that you are trying to use.

Risk leverage is the difference in risk exposure divided by the cost of reducing the risk. More formally,

$$\text{risk leverage} = \frac{\text{risk exposure before reduction} - \text{risk exposure after reduction}}{\text{cost of reduction}}$$

UNIT 2
(PART-2)

REQUIREMENTS ANALYSIS AND SPECIFICATION

Experienced developers take considerable time to understand the exact requirements of the customer and to meticulously document those. They know that without a clear understanding of the problem and proper documentation of the same, it is impossible to develop a satisfactory solution.

For any type of software development project, availability of a good quality requirements document has been acknowledged to be a key factor in the successful completion of the project.

Good requirements document not only helps to form a clear understanding of various features required from the software, but also serves as the basis for various activities carried out during later life cycle phases

The goal of the requirements analysis and specification phase is to clearly understand the customer requirements and to systematically organise the requirements into a document called the Software Requirements Specification (SRS) document.

Who carries out requirements analysis and specification?

- Requirements analysis and specification activity is usually carried out by a few experienced members of the development team and it normally requires them to spend some time at the customer site.
- The engineers who gather and analyse customer requirements and then write the requirements specification document are known as system analysts in the software industry parlance.

The SRS document is the final outcome of the requirements analysis and specification phase.

How is the SRS document validated?

Once the SRS document is ready, it is first reviewed internally by the project team to ensure that it accurately captures all the user requirements, and that it is understandable, consistent, unambiguous, and complete.

The SRS document is then given to the customer for review. After the customer has reviewed the SRS document and agrees to it, it forms the basis for all future development activities and also serves as a contract document between the customer and the development organisation.

What are the main activities carried out during requirements analysis and specification phase?

Requirements analysis and specification phase mainly involves carrying out the following two important activities:

• Requirements gathering and analysis

• Requirements specification

REQUIREMENTS GATHERING AND ANALYSIS

The complete set of requirements are almost never available in the form of a single document from the customer.

In fact, it would be unrealistic to expect the customers to produce a comprehensive document containing a precise description of what he wants.

Further, the complete requirements are rarely obtainable from any single customer representative.

Therefore, the requirements have to be gathered by the analyst from several sources in bits and pieces.

We can conceptually divide the requirements gathering and analysis activity into two separate tasks:

• Requirements gathering

• Requirements analysis

Requirements Gathering

- Requirements gathering is also popularly known as requirements elicitation.
- The primary objective of the requirement gathering task is to collect the requirements from the stakeholders.

> A stakeholder is a source of the requirements and is usually a person, or a group of persons who either directly or indirectly are concerned with the software.

- Requirements gathering may sound like a simple task.
- However, in practice it is very difficult to gather all the necessary information from a large number of stakeholders and from information scattered across several pieces of documents.
- Gathering requirements turns out to be especially challenging if there is no working model of the software being developed.

In the following, we briefly discuss the important ways in which an experienced analyst gathers requirements:

1. **Studying existing documentation:** The analyst usually studies all the available documents regarding the system to be developed before visiting the customer site.
2. **Interview:** Typically, there are many differe nt categories of users of a software. Each category of users typically requires a different set of features from the software. Therefore, it is important for the analyst to first identify the different categories of users and then determine the requirements of each.
3. **Task analysis:** The users usually have a black-box view of a software and consider the software as something that provides a set of services (functionalities). A service supported by a software is also called a task.
4. **Scenario analysis:** A task can have many scenarios of operation. The different scenarios of a task may take place when the task is invoked under different situations.
5. **Form analysis:** Form analysis is an important and effective requirement gathering activity that is undertaken by the analyst, when the project involves automating an existing manual system (manually filled forms).

Requirements Analysis

- After requirements gathering is complete, the analyst analyses the gathered requirements to form a clear understanding of the exact customer requirements and to weed out any problems in the gathered requirements.
- It is natural to expect that the data collected from various stakeholders to contain several contradictions, ambiguities, and incompleteness, since each stakeholder typically has only a partial and incomplete view of the software.
- Therefore, it is necessary to identify all the problems in the requirements and resolve them through further discussions with the customer.

> The main purpose of the requirements analysis activity is to analyse the gathered requirements to remove all ambiguities, incompleteness, and inconsistencies from the gathered customer requirements and to obtain a clear understanding of the software to be developed.

The following basic questions pertaining to the project should be clearly understood by the analyst before carrying out analysis:

1. What is the problem?
2. Why is it important to solve the problem?
3. What exactly are the data input to the system and what exactly are the data output by the system?
4. What are the possible procedures that need to be followed to solve the problem?
5. What are the likely complexities that might arise while solving the problem?
6. If there are external software or hardware with which the developed software has to interface, then what should be the data interchange formats with the external systems?

During requirements analysis,the analyst needs to identify and resolve three main types of problems in the requirements:
- Anomaly
- Inconsistency
- Incompleteness

Anomaly: It is an anomaly is an ambiguity in a requirement. When a requirement is anomalous, several interpretations of that requirement are possible.

Inconsistency: Two requirements are said to be inconsistent, if one of the requirements contradicts the other.

Incompleteness: An incomplete set of requirements is one in which some requirements have been overlooked

SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

After the analyst has gathered all the required information regarding the software to be developed, and has removed all incompleteness, inconsistencies, and anomalies from the specification, he starts to systematically organise the requirements in the form of an SRS document

Among all the documents produced during a software development life cycle, SRS document is probably the most important document and is the toughest to write. One reason for this difficulty is that the SRS document is expected to cater to the needs of a wide variety of audience.

(i) Users of SRS Document

Usually, a large number of different people need the SRS document for very different purposes. Some of the important categories of users of the SRS document and their needs for use are as follows:

Users, customers, and marketing personnel: These stakeholders need to refer to the SRS document to ensure that the system as described in the document will meet their needs.

Software developers: The software developers refer to the SRS document to make sure that they are developing exactly what is required by the customer.

Test engineers: The test engineers use the SRS document to understand the functionalities, and based on this write the test cases to validate its working

User documentation writers: The user documentation writers need to read the SRS document to ensure that they understand the features of the product well enough to be able to write the users' manuals.
Project managers: The project managers refer to the SRS document to ensure that they can estimate the cost of the project easily by referring to the SRS document
Maintenance engineers: The SRS document helps the maintenance engineers to under- stand the functionalities supported by the system. A clear knowledge of the functionalities can help them to understand the design and code.

(ii) Uses of SRS Document

Forms an agreement between the customers and the developers: A good SRS document sets the stage for the customers to form their expectation about the software and the developers about what is expected from the software
Reduces future reworks: The process of preparation of the SRS document forces the stakeholders to rigorously think about all of the requirements before design and development get underway.
Provides a basis for estimating costs and schedules: Project managers usually estimate the size of the software from an analysis of the SRS document.
Provides a baseline for validation and verification: The SRS document provides a baseline against which compliance of the developed software can be checked. It is also used by the test engineers to create the test plan.
Facilitates future extensions: The SRS document usually serves as a basis for planning future enhancements.

(iii) Characteristics of a Good SRS Document

The skill of writing a good SRS document usually comes from the experience gained from writing SRS documents for many projects.
However, the analyst should be aware of the desirable qualities that every good SRS document should possess.
Some of the identified desirable qualities of an SRS document are the following:

- **Concise:** The SRS document should be concise and at the same time unambiguous, consistent, and complete.
- **Implementation-independent:** The SRS should be free of design and implementation decisions unless those decisions reflect actual requirements.
- **Traceable:** It should be possible to trace a specific requirement to the design elements that implement it and vice versa. Similarly, it should be possible to trace a requirement to the code segments that implement it and the test cases that test this requirement and vice versa.
- **Modifiable:** Customers frequently change the requirements during the software development due to a variety of reasons. Therefore, in practice the SRS document undergoes several revisions during software development.
- **Identification of response to undesired events:** The SRS document should discuss the system responses to various undesired events and exceptional conditions that may arise.
- **Verifiable:** All requirements of the system as documented in the SRS document should be verifiable.

Functional Requirements

In order to document the functional requirements of a system, it is necessary to first learn to identify the high-level functions of the systems by reading the informal documentation of the gathered requirements.

The high-level functional requirements often need to be identified either from an informal problem description document or from a conceptual understanding of the problem.

Each high-level requirement characterises a way of system usage (service invocation) by some user to perform some meaningful piece of work.

Non-functional requirements

Non-functional requirements are the criteria that define how a system should behave, rather than what it is supposed to do. Unlike functional requirements, which describe specific system functions, non-functional requirements define aspects like performance, security, usability, reliability, and scalability.

Functional Requirements	**Non-functional requirements**
Functional requirements help to understand the functions of the system.	They help to understand the system's performance.
Functional requirements are mandatory.	While non-functional requirements are not mandatory.
They are easy to define.	They are hard to define.
They describe what the product does.	They describe the working of product.
It concentrates on the user's requirement.	It concentrates on the expectation and experience of the user.
It helps us to verify the software's functionality.	It helps us to verify the software's performance.
These requirements are specified by the user.	These requirements are specified by the software developers, architects, and technical persons.
There is functional testing such as API testing, system, integration, etc.	There is non-functional testing such as usability, performance, stress, security, etc.
Examples of the functional requirements are - Authentication of a user on trying to log in to the system.	Examples of the non-functional requirements are - The background color of the screens should be light blue.
These requirements are important to system operation.	These are not always the important requirements, they may be desirable.

Completion of Functional requirements allows the system to perform, irrespective of meeting the non-functional requirements.	While system will not work only with non-functional requirements.