NARAYANA ENGINEERING COLLEGE::NELLORE		
Class: B. Tech	**Year-Semester:** II-II	**Year:** 2024-25
Course Title: SOFTWARE ENGINEERING	**Course Code: 23A05403**	**Credits:** 3
Program/Dept.: Computer Science and Engineering	**Section:** CSE- C	**Batch:** 2023-27
Regulation: NECR-23	**Faculty: Dr. Sunil Kumar Battarusetty**	

MODULE– 2		10H

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

A Software Project is the complete procedure of software development from requirement gathering to testing and maintenance, carried out according to the execution methodologies, in a specified period of time to achieve intended software product.

Software project management is an art and discipline of planning and supervising software projects. It is a sub-discipline of software project management in which software projects planned, implemented, monitored and controlled.
It is a procedure of managing, allocating and timing resources to develop computer software that fulfills requirements.

SOFTWARE PROJECT MANAGEMENT COMPLEXITIES

- Software project management complexities refer to the various challenges and difficulties involved in managing software development projects.
- The primary goal of software project management is to guide a team of developers to complete a project successfully within a given timeframe.
- However, this task is quite challenging due to several factors. Many projects have failed in the past due to poor project management practices.
- Software projects are often more complex to manage than other types of projects.

The main factors contributing to the complexity of managing a software project, as identified by are the following:

- Invisibility
- Changeability
- Complexity
- Uniqueness
- Exactness of the solution
- Team-oriented and intellect-intensive work

- **Invisibility:** Until the development of a software project is complete, Software remains invisible. Anything that is invisible, is difficult to manage and control.
  Software project managers cannot view the progress of the project due to the invisibility of the software until it is completely developed.
  The project manager can monitor the modules of the software that have been completed by the development team and the documents that have been prepared, which are rough indicators of the progress achieved. Thus invisibility causes a major problem in the complexity of managing a software project.
- **Changeability:** Requirements of a software product are undergone various changes. Most of these change requirements come from the customer during the software development.

Sometimes these change requests resulted in redoing of some work, which may cause various risks and increase expenses. Thus frequent changes to the requirements play a major role to make software project management complex.

- **Interaction/Complexity:** Even moderate-sized software has millions of parts (functions) that interact with each other in many ways such as data coupling, serial and concurrent runs, state transitions, control dependency, file sharing, etc.
  Due to the inherent complexity of the functioning of a software product in terms of the basic parts making up the software, many types of risks are associated with its development. This makes managing software projects much more difficult compared to many other kinds of projects.

- **Uniqueness:** Every software project is usually associated with many unique features or situations. This makes every software product much different from the other software projects.
  This is unlike the projects in other domains such as building construction, bridge construction, etc. where the projects are more predictable.
  Due to this uniqueness of the software projects, during the software development, a project manager faces many unknown problems that are quite dissimilar to other software projects that he had encountered in the past. As a result, a software project manager has to confront many unanticipated issues in almost every project that he manages.

- **The exactness of the Solution:** A small error can create a huge problem in a software project. The solution must be exact according to its design. The parameters of a function call in a program are required to be correct with the function definition. This requirement of exact conformity of the parameters of a function introduces additional risks and increases the complexity of managing software projects.

- **Team-oriented and Intellect-intensive work:** Software development projects are team-oriented and intellect-intensive work. The software cannot be developed without interaction between developers. In a software development project, the life cycle activities are not only intellect-intensive, but each member has to typically interact, review the work done by other members, and interface with several other team members creating various complexity to manage software projects.

- **The huge task regarding Estimation:** One of the most important aspects of software project management is Estimation. During project planning, a project manager has to estimate the cost of the project, the probable duration to complete the project, and how much effort is needed to complete the project based on size estimation. This estimation is a very complex task, which increases the complexity of software project management.

RESPONSIBILITIES OF A SOFTWARE PROJECT MANAGER:

a) Job responsibilities for managing software projects:

- A software project manager takes the overall responsibility of steering a project to success.
- Most managers takes the responsibilities of 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.
- The activities can be broadly classified into two major types
  1. project planning
  2. project monitoring and control.

1. **Project planning**: Project planning is done immediately after the feasibility

study 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.

2. **Project monitoring and control**: This is 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.

b) <u>Skills necessary for managing software projects</u>:

- ✓ Effective software project management calls for good qualitative judgment and decision taking capabilities.
- ✓ Also good grasp of latest software project management techniques like cost estimation, risk management and configuration management, good communication skills and the ability to get work done.
- ✓ Three skills that are most critical to successful project management are the following:
  1. Knowledge of project management techniques.
  2. Decision taking capabilities
  3. Previous experience in managing similar projects.

Features of a Good Project Manager

- ➢ Knowledge of project estimation techniques.
- ➢ Having good grasp of the latest SPM techniques such as cost estimation, risk management, and configuration management.
- ➢ Good decision-making abilities at the right time.
- ➢ Previous experience managing a similar type of projects.
- ➢ Good communication skills to meet the customer satisfaction.
- ➢ A project manager must encourage all the team members to successfully develop the product.
- ➢ He must know the various type of risks that may occur and the solution to these problems.
- ➢ Some skills like tracking and controlling the progress of the project, customer interaction, managerial presentations and team building are largely acquired through experience.

PROJECT PLANNING

- ➢ Once a project has been found to be feasible, software project managers undertake project planning.
- ➢ Project planning is undertaken and completed before any development activity starts.
- ➢ Project planning requires utmost care and attention as schedule delays can cause customer dissatisfaction. During project planning, the project manager performs the following activities:

a) Estimation: the following attributes are estimated.

- ✓ Cost: How much is it going to cost to develop the software product?
- ✓ Duration: How long is it going to take to develop the product?
- ✓ Effort: How much effort would be necessary to develop the product?

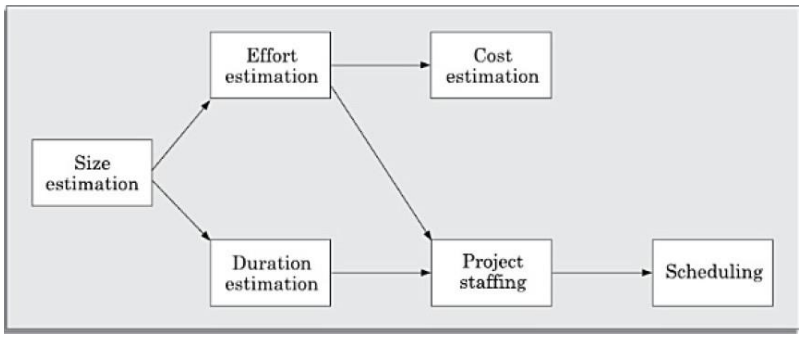
b) Scheduling: After all the project parameters are estimated, the schedules of manpower and other resources are developed.

c) Staffing: staff organization and staffing plans are made.

d) Risk Management: This includes risk identification, analysis and abatement planning.

e) Miscellaneous plans: Plans like quality assurance plan and configuration management plan etc.

Size is the most fundamental parameter based on which all other estimations and project plans are made. Figure shows the precedence ordering among planning activities.



Sliding window planning:

- ✓ Project managers undertake project planning over several stages.
- ✓ Planning a project over a number of stages protects managers from making big commitments at the start of the project.
- ✓ This technique of staggered planning is known as sliding window planning.
- ✓ In the sliding window planning technique starting with an initial plan, the project is planned more accurately over a number of stages.

SPMP Document of project planning

Introduction:

- Objectives
- Major Functions
- Performance Issues
- Management and Technical Constraints

Project Estimates:

- Historical Data Used
- Estimation Techniques Used
- Effort, Resource, Cost, and Project Duration Estimates

Schedule:

- Work Breakdown Structure
- Task Network Representation
- Gantt Chart Representation
- PERT Chart Representation

Project Resources:

- People
- Hardware and Software
- Special Resources

Staff Organization:

- Team Structure
- Management Reporting

Risk Management Plan:

- Risk Analysis
- Risk Identification
- Risk Estimation
- Risk Abatement Procedures

Project Tracking and Control Plan:

- Metrics to be tracked
- Tracking plan
- Control plan

Miscellaneous Plans:

- Process Tailoring
- Quality Assurance Plan

- Configuration Management Plan
- Validation and Verification
- System Testing Plan
- Delivery, Installation, and Maintenance Plan

METRICS FOR PROJECT SIZE ESTIMATION

In order to be able to accurately estimate the project size, some important metrics should be defined in terms of which the project size can be expressed.
Currently two metrics are popularly being used widely to estimate size:

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

LINES OF CODE (LOC)

- LOC is the simplest, very popular among all metrics available to estimate project size.
- Using this metric, the project size is estimated by counting the number of source instructions/lines in the developed program.
    - While counting the number of source instructions, lines used for commenting the code and the blank lines should be ignored.
- Determining the LOC count at the end of a project is a very simple job. But, at the beginning of a project is very difficult to accurate measure the LOC count.
- In order to estimate the LOC count at the beginning of a project, project managers usually divide the problem into modules and each module into sub modules and so on, until the sizes of the different leaf-level modules can be approximately predicted.
- To be able to do this, past experience in developing similar products is helpful.

Disadvantages of Using LOC

- Size can vary with coding style.
- Focuses on coding activity alone.
- Measures lexical / textual complexity only.
  → Does not address the issues of structural or logical complexity.
- Difficult to estimate LOC from problem description.

EXAMPLE OF LOC

```
1. /* Now how many lines of code is this? */
2. for (i=0; i < 100; i++)
3. {
4. printf ("hello");
5. }
```

Here, in this case the number of lines of program is 5 but the actual lines of the code is 4.

- LOC depends on the programming language chosen for the project.
- The exact number of the lines of code can only be determined after the project is completed.

FUNCTION POINT (FP)

- Function point metric was proposed by Albrecht [1983].
- This metric overcomes many of the shortcomings of the LOC metric.
- By using the function point metric we can easily estimate the size of a software product directly from the problem specification. .
- This is in contrast to the LOC metric, where the size can be accurately determined only after the product has fully been developed.

The FP can be computed in two steps:

- Unadjusted Function Point (UFP)
- Technical complexity Factor (TFC)

Once the UFP is computed, the TFC is computed next.

- FP=UFP+TFC

- Albrecht identified ***14 parameters*** that can influence the development effort.
- Each of these 14 parameters is assigned a value from **0** (not present or no influence) to **5** (strong influence).
- The resulting numbers are summed, yielding the total Degree of Influence (DI).
- A Technical Complexity Factor (TCF) for the project is computed.

  TCF is computed as (0.65+0.01*DI). Where $DI = 14 \times 4 = 56$

  As DI can vary from 0 to 84, TCF can vary from 0.65 to 1.49.
- Finally, **FP is given as the product of UFP and TCF**.

  That is, **FP=UFP*TCF.**

Ex: Given the following values, compute function point when all complexity adjustment factor (CAF) and weighting factors are ***average***.
Number of user inputs = 02
Number of user outputs = 03
Number of user enquiries = 01
Number of user files = 02
Number of external interfaces= 00

Step1: Calculate Unadjusted Function Point (UFP)
UFP = (no. of input)*4+(no. of outputs)*5+
(no. of enquiries)*4+(no. of files)*10+ (no. of interfaces)*10
UFP = (2*4) + (3*5) + (1*4) + (2*10) + (0*10) = 47
Step 2: Refine UFP
all the parameters are of moderate complexity, ***except output*** parameter, the complexity of the **output** parameter can be categorized as **simple.**
UFP = (2*5)+(3*4) + (1*4) + (2*10) + (0*10) = 46
Step 3: Since the complexity adjustment factors have ***average*** values, therefore the total degrees of influence would be:
DI = 14 × 4 = 56
TCF = 0.65 + 0.01 * DI = 0.65 + 0.01 * 56 = 1.21
Therefore, the adjusted
FP=UFP*TCF.
FP=46*1.21=55.66

Example : Functions points may compute the following important metrics:
Assume FP= 672
Average productivity is → 6.5 FP per person-month
(i.e. 1 person works for 1 month – to develop 6.5 FP)
Average labor cost is → Rs. 6000/- per month
1. Cost per FP = 6000/6.5 = Rs. 923 per function point
2. Total estimated project cost is
= Rs.923 * 672 FP = Rs. 6,20,256/-
3. Total estimated effort = (672 / 6.5) = 103 person – month.

i.e. 103 person will work for 1 month to complete the project.
Or 52 person will work for 2 months to complete the project.
Or 26 person will work for 4 months to complete the project.
Or 13 person will work for 8 months to complete the project.

PROJECT ESTIMATION TECHNIQUES

- The basic project planning activity is Estimation of various project parameters.
- The important project parameters are: project size, effort required to develop the software, project duration, and cost.
- These estimates not only help in quoting the project cost to the customer, but are also useful in resource planning and scheduling.

Project estimation techniques an broadly be classified into three main categories:

1. Empirical estimation techniques
2. Heuristic techniques and 3. Analytical estimation techniques

1. 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 formalized to a large extent.

Two popular empirical estimation techniques are:

Expert judgment technique and Delphi cost estimation

Expert Judgment Technique

- It is one of the most widely used estimation techniques.
- In this approach, an expert makes an educated guess of the problem size after analyzing the problem thoroughly.
- Usually, the expert estimates the cost of the different components (i.e. modules or subsystems) of the system and then combines them to arrive at the overall estimate.

Shortcomings

- The outcome of the expert judgment technique is subject to human errors and individual bias.
- Expert may overlook some factors inadvertently.
- Expert making an estimate may not have relevant experience and knowledge of all aspects of a project.
- Issues arising out of political or social considerations.
- Decision made by a group may be dominated by overly assertive members.

Delphi cost estimation

- This approach tries to overcome some of the shortcomings of the expert judgment approach.
- It is carried out by a team comprising of a group of experts and a coordinator.
- In this approach, the coordinator provides each estimator with a copy of the software requirements specification (SRS) document and a form for recording his cost estimate.
- Estimators complete their individual estimates anonymously and submit to the coordinator.
- In their estimates, the estimators mention any unusual characteristic of the product which has influenced his estimation.
- The coordinator prepares and distributes the summary of the responses of all the estimators, and includes any unusual rationale noted by any of the estimators. Based on this summary, the estimators re-estimate. This process is iterated for several rounds.
- However, no discussion among the estimators is allowed during the entire estimation process.
- The idea behind this is that if any discussion is allowed among the estimators, then many estimators may easily get influenced by the rationale of an estimator who may be more experienced or senior.
- After the completion of several iterations of estimations, the coordinator takes the responsibility of compiling the results and preparing the final estimate.

2. COCOMO- A HEURISTIC ESTIMATION TECHNIQUE

- This technique assumes that the relationships among the different project parameters can be modeled using suitable mathematical expressions.
- Once the basic (independent) parameters are known, the other (dependent) parameters can be easily determined by substituting the value of the basic parameters in the corresponding mathematical expression.
- Different heuristic estimation models can be divided into the following two classes:
  - ✓ Single variable model and
  - ✓ The multi variable model

COCOMO Model:

- The COnstructive COst MOdel (COCOMO) is an heuristic estimation model i.e., the model uses a theoretically derived formula to predict cost related factors. This model was created by "Barry Boehm".
- The COCOMO models are defined for three classes of software projects, stated as follows i.e. it divides software product developments into 3 categories:
  - ✓ **Organic**
  - ✓ **Semidetached**
  - ✓ **Embedded**

Organic: A development project can be considered of organic type, if
- ✓ The project deals with developing a well understood application program.
- ✓ The size of the development team is reasonably small.
- ✓ The team members are experienced in developing similar types of projects.

Semidetached: A development project can be considered of semidetached type, if
- ✓ The development consists of a mixture of experienced and inexperienced staff.
- ✓ Team members may have limited experience on related System.

Embedded: A development project is considered to be of embedded type, if
- ✓ The software being developed is strongly coupled to complex hardware.

The COCOMO model consists of three models(i.e. Software cost estimation is done through three stages):

- **Basic COCOMO,**
- **Intermediate COCOMO,**
- **Complete COCOMO.**

Basic COCOMO Model

- The **basic model** is used for ***quick and rough*** cost calculations for the software. It calculates the effort, time, and number of people required to use a project's kLOC (kilo lines of code). Gives only an approximate estimation. Effort estimation is obtained in terms of person-months.
- The basic COCOMO estimation model is given by the following expressions:

$$Effort(E) = a(kLOC)^b\ PM$$
$$\text{Time}(T) = c(E)^d$$
$$\text{People required} = E/T$$

Where

- KLOC is the estimated size of the software product expressed in Kilo Lines of Code,
- The effort is measured in person-months and time in months. The constants ***a,b,c***, and ***d*** vary for each model type.

- **Person-Month (PM)** is considered to be an appropriate unit for measuring effort, because developers are typically assigned to a project for a certain number of months.

The following are the constant values for the basic model:

Software Project	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Example

Suppose a project was estimated to be made in 400 kLOC.

Lets calculate its effort, time, and the number of people required while considering the project is of ***organic type***:

Effort(E)=$2.4(400\ \text{kLOC})^{1.05}$=1295.31 person−months

Time(T)=$2.5(1295.31)^{0.38}$=30.07 months

People required=1295.31/30.07=43.07 persons

Intermediate COCOMO model:

- The **intermediate model** is an extension of the basic model and includes a set of cost drivers to calculate the estimates with better accuracy.
- The intermediate COCOMO model refines initial estimate obtained by using the basic COCOMO.
- Uses a set of 15 cost drivers that are multiplied with the initial cost and effort estimated by the basic COCOMO.
- As per Boehm, the scale of cost drivers is in between 0 and 3.

In general, the cost drivers can be classified as being attributes of the following items:

Product

- Required software reliability extent
- Size of the application database
- The complexity of the product

Computer

- Run-time performance constraints
- Memory constraints
- The volatility of the virtual machine environment
- Required turnabout time

Personal

- Analyst capability
- Software engineering capability
- Applications experience
- Virtual machine experience
- Programming language experience

Development Environment

- Use of software tools
- Application of software engineering methods
- Required development schedule

Each of the 15 such attributes can be rated on a six-point scale ranging from “very low” to “extra high” in their relative order of importance. Each attribute has an effort multiplier fixed as per the rating. Table give below represents Cost Drivers and their respective rating:

The **Effort Adjustment Factor (EAF)** is determined by multiplying the effort multipliers associated with each of the 15 attributes.

The formulae to calculate these entities are:

$$\text{Effort}(E) = a(\text{kLOC})^{b} * \text{EAF} \quad \text{PM}$$

$$\text{Time}(T) = c(E)^{d}$$

Where,

- *E is effort applied in Person-Months*
- *KLOC is the estimated size of the software product indicate in Kilo Lines of Code*
- *EAF is the Effort Adjustment Factor (EAF) is a multiplier used to refine the effort estimate obtained from the basic COCOMO model.*
- *T is the development time in months and a, b, c, d are constants determined by the category of software project given in below table.*

Software Projects	a	b	c	d
Organic	**3.2**	**1.05**	**2.5**	**0.38**
Semi-Detached	**3.0**	**1.12**	**2.5**	**0.35**
Embedded	**2.8**	**1.20**	**2.5**	**0.32**

Example

Suppose a project was estimated to be made in 400 kLOC.let's calculate its effort, time, and the number of people required while considering the project is of organic type and has a nominal complexity. The developer has a high virtual machine experience.

The value of the nominal complexity of a project is 1.00, and the high virtual experience of the developer is 0.90, according to the tables mentioned above:

- **EAF =** It is an Effort Adjustment Factor, which is calculated by multiplying the parameter values of different cost driver parameters. For ideal, the value is 1.

$\text{EAF} = 1.00 * 0.90 = 0.9$

$\text{Effort}(E) = 3.2(400\ \text{kLOC})^{1.05} * 0.9 = 1554.37\ \text{person-months}$

$\text{Time}(T) = 2.5(1554.37)^{0.38} = 40.80\ \text{months}$

$\text{People required} = 1554.37/40.80 = 38.09\ \text{persons}$

Complete/Detailed COCOMO model:

- A major shortcoming of both the basic and intermediate COCOMO models is that they consider a software product as a single homogeneous entity.
- However, most large systems are made up several smaller sub-systems. These sub-systems may have widely different characteristics.
- For example, some sub-systems may be considered as organic type, some semidetached, and some embedded.
- Not only that the inherent development complexity of the subsystems.
- In the Detailed COCOMO Model, the cost of each subsystem is estimated separately.
- This approach reduces the margin of error in the final estimate.

Example:

A distributed Management Information System (MIS) product for an organization having offices at several places across the country can have the following sub-components:

- ***Database part***
- ***Graphical User Interface (GUI) part***
- ***Communication part***

Of these, the communication part can be considered as embedded software. The database part could be Semi-detached software, and the GUI part Organic software. The costs for these three components can be estimated separately, and summed up to give the overall cost of the system.

ANALYTICAL ESTIMATION TECHNIQUES

- It derives the required results starting with basic assumptions regarding the project.
- Thus, unlike empirical and heuristic techniques, analytical techniques do have scientific basis. Halstead's software science is an example of an analytical technique.
- Halstead's software science can be used to derive some interesting results starting with a few simple assumptions.

Halstead's Software Science – An Analytical Technique

- It is an analytical technique to measure size, development effort, and development cost of software products.
- Halstead used a few primitive program parameters to develop the expressions for overall program length, potential minimum value, actual volume, effort, and development time.
- For a given program, let:
  - η1 be the number of unique operators used in the program,
  - η 2 be the number of unique operands used in the program,
  - N1 be the total number of operators used in the program,
  - N2 be the total number of operands used in the program.

- It derived expressions for:
  - over all program length,
  - potential minimum volume
  - actual volume,
  - language level,
  - effort, and
  - Development time.
  - Staffing level estimation:

LENGTH AND VOCABULARY

- The **length** of a program as defined by Halstead, quantifies total usage of ***all operators and operands*** in the program i.e Total number of **tokens** used in the program.

Thus, **length N = N1 +N2.**

- The program **vocabulary** is the number of ***unique operators and operands*** used in the program.

Thus, program **vocabulary η = η1 + η2**

PROGRAM VOLUME

Program vocabulary and length depends on the programming style

Different lengths of programs, corresponding to the same problem

when different languages are used.

We need to express the program length by taking the programming language into consideration.

- ✓ ***Program volume(V) is the minimum number of bits needed to encode the program.***

$$V = N \log_2 \eta$$

To represent η different tokens we need $\log_2 \eta$ bits

ex: to represent 8 operands we need 3 bits.

For a program of N length we need $N \log_2 \eta$ bits.

Program volume represents the size of the program by approximately compensating the effort of the programming language used.

Program level (L):

This is the ratio of the number of operator occurrences to the number of operand occurrences in the program,

i.e., L = n1/n2

PROGRAMMING EFFORT and TIME

To obtain the needed effort, we divide the program volume (size) on the program level (complexity).

$$E = V / L$$

The programmer's time needed to finish the program

$$T = E / S$$

Where *S is the speed* of mental discriminations.

(recommended value of S is 18)

LENGTH ESTIMATION

Even though the ***length of a program*** can be found by ***calculating the total number of operators and operands*** in a program, Halstead suggests a way to determine the length of a program using the number of unique operators and operands used in the program.

Using this method, the **program parameters** such as ***length, volume, cost, effort,*** etc. can be determined even before the start of any programming activity. His method is summarized below

$$N = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$$

Example : Let us consider the following C program:

```
main( )
{
int a, b, c, avg;
scanf("%d %d %d", &a, &b, &c);
avg = (a+b+c)/3;
printf("avg = %d", avg);
}
```

The unique operators are: main,(),{},int,scanf,&,",",";",=,+,/, printf

The unique operands are: a, b, c, &a, &b, &c, a+b+c, avg, 3, "%d %d %d", "avg = %d"

Therefore, η1 = 12, η2 = 11

Estimated Length $N = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$

$= (12 * \log_2 12 + 11 * \log_2 11)$

$= (12*3.58 + 11*3.45)$

$= (43+38) = 81$

Volume = $N \log_2 \eta$ =Length*log(23) = 81*4.52 = 366.

RISK MANAGEMENT

Risk is an expectation of loss, a potential problem that may or may not occur in the future.

- It is generally caused due to lack of information, control or time.
- A possibility of suffering from loss in software development process is called a **software risk**.
- Loss can be anything, increase in production cost, development of poor quality software, not being able to complete the project on time.
- We cannot eliminate the Risk properly, but we can try to minimize it.

Risk Management refers to the systematic process of **identifying, evaluating, and addressing risks** throughout the lifecycle of a project.

- In software project management, risk refers to the potential events, circumstances, or factors that can negatively affect the project's objectives.
- These risks could impact cost, schedule, scope, or quality

a) Risk identification:

- The project manager needs to anticipate the risks in a project as early as possible.
- When risk is identified, effective risk management plans are made and the possible impacts of the risks is minimized.
- To identify the important risks it is necessary to categorize risks from each class are relevant to the project.
- Three main categories of risks include project risks, technical risks and business risks.

1. **Project risks**: includes budgetary, schedule, personnel, resource and customer related problems. Schedule slippage, software is intangible, difficult to monitor and control a software project. The invisibility of the product being developed is an important reason why many software projects suffer from the risk of schedule slippage.

2. **Technical risks**: includes potential design, implementation, interfacing, testing and maintenance problems. Ambiguous specification, incomplete specification, changing specification, technical uncertainty, development teams insufficient knowledge and technical obsolescence.

3. **Business risks**: Includes risk of building an excellent product that no one wants, losing budgetary commitments etc.

b) Risk Assessment:

- Once risks have been identified, they need to be assessed to understand their potential impact on the project. Risk assessment involves evaluating both the likelihood (***probability of occurrence***) and the impact (**severity of the consequences if the risk occurs**) of each identified risk.
- 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
  - ✓ The consequence of the problems associated with that risk.
- 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

c) Risk Mitigation/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.

There are three main methods to plan for risk management:

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.

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

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)

Calculating Risk Leverage:

Risk reduction amount:

$100,000 (initial risk) - $20,000 (reduced risk) = $80,000

Cost of reduction: $10,000 (cost of maintenance contracts)

Risk Leverage = ($80,000) / ($10,000) = 8

REQUIREMENT ANALYSIS AND SPECIFICATION

Requirements Analysis and Specification

- The goal of the requirements analysis and specification phase is to clearly understand the customer requirements and to systematically organize (remove inconsistencies, anomalies, etc. from requirements)the requirements into a document called the Software Requirements Specification (SRS) document.
- The SRS document is the final outcome of the requirements analysis and specification phase. The SRS document is reviewed by the customer.
- Requirements analysis and specification is considered to be a very important phase of software development and has to be undertaken with outmost care.
- It starts once the feasibility study phase is complete and the project is found to be financially sound and technically feasible.

  This phase consists of the following two activities:

  (i) Requirement gathering and analysis

  (ii) Requirements specification

Overview of requirements analysis and specification:

● The requirements analysis and specification phase starts after the feasibility study stage is complete and the project has been found to be financially viable and technically feasible. ●The requirements analysis and specification phase ends when the requirements specification document has been developed and reviewed.

●The requirements specification document is usually called the software requirements specification (SRS) document.

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

These activities are 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 analyze customer requirements and then write the requirements specification document are known as system analysts in the software industry.

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.

● Complete requirements are rarely obtainable from any single customer representative.

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

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

- Requirements gathering
- Requirements analysis

REQUIREMENTS GATHERING:

It is also known as Requirements Elicitation.

The primary objective of the requirements- 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.

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.

Important ways in which an experienced analyst gathers requirements:

- **Studying existing documentation:**
  - The analyst usually studies all the available documents regarding the system to be developed before visiting the customer site.
  - Customers usually provide a statement of purpose (SoP) document to the developers.
- **Interview:**
  - Typically, there are many different 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. Refer to: Delphi method

- **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 software is also called a task.
  - The analyst tries to identify and understand the different tasks to be performed by the software.
  - For each identified task, the analyst tries to formulate the different steps necessary to realize the required functionality in consultation with the users.
- **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.
  - For different types of scenarios of a task, the behavior of the software can be different.
- **Form analysis:**
  - Form analysis is an important and effective requirements gathering activity that is undertaken by the analyst, when the project involves automating an existing manual system.
  - Inform analysis the existing forms and the formats of the notifications produced are analyzed to determine the data input to the system and the data that are output from the system.

REQUIREMENT 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.

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

Anomaly:

■ An anomaly is an ambiguity in a requirement. When a requirement is anomalous, several interpretations of that requirement are possible.

Example: While gathering the requirements for a process control application, the following requirement was expressed by a certain stakeholder: "When the temperature becomes high, the heater should be switched off". Please note that words such as "high", "low", "good", "bad" etc. are indications of ambiguous requirements as these lack quantification and can be subjectively interpreted.

Inconsistency:

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

Example: Consider the following two requirements that were collected from two different stakeholders in a process control application development project.

- The furnace should be switched-off when the temperature of the furnace rises above 500°C.
- When the temperature of the furnace rises above 500°C, the water shower should be switched-on and the furnace should remain on.

Incompleteness:

■ An incomplete set of requirements is one in which some requirements have been overlooked. The lack of these features would be felt by the customer much later, possibly while using the software.

Example: In a chemical plant automation software, suppose one of the requirements is that if the internal temperature of the reactor exceeds 200°C then an alarm bell must be sounded. However, on an examination of all requirements, it was found that there is no provision for resetting the alarm bell after the temperature has been brought down in any of the requirements. This is clearly an incomplete requirement.

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 organize the requirements in the form of an SRS document.
- The SRS document usually contains all the user requirements in a structured though an informal form.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.
- A well-formulated SRS document finds a variety of usage:
    - Forms an agreement between the customers and the developers.
    - Reduces future reworks.
    - Provides a basis for estimating costs and schedules
    - Provides a baseline for validation and verification
    - Facilitates future extensions

Users of SRS document:

- **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 it is 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 and that it contains all the information required to plan the project.

- **Maintenance engineers**: The SRS document helps the maintenance engineers to understand the functionalities supported by the system.

Characteristics of a Good SRS Document:

● IEEE Recommended Practice for Software Requirements Specifications describes the content and qualities of a good software requirements specification (SRS).

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. It should only specify what the system should do and refrain from stating how to do these.

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. To cope up with the requirements changes, the SRS document should be easily modifiable. For this, an SRS document should be well-structured.

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. This means that it should be possible to design test cases based on the description of the functionality as to whether or not requirements have been met in an implementation.

Organization of SRS document:

1. Introduction
    - (a) Background
    - (b) Overall description
    - (c) Environmental characteristics
        - (i) Hardware
        - (ii) Peripherals
        - (iii) People
2. Goals of implementation
3. Functional requirements
4. Non-functional requirements
5. Behavioral Description
    - (a) System states
    - (b) Events and Actions

FORMAL SYSTEM SPECIFICATION

- A **Formal System Specification** is a detailed, mathematically rigorous description of a system's behavior, structure, and constraints.
- It is written using formal languages, which have well-defined syntax and semantics.
- These formal languages are used to precisely model and specify the desired properties and functionality of a system, ensuring unambiguous communication between stakeholders (e.g., developers, engineers, and clients) and facilitating automated verification, validation, and analysis.

Formal Technique:

- A **Formal Technique** refers to the use of mathematically-based methods to specify, develop, verify, and validate systems, software, or hardware.
- These techniques involve rigorous, formal languages and mathematical reasoning to ensure that systems behave correctly according to their specifications.
- Formal techniques are typically used in high-assurance domains such as aerospace, medical devices, safety-critical systems, and cryptography, where errors can have significant consequences.
- More precisely a formal specification language consists of two sets syn and sem and relation at between them.
- The set syn is called the syntactic domain, the set sem is called semantic domain and the relation set is called the satisfaction relation.

Model Versus Property Oriented Methods

The concepts of **model-oriented** and **property-oriented** methods are related to software specification, design, and analysis. These methods represent different approaches to understanding and building systems, and each has its strengths depending on the context.

Model-oriented methods focus on creating abstract representations (models) of a system. These models describe how the system should behave, including its components, structure, and interactions. The main goal is to build a comprehensive representation that can be analyzed, modified, or used as a blueprint for system development.

Property-oriented methods focus on defining the properties that a system should exhibit, rather than how it is structured or behaves in a detailed sense. The goal is to specify what needs to be true about the system, such as safety properties, performance requirements, or correctness constraints.

In a **model-oriented** style, we would start by defining the basic operations, p (produce) and c (consume). Then we can state that $S\ 1 + p \Rightarrow S$, $S + c \Rightarrow S\ 1$. Thus model-oriented approaches essentially specify a program by writing another, presumably simpler program. A few notable examples of popular model-oriented specification techniques are Z, CSP,CCS, etc.

In an **online payment system**, you might specify that "after a payment transaction starts, the transaction will eventually either succeed or fail." This ensures that no transaction can remain in an undefined state indefinitely.

Operational Semantics :

The operational semantics of a formal method constitute the way computations are represented. There are different type of operational semantics :

1. Linear Semantics : In this approach, run of a system is described by a sequence of events or states.

2. Branching Semantics : In this approach the behavior of a system is represented by a directed graph. The nodes of the graph represent the possible states in the evolution of a system

3. Maximum Parallel Semantics : In this approach, all the concurrent actions enabled at any state or assumed to be taken.

4. Partial Order Semantics : Under this view, the semantics described to a system constitute a structure of states satisfying a partial order relation among the states.

AXIOMATIC SPECIFICATION

- In axiomatic specification of a system, the first order logic is used to write the pre and post condition in order to specify the operations of the system in the form of axioms.
- The pre-conditions basically capture the conditions that must be satisfied before an operation can be successfully involved.
- In essence the pre-conditions capture the requirements on the input parameters of a function. The post-conditions are the condition that must be satisfied when a function to be considered to have executed successfully.
- Thus, the post condition essentially check the constraints on the result produced for the function execution to be consider successful.

Steps to develop an axiomatic specification :

1. Establish the range of input values over which the function should behave correctly. Establish the constraints on the input parameters as a predicate.
2. Specify a predicate defining the condition which must hold on the output of the function if it behaved properly.
3. Establish the changes made to the functions input parameters after execution of the function. Pure mathematical functions do not change their input and therefore this type of assertion is not necessary for pure functions.
4. Combines all of the above into pre and post condition of the functions.

Example : Specify the pre- and post-conditions of a function that takes a real number as argument and returns half the input value if the input is less than or equal to 100, or else returns double the value.

f (x : real) : real

pre : $x \in R$

post : $\{(x \leq 100) \wedge (f(x) = x/2)\} \vee \{(x > 100) \wedge (f(x) = 2 * x)\}$

ALGEBRAIC SPECIFICATION

- In the algebraic specification technique an object class or type is specified in terms of relationship existing between the operations defined on that type.
- Essentially, algebraic specification define a system as a heterogeneous algebra.
- A heterogeneous algebra is a collection of different sets on which several operational are defined.
- Traditional algebras are homogeneous.
- A homogeneous algebra consists of a single set and several operations (+, –, *, /). Each set of symbols in the algebra is called a sort of the algebra.

An algebraic specification is divided into four sections:

1. **Type Section**: In this section the sorts being used are specified.
2. **Exception section**: This section gives the names of the exceptional conditions that might occur when different operations are carried out.
3. **Syntax section:** This section defines the signature of the interface procedures. The collection of sets that form input domain of an operator and the sort where the output is produced are called the signature of the operator.
   For example, PUSH takes a stack and an element and returns a new stack
   Stack X element → stack
4. **Equations section:** This section given a set of rewrites rules (or equation) defining the meaning of the interface procedures in terms of each other.

Pros and Cons of Algebraic Specifications

- Algebraic specifications have a strong mathematical basis and can be viewed as heterogeneous algebra. Therefore, they are unambiguous and precise.
- Using an algebraic specification, the effect of any arbitrary sequence of operations involving the interface procedures can automatically be studied.
- A major shortcoming of algebraic specifications is that they cannot deal with side effects.
- Therefore, algebraic specifications are difficult to interchange with typical programming languages.
- Also, algebraic specifications are hard to understand.

Executable specification and 4GL

An Executable Specification: Executable specification and 4GL (Fourth-Generation Language) are concepts related to software development, and they aim to make the process more efficient, user-friendly, and high-level.

When the specification of a system is expressed in formal language, then it becomes possible to directly execute the specification to provide a system prototype, without having to design and write code for implementation.
However, executable specifications are usually slow and inefficient, **4GLs(4th Generation Languages)** are examples of executable specification languages.

4GLs(4th Generation Languages)

- A **4GL** is a programming language that is designed to be closer to human language than traditional programming languages (like Java or C).
- 4GLs typically focus on solving specific domains, like database management, business applications, or report generation.
- 4GLs are successful because there is a lot of commonality across data processing applications.
- 4GLs rely software reuse, where the common abstractions have been identified and parameterized.
- Rewriting 4GL programs in higher level languages(3GLs) results in up to 50% lower memory usage and also the program execution time can reduce up to 10 times faster.

Examples of 4GLs include:

- **SQL**: A domain-specific language for querying and manipulating relational databases.
- **MATLAB**: For mathematical computations and modeling.
- **COBOL**: Used in business, finance, and administrative systems.