UNIT 1 SOFTWARE PROJECT PLANNING

Structure **Page Nos.**

1.0 INTRODUCTION

Historically, software projects have dubious distinction of overshooting project schedule and cost. Estimating duration and cost continues to be a weak link in software project management. The aim of this unit is to give an overview of different project planning techniques and tools used by modern day software project managers.

It is the responsibility of the project manager to make as far as possible accurate estimations of effort and cost. This is particularly what is desired by the management of an organisation in a competitive world. This is specially true of projects subject to competition in the market where bidding too high compared with competitors would result in losing the business and a bidding too low could result in financial loss to the organisation. This makes software project estimation crucial for project managers.

1.1 OBJECTIVES

After going through this unit, you should be able to:

- understand different software estimation techniques;
- understand types of metrics used for software project estimation;
- learn about models used for software estimation, and
- learn about different automated tools used for estimation.

1.2 DIFFERENT TYPES OF PROJECT METRICS

It is said that if you cannot measure, then, it is not engineering. Effective management of software development process requires effective measurement of software development process. Often, from the input given by project leaders on the estimation of a software project, the management decides whether to proceed with the project or not.

The process of software estimation can be defined as the set of techniques and procedures that an organisation uses to arrive at an estimate. An important aspect of software projects is to know the cost, time, effort, size etc.

Need for Project metrics : Historically, the process of software development has been witnessing inaccurate estimations of schedule and cost, overshooting delivery target and productivity of software engineers in not commensurate with the growth of demand. Software development projects are quite complex and there was no scientific method of measuring the software process. Thus effective measurement of the process was virtually absent. The following phrase is aptly describing the need for measurement:

If you can not measure it, then, you can not improve it.

This is why measurement is very important to software projects. Without the process of measurement, software engineering cannot be called engineering in the true sense.

Definition of metrics : Metrics deal with measurement of the software process and the software product. Metrics quantify the characteristics of a process or a product. Merics are often used to estimate project cost and project schedule.

Examples of metrics : Lines of code(LOC), pages of documentation, number of man-months, number of test cases, number of input forms.

Types of Project metrics

Metrics can be broadly divided into two categories namely, product metrics and process metrics.

Product metrics provides a measure of a software product during the process of development which can be in terms of lines of code (either source code or object code), pages of documentation etc.

Process metrics is a measure of the software development process such as time, effort etc.

Another way of classification of metrics are primitive metrics and derived metrics.

Primitive metrics are directly observable quantities like lines of code (LOC), number of man-hours etc.

Derived metrics are derived from one or more of primitive metrics like lines of code per man-hour, errors per thousand lines of code.

Now, let us briefly discuss different types of product metrics and process metrics.

Product metrics

Lines of Code(LOC) : LOC metric is possibly the most extensively used for measurement of size of a program. The reason is that LOC can be precisely defined. LOC may include executable source code and non-executable program code like comments etc.

Looking at the following table, we can know the size of a module.

Module	**Effort in man-months**	**LOC**
Module 1	3	24,000
Module 2	4	25,000

Looking at the data above we have a direct measure of the size of the module in terms of LOC. We can derive a productivity metrics from the above primitive metrics i.e., LOC.

Productivity of a person = LOC / man-month

Quality = No. of defects / LOC

It is evident that the productivity of the developer engaged in Module 1 is more than the productivity of the developer engaged in Module 2. It is important to note here how derived metrics are very handy to project managers to measure various aspects of the projects.

Although, LOC provides a direct measure of program size, at the same time, these metrics are not universally accepted by project managers. Looking at the data in the table below, it can be easily observed that LOC is not an absolute measure of program size and largely depends on the computer language and tools used for development activity.

Consider the following table:

Module	**Effort in man-months**	**LOC in COBOL**	**LOC in Assembly**	**LOC in 4 GL**
Module- 1	3	24,000	800,000	400
Module- 2	4	25,000	100,000	500

The LOC of same module varies with the programming language used. Hence, just LOC cannot be an indicator of program size. The data given in the above table is only assumed and does not correspond to any module(s).

There are other attributes of software which are not directly reflected in Lines of Code (LOC), as the complexity of the program is not taken into account in LOC and it penalises well designed shorter program. Another disadvantage of LOC is that the project manager is supposed to estimate LOC before the analysis and design is complete.

Function point : Function point metrics instead of LOC measures the functionality of the program. Function point analysis was first developed by Allan J. Albrecht in the 1970s. It was one of the initiatives taken to overcome the problems associated with LOC.

In a Function point analysis, the following features are considered:

- **External inputs** : A process by which data crosses the boundary of the system. Data may be used to update one or more logical files. It may be noted that data here means either business or control information.

- **External outputs** : A process by which data crosses the boundary of the system to outside of the system. It can be a user report or a system log report.

- **External user inquires** : A count of the process in which both input and output results in data retrieval from the system. These are basically system inquiry processes.

- **Internal logical files** : A group of logically related data files that resides entirely within the boundary of the application software and is maintained

through external input as described above.

- **External interface files** : A group of logically related data files that are used by the system for reference purposes only. These data files remain completely outside the application boundary and are maintained by external applications.

As we see, function points, unlike LOC, measure a system from a functional perspective independent of technology, computer language and development method. The number of function points of a system will remain the same irrespective of the language and technology used.

For transactions like external input, external output and user inquiry, the ranking of high, low and medium will be based on number of file updated for external inputs or number of files referenced for external input and external inquiries. The complexity will also depend on the number of data elements.

Consider the following classification for external inquiry:

No. of Data elements	**Number of file references**		
	0 to 2	**3 to 4**	**5 and above**
1 to 5	Low	Low	Low
6 to 10	Low	Medium	Medium
10 to 20	Medium	Medium	High
More than 20	Medium	High	High

Also, External Inquiry, External Input and External output based on complexity can be assigned numerical values like rating.

Component type	**Values**		
	Low	**Medium**	**High**
External Output	4	6	8
External Input	2	4	6
External Inquiry	3	5	7

Similarly, external logical files and external interface files are assigned numerical values depending on element type and number of data elements.

Component type	**Values**		
	Low	**Medium**	**High**
External Logical files	6	8	10
External Interface	5	7	9

Organisations may develop their own strategy to assign values to various function points. Once the number of function points have been identified and their significance has been arrived at, the total function point can be calculated as follows.

Type of Component	**Complexity of component**			
	Low	**Medium**	**High**	**Total**
Number of External Output	X 4 =	X 6 =	X 8 =	
Number of External Input	X 2 =	X 4 =	X 6 =	
Number of External Inquiry	X3 =	X 5 =	X 7 =	
Number of Logical files	X 6 =	X 8 =	X 10 =	
Number of Interface file	X 5 =	X 7 =	X 9 =	
			Total of function point	

Total of function points is calculated based on the above table. Once, total of function points is calculated, other derived metrics can be calculated as follows:

Productivity of Person = Total of Function point / man-month
Quality = No. of defects / Total of Function points.

Benefits of Using Function Points

- Function points can be used to estimate the size of a software application correctly irrespective of technology, language and development methodology.
- User understands the basis on which the size of software is calculated as these are derived directly from user required functionalities.
- Function points can be used to track and monitor projects.
- Function points can be calculated at various stages of software development process and can be compared.

Other types of metrics used for various purposes are quality metrics which include the following:

- **Defect metrics** : It measures the number of defects in a software product. This may include the number of design changes required, number of errors detected in the test, etc.

- **Reliability metrics** : These metrics measure mean time to failure. This can be done by collecting data over a period of time.

☞ **Check Your Progress 1**

1) Lines of Code (LOC) is a product metric. True ☐ False ☐

2) _____ are examples of process metrics.

1.3 SOFTWARE PROJECT ESTIMATION

Software project estimation is the process of estimating various resources required for the completion of a project. Effective software project estimation is an important activity in any software development project. Underestimating software project and under staffing it often leads to low quality deliverables, and the project misses the target deadline leading to customer dissatisfaction and loss of credibility to the company. On the other hand, overstaffing a project without proper control will increase the cost of the project and reduce the competitiveness of the company.

Software project estimation mainly encompasses the following steps:

- Estimating the size of project. There are many procedures available for estimating the size of a project which are based on quantitative approaches like estimating Lines of Code or estimating the functionality requirements of the project called Function point.

- Estimating efforts based on man-month or man-hour. Man-month is an estimate of personal resources required for the project.

- Estimating schedule in calendar days/month/year based on total man-month required and manpower allocated to the project

  Duration in calendar month = Total man-months / Total manpower allocated.

- Estimating total cost of the project depending on the above and other resources.

In a commercial and competitive environment, Software project estimation is crucial for managerial decision making. The following *Table* give the relationship between various management functions and software metrics/indicators. Project estimation and tracking help to plan and predict future projects and provide baseline support for project management and supports decision making.

Consider the following table:

Activity	Tasks involved
Planning	Cost estimation, planning for training of manpower, project scheduling and budgeting the project
Controlling	Size metrics and schedule metrics help the manager to keep control of the project during execution
Monitoring/improving	Metrics are used to monitor progress of the project and wherever possible sufficient resources are allocated to improve.

Figure 1.1 depicts the software project estimation.

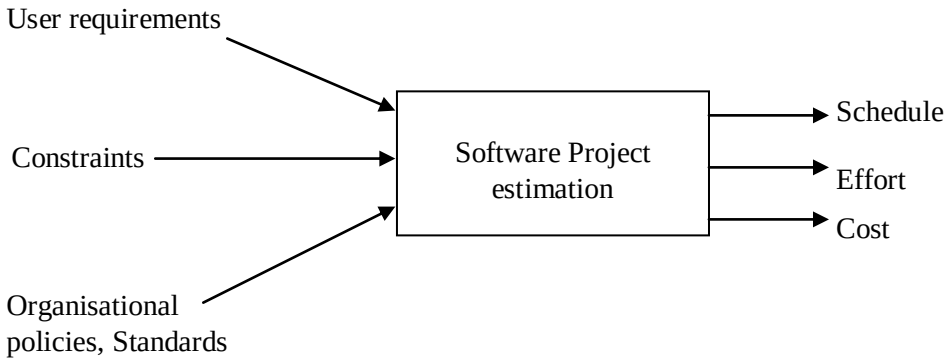

Figure 1.1: Software project estimation

1.3.1 Estimating the size

Estimating the size of the software to be developed is the very first step to make an effective estimation of the project. Customer's requirements and system specification forms a baseline for estimating the size of a software. At a later stage of the project, system design document can provide additional details for estimating the overall size of a software.

- The ways to estimate project size can be through past data from an earlier developed system. This is called estimation by analogy.
- The other way of estimation is through product feature/functionality. The system is divided into several subsystems depending on functionality, and size of each subsystem is calculated.

1.3.2 Estimating effort

Once the size of software is estimated, the next step is to estimate the effort based on the size. The estimation of effort can be made from the organisational specifics of software development life cycle. The development of any application software system is more than just coding of the system. Depending on deliverable requirements, the estimation of effort for project will vary. Efforts are estimated in number of man-months.

- The best way to estimate effort is based on the organisation's own historical data of development process. Organizations follow similar development life cycle for developing various applications.
- If the project is of a different nature which requires the organisation to adopt a different strategy for development then different models based on algorithmic approach can be devised to estimate effort.

1.3.3 Estimating Schedule

The next step in estimation process is estimating the project schedule from the effort estimated. The schedule for a project will generally depend on human resources involved in a process. Efforts in man-months are translated to calendar months.

Schedule estimation in calendar-month can be calculated using the following model [McConnell]:

Schedule in calendar months = 3.0 * (man-months) $^{1/3}$

The parameter 3.0 is variable, used depending on the situation which works best for the organisation.

1.3.4 Estimating Cost

Cost estimation is the next step for projects. The cost of a project is derived not only from the estimates of effort and size but from other parameters such as hardware, travel expenses, telecommunication costs, training cost etc. should also be taken into account.

Figure 1.2 depicts the cost estimation process.

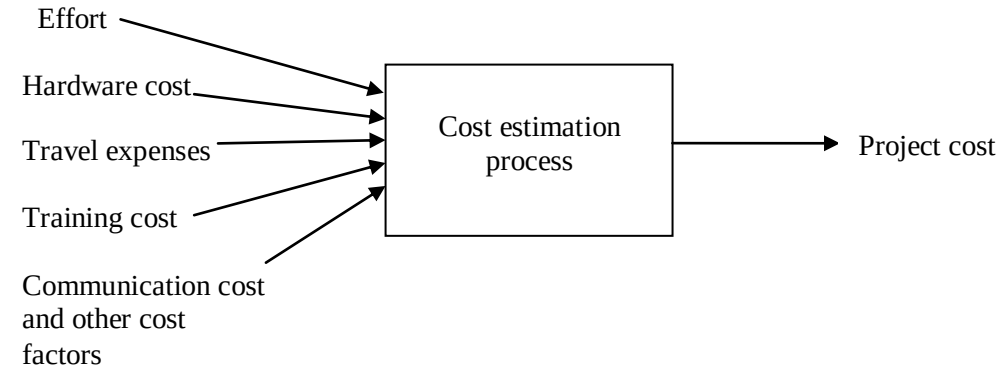

Figure 1.2 : Cost estimation process

Figure 1.3 depicts project estimation process.

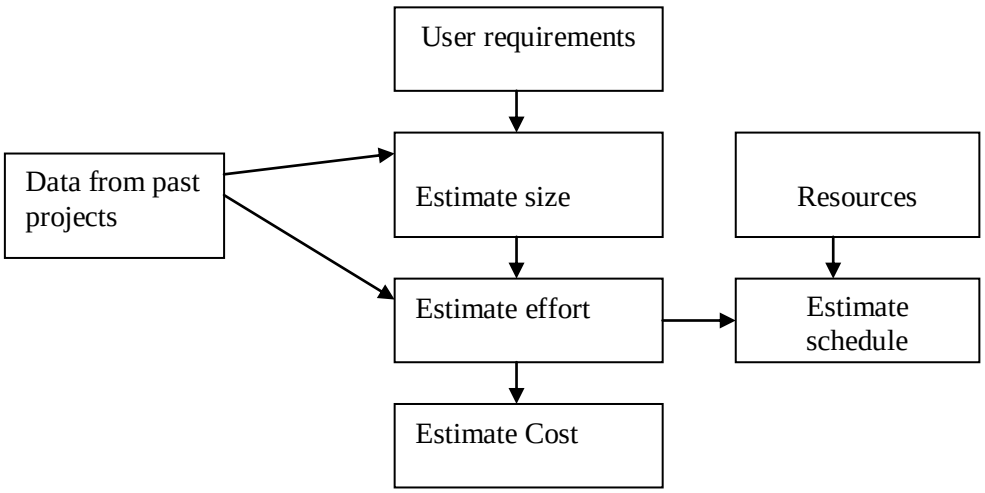

Figure 1.3: Project estimation process

Now, once the estimation is complete, we may be interested to know how accurate the estimates are to reality. The answer to this is "we do not know until the project is complete". There is always some uncertainty associated with all estimation techniques. The accuracy of project estimation will depend on the following:

- Accuracy of historical data used to project the estimation
- Accuracy of input data to various estimates
- Maturity of organisation's software development process.

The following are some of the reasons which make the task of cost estimation difficult:

- Software cost estimation requires a significant amount of effort. Sufficient time is not allocated for planning.
- Software cost estimation is often done hurriedly, without an appreciation for the actual effort required and is far from real issues.
- Lack of experience for developing estimates, especially for large projects.
- An estimator uses the extrapolation technique to estimate, ignoring the non-linear aspects of software development process

The following are some of the reasons for poor and inaccurate estimation:

- Requirements are imprecise. Also, requirements change frequently.
- The project is new and is different from past projects handled.
- Non-availability of enough information about past projects.
- Estimates are forced to be based on available resources.

Cost and time tradeoffs

If we elongate the project, we can reduce overall cost. Usually, long project durations are not liked by customers and managements. There is always shortest possible duration for a project, but it comes at a cost.

The following are some of the problems with estimates:

- Estimating size is often skipped and a schedule is estimated which is of more relevance to the management.
- Estimating size is perhaps the most difficult step which has a bearing on all other estimates.
- Let us not forget that even good estimates are only projections and subject to various risks.
- Organisations often give less importance to collection and analysis of historical data of past development projects. Historical data is the best input to estimate a new project.
- Project managers often underestimate the schedule because management and customers often hesitate to accept a prudent realistic schedule.

Project estimation guidelines

- Preserve and document data pertaining to organisation's past projects.
- Allow sufficient time for project estimation especially for bigger projects.
- Prepare realistic developer-based estimate. Associate people who will work on the project to reach at a realistic and more accurate estimate.
- Use software estimation tools.
- Re-estimate the project during the life cycle of development process.

- Analyse past mistakes in the estimation of projects.

☞ **Check Your Progress 2**

1) What is the first step in software project estimation?

……

2) What are the major inputs for software project estimation?

……

1.4 MODELS FOR ESTIMATION

Estimation based on models allows us to estimate projects ignoring less significant parameters and concentrating on crucial parameters that drive the project estimate. Models are analytic and empirical in nature. The estimation models are based on the following relationship:

$$E = f(v_i)$$

E = different project estimates like effort, cost, schedule etc.

v_i = directly observable parameter like LOC, function points

1.4.1 COCOMO Model

COCOMO stands for Constructive Cost Model. It was introduced by Barry Boehm. It is perhaps the best known and most thoroughly documented of all software cost estimation models. It provides the following three level of models:

- **Basic COCOMO** : A single-value model that computes software development cost as a function of estimate of LOC.

- **Intermediate COCOMO** : This model computes development cost and effort as a function of program size (LOC) and a set of cost drivers.

- **Detailed COCOMO** : This model computes development effort and cost which incorporates all characteristics of intermediate level with assessment of cost implication on each step of development (analysis, design, testing etc.).

This model may be applied to three classes of software projects as given below:

- Organic : Small size project. A simple software project where the development team has good experience of the application

- Semi-detached : An intermediate size project and project is based on rigid and semi-rigid requirements.

- Embedded : The project developed under hardware, software and operational constraints. Examples are embedded software, flight control software.

In the COCOMO model, the development effort equation assumes the following form:

$$E = aS^b m$$

where **a** and **b** are constraints that are determined for each model.

E = Effort

S = Value of source in LOC

m = multiplier that is determined from a set of 15 cost driver's attributes.

The following are few examples of the above cost drivers:

- Size of the application database
- Complexity of the project
- Reliability requirements for the software
- Performance constraints in run-time
- Capability of software engineer
- Schedule constraints.

Barry Boehm suggested that a detailed model would provide a cost estimate to the accuracy of ± 20 % of actual value

1.4.2 Putnam's model

L. H. Putnam developed a dynamic multivariate model of the software development process based on the assumption that distribution of effort over the life of software development is described by the Rayleigh-Norden curve.

$$P = Kt \exp(t^2/2T^2) / T^2$$

P = No. of persons on the project at time 't'
K = The area under Rayleigh curve which is equal to total life cycle effort
T = Development time

The Rayleigh-Norden curve is used to derive an equation that relates lines of code delivered to other parameters like development time and effort at any time during the project.

$$S = C_k K^{1/3} T^{4/3}$$

S = Number of delivered lines of source code (LOC)
C_k = State-of-technology constraints
K = The life cycle effort in man-years
T = Development time.

1.4.3 Statistical Model

From the data of a number of completed software projects, C.E. Walston and C.P. Felix developed a simple empirical model of software development effort with respect to number of lines of code. In this model, LOC is assumed to be directly related to development effort as given below:

$$E = a\ L^b$$

Where L = Number of Lines of Code (LOC)

E = total effort required

a and **b** are parameters obtained from regression analysis of data. The final equation is of the following form:

$$E = 5.2\ L^{0.91}$$

The productivity of programming effort can be calculated as

$$P = L/E$$

Where P = Productivity Index

1.4.4 Function Points

It may be noted that COCOMO, Putnam and statistical models are based on LOC. A number of estimation models are based on function points instead of LOC. However, there is very little published information on estimation models based on function points.

1.5 AUTOMATED TOOLS FOR ESTIMATION

After looking at the above models for software project estimation, we have reason to think of software that implements these models. This is what exactly the automated estimation tools do. These estimation tools, which estimate cost and effort, allow the project managers to perform "What if analysis". Estimation tools may only support size estimation or conversion of size to effort and schedule to cost.

There are more than dozens of estimation tools available. But, all of them have the following common characteristics:

- Quantitative estimates of project size (e.g., LOC).
- Estimates such as project schedule, cost.
- Most of them use different models for estimation.

Figure 1.4 depicts a typical structure of estimation tools.

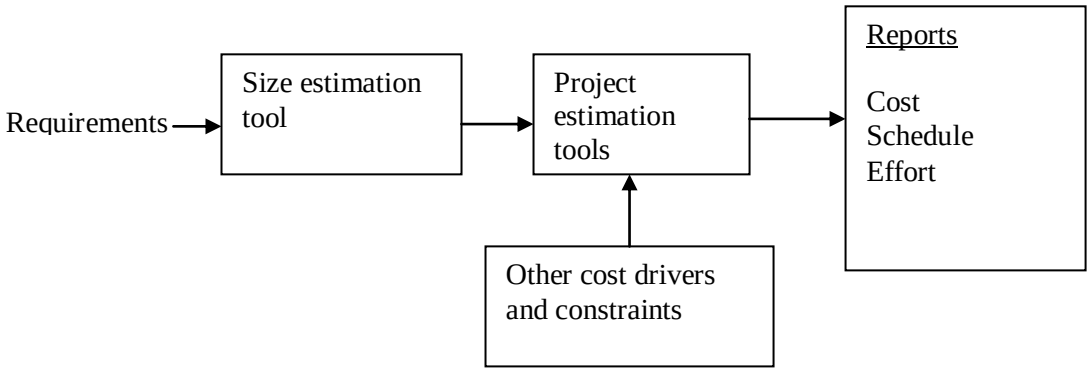

Figure 1.4: Typical structure of estimation tools

No estimation tool is the solution to all estimation problems. One must understand that the tools are just to help the estimation process.

Problems with Models

Most models require an estimate of software product size. However, software size is difficult to predict early in the development lifecycle. Many models use LOC for sizing, which is not measurable during requirements analysis or project planning. Although, function points and object points can be used earlier in the lifecycle, these measures are extremely subjective.

Size estimates can also be very inaccurate. Methods of estimation and data collection must be consistent to ensure an accurate prediction of product size. Unless the size

metrics used in the model are the same as those used in practice, the model will not yield accurate results (Fenton, 1997).

The following *Table* gives some estimation tools:

Automated Estimation Tools

Tool	Tool vendor site	Functionality/Remark
EstimatorPal	http://www.effortestimator.com/	EstimatorPal© is a software tool that assists software developers estimate the effort required to be spent on various activities. This tool facilitates use of the following Estimation techniques – Effort estimation tools which supports Function Point Analysis Technique., Objects Points Technique, Use Case Points Technique and Task-Based Estimation Technique
Estimate Easy Use Case	Duvessa Software http://www.duvessa.com/	Effort estimation tool based on use cases
USC COCOMO II	USC Center for Software Engineering http://sunset.usc.edu/research/COCOMOII/index.html	Based on COCOMO
ESTIMACS	Computer Associates International Inc. http://www.cai.com/products/estimacs.htm	Provides estimates of the effort, duration, cost and personnel requirements for maintenance and new application development projects.
Checkpoint	Software Productivity Research Inc. http://www.spr.com/	Guides the user through the development of a software project estimate and plan.
Function Point Workbench	Software Productivity Research Inc. http://www.spr.com/	Automated software estimation tools, implementing function point sizing techniques, linking project estimation to project management initiatives, and collecting historical project data to improve future estimation efforts
ESTIMATE Professional	Software Productivity Center http://www.spc.com	Based on Putnam, COCOMO II
SEER-SEM	Galorath http://www.galorath.com/	Predicts, measures and analyzes resources, staffing, schedules, risk and cost for software projects
ePM.Ensemble	InventX http://www.inventx.com/	Support effort estimation, among other things.
CostXpert	Marotz, Inc.	Based on COCOMO

Tool	Tool vendor site	Functionality/Remark
	http://www.costxpert.com/	

1.6 SUMMARY

Estimation is an integral part of the software development process and should not be taken lightly. A well planned and well estimated project is likely to be completed in time. Incomplete and inaccurate documentation may pose serious hurdles to the success of a software project during development and implementation. Software cost estimation is an important part of the software development process. Metrics are important tools to measure software product and process. Metrics are to be selected carefully so that they provide a measure for the intended process/product. Models are used to represent the relationship between effort and a primary cost factor such as software product size. Cost drivers are used to adjust the preliminary estimate provided by the primary cost factor. Models have been developed to predict software cost based on empirical data available, but many suffer from some common problems. The structure of most models is based on empirical results rather than theory. Models are often complex and rely heavily on size estimation. Despite problems, models are still important to the software development process. A model can be used most effectively to supplement and corroborate other methods of estimation.

1.7 SOLUTIONS/ANSWERS

Check Your Progress 1

1. True
2. Time, Schedule

Check Your Progress 2

1. Estimating size of the Project
2. User requirements, and project constraints.

1.8 FURTHER READINGS

1) *Software Engineering*, Ian Sommerville; *Sixth Edition, 2001*, Pearson Education.

2) *Software Engineering – A Practitioner's Approach*, Roger S. Pressman; McGraw-Hill International Edition.

Reference websites

http://www.rspa.com
http://www.ieee.org
http://www.ncst.ernet.in

UNIT 2 RISK MANAGEMENT AND PROJECT SCHEDULING

Structure **Page Nos.**

2.0 INTRODUCTION

As human beings, we would like life to be free from dangers, difficulties and any risks of any type. In case a risk arises, we would take proper measures to recover as soon as possible. Similarly, in software engineering, risk management plays an important role for successful deployment of the software product. Risk management involves monitoring of risks, taking necessary actions in case risk arises by applying risk recovery methods.

2.1 OBJECTIVES

After going through this unit, you should be able to:

- know the meaning of risk;
- identify risks; and
- manage the risks

2.2 IDENTIFICATION OF SOFTWARE RISKS

A risk may be defined as a potential problem. It may or may not occur. But, it should always be assumed that it may occur and necessary steps are to be taken.
Risks can arise from various factors like improper technical knowledge or lack of communication between team members, lack of knowledge about software products, market status, hardware resources, competing software companies, etc.

Basis for Different Types of Software risks

- **Skills or Knowledge**: The persons involved in activities of problem analysis, design, coding and testing have to be fully aware of the activities and various

techniques at each phase of the software development cycle. In case, they have partial knowledge or lacks adequate skill, the products may face many risks at the current stage of development or at later stages.

- **Interface modules**: Complete software contains various modules and each module sends and receives information to other modules and their concerned data types have to match.

- **Poor knowledge of tools**: If the team or individual members have poor knowledge of tools used in the software product, then the final product will have many risks, since it is not thoroughly tested.

- **Programming Skills:** The code developed has to be efficient, thereby, occupying less memory space and less CPU cycles to compute given task. The software product should be able to implement various object oriented techniques and be able to catch exceptions in case of errors. Various data values have to be checked and in case of improper values, appropriate messages have to be displayed. If this is not done, then it leads to risk, thereby creating panic in the software computations.

- **Management Issues** : The management of the organisation should give proper training to the project staff, arrange some recreation activities, give bonus and promotions and interact with all members of the project and try to solve their necessities at the best. It should take care that team members and the project manager have healthy coordination, and in case there are some differences they should solve or make minor shuffles.

- **Updates in the hardware resources:** The team should be aware of the latest updates in the hardware resources, such as latest CPU (Intel P4, Motorola series, etc.), peripherals, etc. In case the developer makes a product, and later in the market, a new product is released, the product should support minimum features. Otherwise, it is considered a risk, and may lead to the failure of the project.

- **Extra support:** The software should be able to support a set of a few extra features in the vicinity of the product to be developed.

- **Customer Risks:** Customer should have proper knowledge of the product needed, and should not be in a hurry to get the work done. He should take care that all the features are implemented and tested. He should take the help of a few external personnel as needed to test the product and should arrange for demonstrations with a set of technical and managerial persons from his office.

- **External Risks:** The software should have backup in CD, tapes, etc., fully encrypted with full licence facilities. The software can be stored at various important locations to avoid any external calamities like floods, earthquakes, etc. Encryption is maintained such that no external persons from the team can tap the source code.

- **Commercial Risks:** The organisation should be aware of various competing vendors in the market and various risks involved if their product is not delivered on time. They should have statistics of projects and risks involved from their previous experience and should have skilled personnel.

2.3 MONITORING OF RISKS

Various risks are identified and a risk monitor table with attributes like risk name, module name, team members involved, lines of code, codes affecting this risk, hardware resources, etc., is maintained. If the project is continued further to 2-3 weeks, and then further the risk table is also updated. It is seen whether there is a ripple effect in the table, due to the continuity of old risks. Risk monitors can change the ordering of risks to make the table easy for computation. *Table 2.1* depicts a risk table monitor. It depicts the risks that are being monitored.

Table 2.1: Risk table monitor

Sl. No.	Risk Name	Week 1	Week 2	Remarks
1.	Module compute()	Line 5, 8, 20	Line 5,25	Priority 3
2.	More memory and peripherals	Module f1(), f5() affected	Module f2() affected	Priority 1
.....	.....	.....	.....	.....

The above risk table monitor has a risk in module compute () where there is a risk in line 5, 8 and 20 in week 1. In week 2, risks are present in lines 5 and 25. Risks are reduced in week 2. The priority 3 is set. Similarly, in the second row, risk is due to more memory and peripherals, affecting module f1 (), f5 () in week-1. After some modifications in week 2, module f2 () is affected and the priority is set to 1.

☞ Check Your Progress 1

1) Define the term risk and how it is related to software engineering.

……

……

2) List at least two risks involved with the management of team members.

……

……

3) What are commercial risks?

……

……

4) What do you mean by monitoring of risks and describe risk table.

……

……

5) Mention any two ways to prioritise risks.

……

……

2.4 MANAGEMENT OF RISKS

Risk management plays an important role in ensuring that the software product is error free. Firstly, risk management takes care that the risk is avoided, and if it not avoidable, then the risk is detected, controlled and finally recovered.

The flow of risk management is as follows:

2.4.1 Risk Management

A priority is given to risk and the highest priority risk is handled first. Various factors of the risk are who are the involved team members, what hardware and software items are needed, where, when and why are resolved during risk management. The risk manager does scheduling of risks. Risk management can be further categorised as follows:

1. Risk Avoidance
   a. Risk anticipation
   b. Risk tools
2. Risk Detection
   a. Risk analysis
   b. Risk category
   c. Risk prioritisation
3. Risk Control
   a. Risk pending
   b. Risk resolution
   c. Risk not solvable
4. Risk Recovery
   a. Full
   b. Partial
   c. Extra/alternate features

Figure 2.1 depicts a risk manager tool.

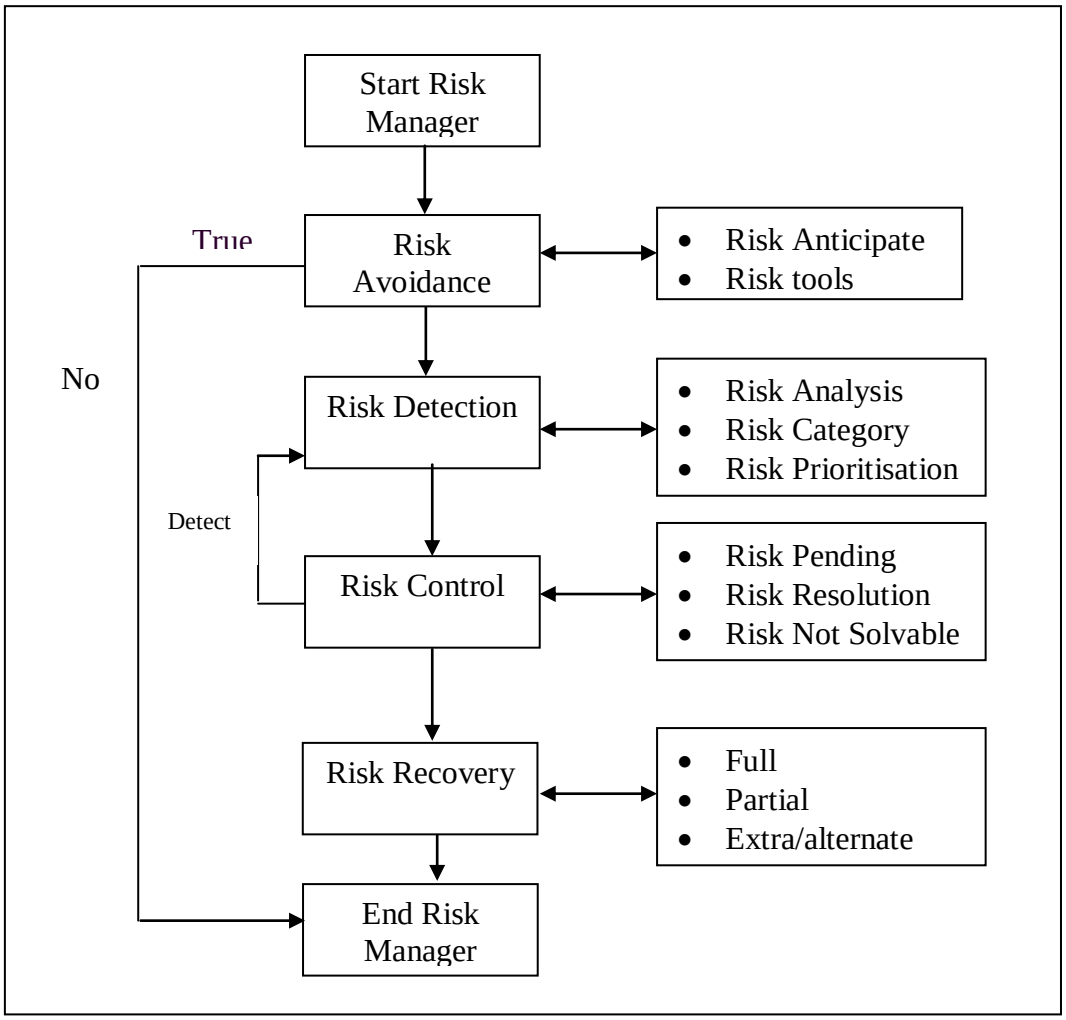

Figure 2.1 : Risk Manager Tool

From the *Figure 2.1*, it is clear that the first phase is to avoid risk by anticipating and using tools from previous project history. In case there is no risk, risk manager halts. In case there is risk, detection is done using various risk analysis techniques and further prioritising risks. In the next phase, risk is controlled by pending risks, resolving risks and in the worst case (if risk is not solved) lowering the priority. Lastly, risk recovery is done fully, partially or an alternate solution is found.

2.4.2 Risk Avoidance

Risk Anticipation: Various risk anticipation rules are listed according to standards from previous projects' experience, and also as mentioned by the project manager.

Risk tools: Risk tools are used to test whether the software is risk free. The tools have built-in data base of available risk areas and can be updated depending upon the type of project.

2.4.3 Risk Detection

The risk detection algorithm detects a risk and it can be categorically stated as :

Risk Analysis: In this phase, the risk is analyzed with various hardware and software parameters as probabilistic occurrence (*pr*), weight factor (*wf)* (hardware resources, lines of code, persons), risk exposure (*pr * wf*).

Table 2.1 depicts a risk analysis table.

Table 2.1: Risk analysis table

Sl.No.	Risk Name	Probability of occurrence (pr)	Weight factor (wf)	Risk exposure (pr * wf)
1.	Stack overflow	5	15	75
2.	No Password forgot option	7	20	140
....	.....	.....	.....	.....

Maximum value of risk exposure indicates that the problem has to solved as soon as possible and be given high priority. A risk analysis table is maintained as shown above.

Risk Category: Risk identification can be from various factors like persons involved in the team, management issues, customer specification and feedback, environment, commercial, technology, etc. Once proper category is identified, priority is given depending upon the urgency of the product.

Risk Prioritisation: Depending upon the entries of the risk analysis table, the maximum risk exposure is given high priority and has to be solved first.

2.5 RISK CONTROL

Once the prioritisation is done, the next step is to control various risks as follows:

- **Risk Pending***:* According to the priority, low priority risks are pushed at the end of the queue with a view of various resources (hardware, man power, software) and in case it takes more time their priority is made higher.

- **Risk Resolution**: Risk manager makes a strong resolve how to solve the risk.
- **Risk elimination**: This action leads to serious error in software.
- **Risk transfer***:* If the risk is transferred to some part of the module, then risk analysis table entries get modified. Thereby, again risk manager will control high priority risk.
- **Disclosures***:* Announce the risk of less priority to the customer or display message box as a warning. And thereby the risk is left out to the user, such that he should take proper steps during data entry, etc.
- **Risk not solvable***:* If a risk takes more time and more resources, then it is dealt in its totality in the business aspect of the organisation and thereby it is notified to the customer, and the team member proposes an alternate solution. There is a slight variation in the customer specification after consultation.

2.6 RISK RECOVERY

Full : The risk analysis table is scanned and if the risk is fully solved, then corresponding entry is deleted from the table.

Partial : The risk analysis table is scanned and due to partially solved risks, the entries in the table are updated and thereby priorities are also updated.

Extra/alternate features : Sometimes it is difficult to remove some risks, and in that case, we can add a few extra features, which solves the problem. Therefore, a bit of coding is done to get away from the risk. This is later documented or notified to the customer.

☞ Check Your Progress 2

1) Define the term risk management

……

……

2) What are various phases of risk manager?

……

……

3) What are the attributes mentioned in risk analysis table?

……

……

4) What is meant by risk resolution?

……

……

5) Why do we add extra features to recover from risks?

……

……

2.7 FORMULATING A TASK SET FOR THE PROJECT

The objective of this section is to get an insight into project scheduling by defining various task sets dependent on the project and choosing proper tasks for software engineering.

Various static and dynamic scheduling methods are also discussed for proper implementation of the project.

Factors affecting the task set for the project

- **Technical staff expertise**: All staff members should have sufficient technical expertise for timely implementation of the project. Meetings have to be conducted, weekly and status reports are to be generated.
- **Customer satisfaction** : Customer has to be given timely information regarding the status of the project. If not, there might be a communication gap between the customer and the organisation.
- **Technology update** : Latest tools and existing tested modules have to be used for fast and efficient implementation of the project.
- **Full or partial implementation of the project** : In case, the project is very large and to meet the market requirements, the organisation has to satisfy the customer with at least a few modules. The remaining modules can be delivered at a later stage.
- **Time allocation** : The project has to be divided into various phases and time for each phase has to be given in terms of person-months, module-months, etc.
- **Module binding** : Module has to bind to various technical staff for design, implementation and testing phases. Their necessary inter-dependencies have to be mentioned in a flow chart.
- **Milestones** : The outcome for each phase has to be mentioned in terms of quality, specifications implemented, limitations of the module and latest updates that can be implemented (according to the market strategy).
- **Validation and Verification** : The number of modules verified according to customer specification and the number of modules validated according to customer's expectations are to be specified.

2.8 CHOOSING THE TASKS OF SOFTWARE ENGINEERING

Once the task set has been defined, the next step is to choose the tasks for software project. Depending upon the software process model like linear sequential, iterative, evolutionary model etc., the corresponding task is selected. From the above task set, let us consider how to choose tasks for project development (as an example) as follows:

- **Scope** : Overall scope of the project.

- **Scheduling and planning** : Scheduling of various modules and their milestones, preparation of weekly reports, etc.
- **Technology used :** Latest hardware and software used.
- **Customer interaction** : Obtaining feedback from the customer.
- **Constraints and limitations** : Various constraints in the project and how they can be solved. Limitations in the modules and how they can be implemented in the next phase of the project, etc.
- **Risk Assessment** : Risk involved in the project with respect to limitations in the technology and resources.

2.9 SCHEDULING METHODS

Scheduling of a software project can be correlated to prioritising various tasks (jobs) with respect to their cost, time and duration. Scheduling can be done with resource constraint or time constraint in mind. Depending upon the project, scheduling methods can be static or dynamic in implementation.

Scheduling Techniques

The following are various types of scheduling techniques in software engineering are:

- **Work Breakdown Structure** : The project is scheduled in various phases following a bottom-up or top-down approach. A tree-like structure is followed without any loops. At each phase or step, milestone and deliverables are mentioned with respect to requirements. The work breakdown structure shows the overall breakup flow of the project and does not indicate any parallel flow. *Figure 2.2* depicts an example of a work breakdown structure.

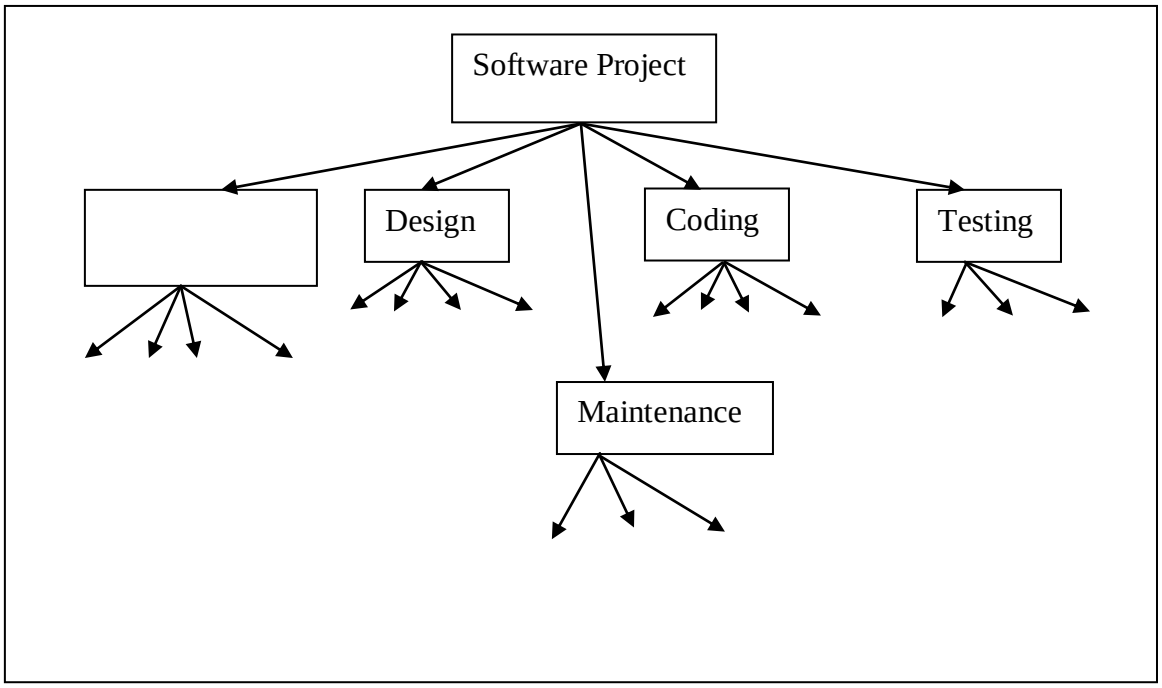

Figure 2.2: An example WBS

The project is split into requirement and analysis, design, coding, testing and maintenance phase. Further, requirement and analysis is divided into R1,R2 .. Rn; design is divided into D1,D2..Dn; coding is divided into C1,C2..Cn; testing is divided into T1,T2.. Tn; and maintenance is divided into M1, M2.. Mn. If the project

is complex, then further sub division is done. Upon the completion of each stage, integration is done.

- **Flow Graph :** Various modules are represented as nodes with edges connecting nodes. Dependency between nodes is shown by flow of data between nodes. Nodes indicate milestones and deliverables with the corresponding module implemented. Cycles are not allowed in the graph. Start and end nodes indicate the source and terminating nodes of the flow. *Figure 2.3* depicts a flow graph.

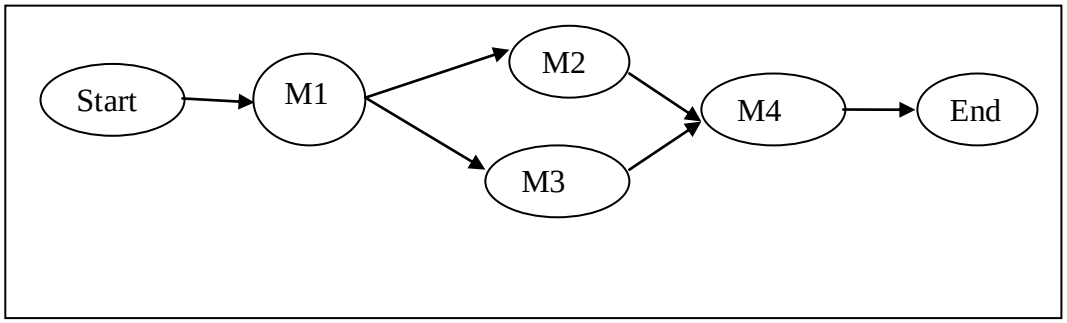

Figure 2.3 : Flow Graph

M1 is the starting module and the data flows to M2 and M3. The combined data from M2 and M3 flow to M4 and finally the project terminates. In certain projects, time schedule is also associated with each module. The arrows indicate the flow of information between modules.

- **Gantt Chart or Time Line Charts :** A Gantt chart can be developed for the entire project or a separate chart can be developed for each function. A tabular form is maintained where rows indicate the tasks with milestones and columns indicate duration (weeks/months) . The horizontal bars that spans across columns indicate duration of the task. *Figure 2.4* depicts a Gantt Chart. The circles indicate the milestones.

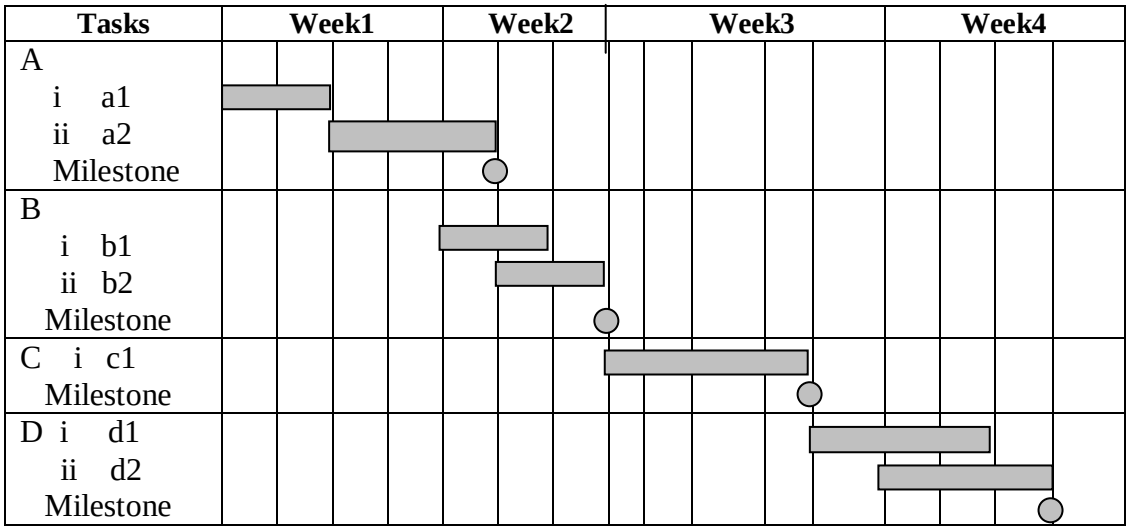

Figure 2.4: Gantt Chart

- **Program Evaluation Review Technique :** Mainly used for high-risk projects with various estimation parameters. For each module in a project, duration is estimated as follows:

  1. Time taken to complete a project or module under normal conditions, *tnormal.*

2. Time taken to complete a project or module with minimum time (all resources available), *tmin* .

3. Time taken to complete a project or module with maximum time (resource constraints), *tmax.*

4. Time taken to complete a project from previous related history, *Thistory.*

An average of tnormal, tmin, tmax and thistory is taken depending upon the project. Sometimes, various weights are added as 4*tnormal, 5*tmin, 0.9*tmax and 2*thistory to estimate the time for a project or module. Parameter fixing is done by the project manager.

2.10 THE SOFTWARE PROJECT PLAN

Planning is very important in every aspect of development work. Good managers carefully monitor developments at various phases. Improper planning leads to failure of the project. Software project plan can be viewed as the following :

1. Within the organisation: How the project is to be implemented? What are various constraints(time, cost, staff) ? What is market strategy?

2. With respect to the customer: Weekly or timely meetings with the customer with presentations on status reports. Customer feedback is also taken and further modifications and developments are done. Project milestones and deliverables are also presented to the customer.

For a successful software project, the following steps can be followed:

- Select a project
  - Identifying project's aims and objectives
  - Understanding requirements and specification
  - Methods of analysis, design and implementation
  - Testing techniques
  - Documentation

- Project milestones and deliverables

- Budget allocation
  - Exceeding limits within control

- Project Estimates
  - Cost
  - Time
  - Size of code
  - Duration

- Resource Allocation
  - Hardware
  - Software
  - Previous relevant project information
  - Digital Library

- Risk Management
  - Risk Avoidance
  - Risk Detection

- Risk Control
- Risk Recovery

- Scheduling techniques
  - Work Breakdown Structure
  - Activity Graph
  - Critical path method
  - Gantt Chart
  - Program Evaluation Review Technique

- People
  - Staff Recruitment
  - Team management
  - Customer interaction

- Quality control and standard

All of the above methods/techniques are not covered in this unit. The student is advised to study references for necessary information.

☞ **Check Your Progress 3**

1) Mention at least two factors to formulate a task set for a software project.

……

……

2) What are the drawbacks of work breakdown structure?

……

……

3) What are the advantages of Gantt chart?

……

……..

4) What is the purpose of software project plan?

……

……

2.11 SUMMARY

This unit describes various risk management and risk monitoring techniques. In case, major risks are identified, they are resolved and finally risk recovery is done. Risk manager takes care of all the phases of risk management. Various task sets are defined for a project from the customer point of view, the developer's point of view, the market strategy view, future trends, etc. For the implementation of a project, a proper task set is chosen and various attributes are defined. For successful implementation of a project, proper scheduling (with various techniques) and proper planning are done.

2.12 SOLUTIONS/ANSWERS

Check Your Progress 1

1) Any problem that occurs during customer specification, design, coding, implementation and testing can be termed as a risk. If they are ignored, then they propagate further down and it is termed ripple effect. Risk management deals with avoidance and detection of risk at every phase of the software development cycle.

2) Two risks involved with team members are as follows:

   - Improper training of the technical staff.
   - Lack of proper communication between the developers.

3) The organisation should be aware of various competing vendors in the market, and various risks involved if the product is not delivered on time. It should have statistics of projects and risks involved from their previous experience or should have expertise personnel. The organisation should take care that the product will be able to support upcoming changes in hardware and software platforms.

4) Monitoring of risks means identifying problems in software functions, hardware deficiencies (lack of memory , peripherals, fast cpu and so on), etc. Risk table has entries for all the risk types and their timely weekly solution . Priorities of various risks are maintained.

5) Risks can be prioritised upon their dependencies on other modules and external factors. If a module is having many dependencies then its priority is given higher value compared to independent modules. If a module often causes security failure in the system, its priority can be set to a higher value.

Check Your Progress 2

1) Risk management means taking preventive measures for a software project to be free from various types of risks such as technical, customer, commercial, etc.

2) Various phases of risk management are risk avoidance, risk detection, risk analysis, risk monitoring, risk control and risk recovery.

3) Attributes mentioned in the risk analysis table are risk name, probability of occurrence of risk, weight factor and risk exposure.

4) Risk resolution means taking final steps to free the module or system from risk. Risk resolution involves risk elimination, risk transfer and disclosure of risk to the customer.

5) Some times, it is difficult to recover from the risk and it is better to add extra features or an alternate solutions keeping in view of customer specification with slight modifications in order to match future trends in hardware and software markets.

Check Your Progress 3

1) The two factors to formulate a task set for a software project are as follows:

   - Customer satisfaction
   - Full or partial implementation of the project

2) Work breakdown structure does not allow parallel flow design.

3) Gantt chart or time line chart indicates timely approach and milestone for each task and their relevant sub tasks.

4) Software project plan indicates scope of the project, milestones and deliverables, project estimates, resource allocation, risk management, scheduling techniques and quality control and standard .

2.13 FURTHER READINGS

1) *Software Engineering*, Ian Sommerville; *Sixth Edition, 2001*, Pearson Education.

2) *Software Engineering – A Practitioner's Approach*, Roger S. Pressman; McGraw-Hill International Edition.

Reference websites

http://www.rspa.com
http://www.ieee.org
http://www.ncst.ernet.in

UNIT 3 SOFTWARE QUALITY ASSURANCE

Structure **Page Nos.**

3.0 INTRODUCTION

Software quality assurance is a series of activities undertaken throughout the software engineering process. It comprises the entire gamut of software engineering life cycle. The objective of the software quality assurance process is to produce high quality software that meets customer requirements through a process of applying various procedures and standards.

3. 1 OBJECTIVES

The purpose of this unit is to give an insight as to how software quality assurance activity is undertaken in the software engineering process.

After studying this unit, you should be able to understand the:

- concepts of Software quality and quality assurance;
- process of a formal technical review in a quality assurance process, and
- concepts of software reliability and software quality standards.

3.2 SOFTWARE QUALITY

Quality is defined as conformance to the stated and implied needs of customer. Quality also refers to the measurable characteristics of a software product and these can be compared based on a given set of standards. In the same way, software quality can be defined as conformance to explicitly stated and implicitly stated functional requirements. Here, the explicitly stated functional requirement can be derived from the requirements stated by the customer which are generally documented in some form. Implicit requirements are requirements which are not stated explicitly but are intended. Implicit functional requirements are standards which a software development company adheres to during the development process. Implicit functional requirements also include the requirements of good maintainability.

Quality software is reasonably bug-free, delivered on time and within budget, meets requirements and is maintainable. However, as discussed above, quality is a subjective term. It will depend on who the ‘customer’ is and their overall influence in the scheme of things. Each type of ‘customer’ will have their own slant on ‘quality’. The end-user might define quality as some thing which is user-friendly and bug-free.

Good quality software satisfies both explicit and implicit requirements. Software quality is a complex mix of characteristics and varies from application to application and the customer who requests for it.

.

Attributes of Quality

The following are some of the attributes of quality:

Auditability : The ability of software being tested against conformance to standard.

Compatibility : The ability of two or more systems or components to perform their required functions while sharing the same hardware or software environment.

Completeness: The degree to which all of the software's required functions and design constraints are present and fully developed in the requirements specification, design document and code.

Consistency: The degree of uniformity, standardisation, and freedom from contradiction among the documents or parts of a system or component.

Correctness: The degree to which a system or component is free from faults in its specification, design, and implementation. The degree to which software, documentation, or other items meet specified requirements.

Feasibility: The degree to which the requirements, design, or plans for a system or component can be implemented under existing constraints.

Modularity : The degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.

Predictability: The degree to which the functionality and performance of the software are determinable for a specified set of inputs.

Robustness: The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions.

Structuredness : The degree to which the SDD (System Design Document) and code possess a definite pattern in their interdependent parts. This implies that the design has proceeded in an orderly and systematic manner (e.g., top-down, bottom-up). The modules are cohesive and the software has minimised coupling between modules.

Testability: The degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met.

Traceability: The degree to which a relationship can be established between two or more products of the development process. The degree to which each element in a software development product establishes its reason for existing (e.g., the degree to which each element in a bubble chart references the requirement that it satisfies). For example, the system's functionality must be traceable to user requirements.

Understandability: The degree to which the meaning of the SRS, SDD, and code are clear and understandable to the reader.

Verifiability : The degree to which the SRS, SDD, and code have been written to facilitate verification and testing.

Causes of error in Software

- Misinterpretation of customers' requirements/communication
- Incomplete/erroneous system specification
- Error in logic
- Not following programming/software standards
- Incomplete testing
- Inaccurate documentation/no documentation
- Deviation from specification
- Error in data modeling and representation.

Measurement of Software Quality (Quality metrics)

Software quality is a set of characteristics that can be measured in all phases of software development.

Defect metrics

- Number of design changes required
- Number of errors in the code
- Number of bugs during different stages of testing
- Reliability metrics
- It measures the mean time to failure (MTTF), that may be defined as probability of failure during a particular interval of time. This will be discussed in software reliability.

Maintainability metrics

- Complexity metrics are used to determine the maintainability of software. The complexity of software can be measured from its control flow.

Consider the graph of *Figure 3.1*. Each node represents one program segment and edges represent the control flow. The complexity of the software module represented by the graph can be given by simple formulae of graph theory as follows:

V(G) = e – n + 2 where

V(G) : is called Cyclomatic complexity of the program
e = number of edges
n = number of nodes

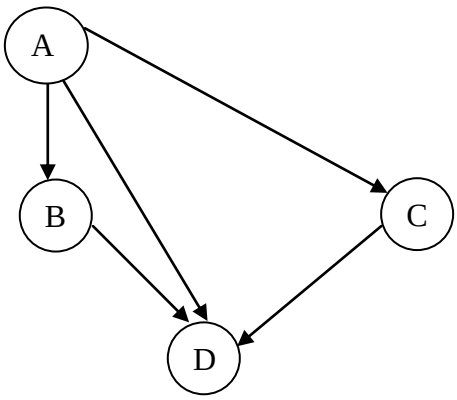

Figure 3.1: A software module

Applying the above equation the complexity V(G) of the graph is found to be 3.

The cyclomatic complexity has been related to programming effort, maintenance effort and debugging effort. Although cyclomatic complexity measures program complexity, it fails to measure the complexity of a program without multiple conditions.

The information flow within a program can provide a measure for program complexity.

Important parameters for measurement of Software Quality

- To the extent it satisfies user requirements; they form the foundation to measure software quality.
- Use of specific standards for building the software product. Standards could be organisation's own standards or standards referred in a contractual agreement.
- Implicit requirements which are not stated by the user but are essential for quality software.

Software Quality Assurance

The aim of the Software Quality Assurance process is to develop high-quality software product.

The purpose of Software Quality Assurance is to provide management with appropriate visibility into the process of the software project and of the products being built. Software Quality Assurance involves reviewing and auditing the software products throughout the development lifecycle to verify that they conform to explicit requirements and implicit requirements such as applicable procedures and standards. Compliance with agreed-upon standards and procedures is evaluated through process monitoring, product evaluation, and audits.

Software Quality Assurance (SQA) is a planned, coordinated and systematic actions necessary to provide adequate confidence that a software product conforms to established technical requirements.

Software Quality Assurance is a set of activities designed to evaluate the process by which software is developed and/or maintained.

The process of Software Quality Assurance

1. Defines the requirements for software controlled system fault/failure detection, isolation, and recovery;

2. Reviews the software development processes and products for software error prevention and/ or controlled change to reduced functionality states; and

3. Defines the process for measuring and analysing defects as well as reliability and maintainability factors.

Software engineers, project managers, customers and Software Quality Assurance groups are involved in software quality assurance activity. The role of various groups in software quality assurance are as follows:

- **Software engineers**: They ensure that appropriate methods are applied to develop the software, perform testing of the software product and participate in formal technical reviews.

- **SQA group**: They assist the software engineer in developing high quality product. They plan quality assurance activities and report the results of review.

☞ **Check Your Progress 1**

1) What is auditability?
……

2) Traceability is the ability of traceacing design back to _____

3) Software quality is designed in to software and can't be infused after the product is released. Explain?
……

……

3.3 FORMAL TECHNICAL REVIEW

What is software Review ? Software review can't be defined as a filter for the software engineering process. The purpose of any review is to discover errors in analysis, design, coding, testing and implementation phase of software development cycle. The other purpose of review is to see whether procedures are applied uniformly and in a manageable manner.

Reviews are basically of two types, informal technical review and formal technical review.

- Informal Technical Review : Informal meeting and informal desk checking.
- Formal Technical Review: A formal software quality assurance activity through various approaches such as structured walkthroughs, inspection, etc.

What is Formal Technical Review: Formal technical review is a software quality assurance activity performed by software engineering practitioners to improve software product quality. The product is scrutinised for completeness, correctness, consistency, technical feasibility, efficiency, and adherence to established standards and guidelines by the client organisation.

Structured walkthrough is a review of the formal deliverables produced by the project team. Participants of this review typically include end-users and management of the client organization, management of the development organisation, and sometimes auditors, as well as members of the project team. As such, these reviews are more formal in nature with a predefined agenda, which may include presentations, overheads, etc.

An inspection is more formalised than a 'Structured walkthrough', typically with 3-8 people. The subject of the inspection is typically a document such as a requirements specification or a test plan, and the purpose is to find problems and see what's missing, not to suggest rectification or fixing. The result of the inspection meeting should be a written report.

Verification **:** Verification generally involves reviews to evaluate whether correct methodologies have been followed by checking documents, plans, code, requirements, and specifications. This can be done with checklists, walkthroughs, etc.

Validation **:** Validation typically involves actual testing and takes place after verifications are completed.

Objectives of Formal Technical Review

- To uncover errors in logic or implementation
- To ensure that the software has been represented accruing to predefined standards
- To ensure that software under review meets the requirements
- To make the project more manageable.

Each Formal Technical Review (FTR) is conducted as a meeting and requires well coordinated planning and commitments.

For the success of formal technical review, the following are expected:

- The schedule of the meeting and its agenda reach the members well in advance
- Members review the material and its distribution
- The reviewer must review the material in advance.

The meeting should consist of two to five people and should be restricted to not more than 2 hours (preferably). The aim of the review is to review the product/work and the performance of the people. When the product is ready, the producer (developer) informs the project leader about the completion of the product and requests for review. The project leader contacts the review leader for the review. The review leader asks the reviewer to perform an independent review of the product/work before the scheduled FTR.

Result of FTR

- Meeting decision
  - Whether to accept the product/work without any modification
  - Accept the product/work with certain changes
  - Reject the product/work due to error

- Review summary report
  - What was reviewed?
  - Who reviewed it?
  - Findings of the review
  - Conclusion

Checklist - Typical activities for review at each phase are described below:

Software Concept and Initiation Phase

Involvement of SQA team in both writing and reviewing the project management plan in order to assure that the processes, procedures, and standards identified in the plan are appropriate, clear, specific, and auditable.

- Have design constraints been taken in to account?
- Whether best alternatives have been selected.

Software Requirements Analysis Phase

During the software requirements phase, review assures that software requirements are complete, testable, and properly expressed as functional, performance, and interface requirements. The output of a software requirement analysis phase is Software Requirements Specification (SRS). SRS forms the major input for review.

Compatibility

- Does the interface enables compatibility with external interfaces?
- Whether the specified models, algorithms, and numerical techniques are compatible?

Completeness

- Does it include all requirements relating to functionality, performance, system constraints, etc.?
- Does SRS include all user requirements?
- Are the time-critical functions defined and identified?
- Are the requirements consistent with available resources and budget?
- Does the SRS include requirements for anticipated future changes?

Consistency

- Are the requirements consistent with each other? Is the SRS free of contradictions?
- Whether SRS uses standard terminology and definitions throughout.
- Has the impact of operational environment on the software been specified in the SRS?

Correctness

- Does the SRS conform to SRS standards?
- Does the SRS define the required responses to all expected types of errors and failure? hazard analysis?
- Were the functional requirements analysed to check if all abnormal situations are covered by system functions?
- Does SRS reference development standards for preparation?
- Does the SRS identify external interfaces in terms of input and output mathematical variables?
- Has the rationale for each requirement been defined?
- Are the requirements adequate and correctly indicated?
- Is there justification for the design/implementation constraints?

Traceability

- Are the requirements traceable to top level?
- Is there traceability from the next higher level of specification?
- Is SRS traceable forward through successive software development phases like the design, coding and testing ?

Verifiability and Testability

- Are validation criteria complete?
- Is there a verification procedure defined for each requirement in the SRS?
- Are the requirements verifiable?

Software Design Phase

Reviewers should be able to determine whether or not all design features are consistent with the requirements. And, the program should meet the requirements. The output of the software design phase is a system design document (SDD) and forms an input for a Formal Technical Review.

Completeness

- Whether all the requirements have been addressed in the SDD?
- Have the software requirements been properly reflected in software architecture?
- Are algorithms adequate, accurate, and complete?
- Does the design implement the required program behavior with respect to each program interface?
- Does the design take into consideration all expected conditions?
- Does the design specify appropriate behaviour in case of an unexpected or improper input and other abnormal conditions?

Consistency

- Are standard terminology and definitions used throughout the SDD? Is the style of presentation and the level of detail consistent throughout the document.
- Does the design configuration ensure integrity of changes?
- Is there compatibility of the interfaces?
- Is the test documentation compatible with the test requirements of the SRS?
- Are the models, algorithms, and numerical techniques that are specified mathematically compatible?
- Are input and output formats consistent to the extent possible?
- Are the designs for similar or related functions are consistent?
- Are the accuracies and units of inputs, database elements, and outputs that are used together in computations or logical decisions compatible?

Correctness

- Does the SDD conform to design documentation standards?
- Does the design perform only that which is specified in the SRS?
- Whether additional functionality is justified?
- Is the test documentation current and technically accurate?
- Is the design logic sound i.e., will the program do what is intended?
- Is the design consistent with documented descriptions?
- Does the design correctly accommodate all inputs, outputs, and database elements ?

Modifiability

- The modules are organised such that changes in the requirements only require changes to a small number of modules.
- Functionality is partitioned into programs to maximize the internal cohesion of programs and to minimize program coupling.
- Is the design structured so that it comprises relatively small, hierarchically related programs or sets of programs, each performing a particular unique function?
- Does the design use a logical hierarchical control structure?

Traceability

- Is the SDD traceable to requirements in SRS?
- Does the SDD show mapping and complete coverage of all requirements and design constraints in the SRS?
- Whether the functions in the SDD which are outside the scope of SRS are defined and identified?

Verifiability

- Does the SDD describe each function using well-defined notation so that the SDD can be verified against the SRS
- Can the code be verified against the SDD?
- Are conditions, constraints identified so that test cases can be designed?

Review of Source Code

The following checklist contains the kinds of questions a reviewer may take up during source code review based on various standards

Completeness

- Is the code complete and precise in implementation?
- Is the code as per the design documented in the SDD?
- Are there any unreferenced or undefined variables, constants, or data types?
- Whether the code follows any coding standard that is referenced?

Consistency

- Is the code consistent with the SDD?
- Are the nomenclature of variables, functions uniform throughout?

Correctness

- Does the code conform to specified coding standards?
- Are all variables properly specified and used?
- Does the code avoid recursion?
- Does the code protect against detectable runtime errors?
- Are all comments accurate and sufficient?
- Is the code free of unintended infinite loops?

Modifiability

- Is the program documentation sufficient to facilitate future changes?
- Does the code refer to constants symbolically so as to facilitate change?
- Is the code written in a language with well-defined syntax and semantics?
- Are the references or data dictionaries included to show variable and constant access by the program?
- Does the code avoid relying on defaults provided by the programming language?

Traceability

- Is the code traceable to design document?
- Does the code identify each program uniquely?
- Is there a cross-reference through which the code can be easily and directly traced to the SDD?
- Does the code contain, or has reference to, a revision history of all code modifications done by different authors?
- Whether the reasons for such modification are specified and authorized?

Understandability

- Do the comment statements in the code adequately describe each routine?
- Whether proper indent and formatting has been used to enhance clarity?
- Has the formatting been used consistently?
- Does naming patterns of the variables properly reflect the type of variable?
- Is the valid range of each variable defined?

Software Testing and Implementation Phase

- Whether all deliverable items are tested.

- Whether test plans are consistent and sufficient to test the functionality of the systems.
- Whether non-conformance reporting and corrective action has been initiated?.
- Whether boundary value is being tested.
- Whether all tests are run according to test plans and procedures and any non-conformances are reported and resolved?
- Are the test reports complete and correct?
- Has it been certified that testing is complete and software including documentation are ready for delivery?

Installation phase

Completeness

- Are the components necessary for installation of a program in this installation medium ?
- Whether adequate information for installing the program, including the system requirements for running the program available.
- Is there more than one operating environment?
- Are installation procedures for different running environments available?
- Are the installation procedures clearly understandable?

☞ **Check Your Progress 2**

1) The purpose of review is to uncover errors and non conformity during testing phase. Yes ☐ No ☐

2) The result of a review does not include
   a) The items that are to be reviewed
   b) The person who reviews it
   c) Findings of the review
   d) Solution to a problem

3) What could become an input for FTR in design phase?

   ……

   ……

3.4 SOFTWARE RELIABILITY

Software Reliability: Unlike reliability of the hardware device, the term software reliability is difficult to measure. In the software engineering environment, software reliability is defined as the probability that software will provide failure-free operation in a fixed environment for a fixed interval of time.

Software reliability is typically measured per a unit of time, whereas probability of failure is generally time independent. These two measures can be easily related if you know the frequency with which inputs are executed per unit of time. Mean-time-to-failure (MTTF) is the average interval of time between failures; this is also sometimes referred to as Mean-time-before-failure.

Possibly the greatest problem in the field of software reliability estimation has to do with the accuracy of operational profiles. Without accurate profiles, estimates will almost certainly be wrong. An operational profile is the probability density function (over the entire input space) that best represents how the inputs would be selected during the life-time of the software. There is nothing fancy about operational profiles; they are really just “guesses” about what inputs will occur in the future.

Definitions of Software reliability

- IEEE Definition: The probability that software will not cause the failure of a system for a specified time under specified conditions. The probability is a function of the inputs to and use of the system in the software. The inputs to the system determine whether existing faults, if any, are encountered.

- The ability of a program to perform its required functions accurately and reproducibly under stated conditions for a specified period of time.

Software Reliability Models

A number of models have been developed to determine software defects/failures. All these models describe the occurrence of defects/failures as a function of time. This allows us to define reliability. These models are based on certain assumptions which can be described as below:

- The failures are independent of each other, i.e., one failure has no impact on other failure(s).
- The inputs are random samples.
- Failure intervals are independent and all software failures are observed.
- Time between failures is exponentially distributed.

The following formula gives the cumulative number of defects observed at a time 't'.

$$D(t) = T_d (1 - e^{-bct})$$

D(t) = Cumulative number of defects observed at a time t

T_d = Total number of defects

'b' and 'c' are constants and depend on historical data of similar software for which the model is being applied

We may find the mean time to failure (MMFT) as below:

$$MTTF(t) = e^{bct} / c\, T_d$$

3.5 SOFTWARE QUALITY STANDARDS

Software standards help an organization to adopt a uniform approach in designing, developing and maintaining software. There are a number of standards for software quality and software quality assurance. Once an organisation decides to establish a software quality assurance process, standards may be followed to establish and operate different software development activities and support activities. A number of organisations have developed standards on quality in general and software quality in specific.

Software Engineering Institute (SEI) has developed what is called a '*Capability Maturity Model'* (CMM) now called the *Capability Maturity Model Integration* (CMMI) to help organisations to improve software development processes.

It is a model of 5 levels of process maturity that determine the effectiveness of an organisation in delivering quality software. Organizations can receive CMMI ratings

by undergoing assessments by qualified auditors. The organizations are rated as CMM Level 1, CMM Level 2 etc. by evaluating their organisational process maturity.

SEI-CMM Level 1: Characterised by un organised, chaos, periodic panics, and heroic efforts required by individuals to successfully complete projects. Successes depend on individuals and may not be repeatable.

SEI-CMM Level 2 : Software project tracking, requirements management, realistic planning, and configuration management processes are in place; successful practices can be repeated.

SEI-CMM Level 3: Standard software development and maintenance processes are integrated throughout an organisation; a Software Engineering Process Group is in place to oversee software processes, and training programs are used to ensure understanding and compliance.

SEI-CMM Level 4: Metrics are used to track productivity, processes, and products. Project performance is predictable, and quality is consistently high.

SEI-CMM Level 5: The focus is on continuous process improvement. The impact of new processes and technologies can be predicted and effectively implemented when required.

The International Organisation for Standardisation (ISO) developed the ISO 9001:2000 standard (which replaces the previous set of three standards of 1994) that helps the organisation to establish, operate, maintain and review a quality management system that is assessed by outside auditors. The standard is generic in nature and can be applied to any organisation involved in production, manufacturing service including an organisation providing software services.

It covers documentation, design, development, production, testing, installation, servicing, and other processes. It may be noted that ISO certification does not necessarily indicate quality products. It only indicates that the organisation follows a well documented established process. *Table 3.1* gives a list of standards along with the corresponding titles.

Table 3.1 : List of standards

Standards	Title
ISO/IEC 90003	Software Engineering. Guidelines for the Application of ISO 9001:2000 to Computer Software
ISO 9001:2000	Quality Management Systems - Requirements
ISO/IEC 14598-1	Information Technology-Evaluation of Software Products-General Guide
ISO/IEC 9126-1	Software Engineering - Product Quality - Part 1: Quality Model
ISO/IEC 12119	Information Technology-Software Packages-Quality Requirements and Testing
ISO/IEC 12207	Information Technology-Software Life Cycle Processes
ISO/IEC 14102	Guideline For the Evaluation and Selection of CASE Tools
IEC 60880	Software for Computers in the Safety Systems of Nuclear Power Stations
IEEE 730	Software Quality Assurance Plans
IEEE 730.1	Guide for Software Assurance Planning
IEEE 982.1	Standard Dictionary of Measures to produce reliable software
IEEE 1061	Standard for a Software Quality Metrics Methodology
IEEE 1540	Software Life Cycle Processes - Risk Management
IEEE 1028	Software review and audits
IEEE 1012	Software verification and validation plans

☞ **Check Your Progress 3**

1) Success of a project depends on individual effort. At what stage of maturity does the organisation's software process can be rated?

   ……..

   ……...

2) Why ISO 9001 : 2000 is called generic standard?

   ……..

   ……..

3) What is the difference between SEI CMM standards and ISO 9000 : 2000 standards?

   ……

   ……

3.6 SUMMARY

Software quality assurance is applied at every stages of system development through a process of review to ensure that proper methodology and procedure has been followed to produce the software product. It covers the entire gamut of software development activity. Software quality is conformance with explicitly defined requirements which may come from a customer requirement or organisation may define their own standards. Software review also called a Formal Technical Review is one of the most important activity of software quality assurance process. Software reliability provides measurement of software dependability and probability of failure is generally time independent. The aim of software quality assurance is to improve both software product and process through which the software is built. The help of various software standards taken to establish and operate software quality assurance process.

3.7 SOLUTIONS/ANSWERS

Check Your Progress 1

1) Auditability is that attribute of software which determines the degree to which a software can be tested for conformance against a standard.

2) Requirements.

3) Software quality assurance activity is an umbrella activity comprising activities like application of standards, technical methods, review, software testing, change control, measurement, control & reporting during the process of software development life cycle. High quality product can come from high quality design specification. Unlike quality, testing quality assurance can't be achieved at the end of the product completion phase.

Check Your Progress 2

1) No. The purpose of any review is to discover errors in the analysis, design, coding, testing and maintenance phases of software development life cycle.

2) Solution to a problem.

3) Output of the software design phase is a system design document (SDD) and it is an input for the Formal Technical Review.

Check Your Progress 3

1) SEI CMM level 1.
2) The standards are called generic in nature as they can be applied to any organisation involved in the production, manufacturing service including organisations providing software services which involve no manufacturing process.
3) SEI CMM standards are developed to rate the maturity of organisation's software development process. The maturity is rated from level 1 to level 5, i.e., from immature (no formal process) to predictable matured process with focus on continuous improvement. In case of ISO 9001 : 2000, the organisation's process (quality system) is tested against conformance to the requirements of ISO 9000 : 2000 standard.

3.8 FURTHER READINGS

1) *Software Quality, 1997,* Mordechai Ben-Menachem and Garry S.Marliss; Thomson Learning.

2) *Software Engineering, Sixth Edition, 2001*, Ian Sommerville; Pearson Education.

3) *Software Engineering – A Practitioner's Approach*, Roger S. Pressman; McGraw-Hill International Edition.

Reference websites

http://www.rspa.com
http://www.ieee.org

UNIT 4 SOFTWARE CHANGE MANAGEMENT

Structure **Page Nos.**

4.0 INTRODUCTION

Software change management is an umbrella activity that aims at maintaining the integrity of software products and items. Change is a fact of life but uncontrolled change may lead to havoc and may affect the integrity of the base product. Software development has become an increasingly complex and dynamic activity. Software change management is a challenging task faced by modern project managers, especially in a environment where software development is spread across a wide geographic area with a number of software developers in a distributed environment. Enforcement of regulatory requirements and standards demand a robust change management. The aim of change management is to facilitate justifiable changes in the software product.

4.1 OBJECTIVES

After studying this unit, you should be able to:

- define baselines;
- know the concept of version control and change control, and
- audit and report software change management activity.

4.2 BASELINES

Baseline is a term frequently used in the context of software change management. Before we understand the concept of baseline, let us define a term which is called software configuration item. A software configuration item is any part of development and /or deliverable system which may include software, hardware, firmware, drawings, inventories, project plans or documents. These items are independently tested, stored, reviewed and changed. A software configuration item can be one or more of the following:

- System specification
- Source code
- Object code
- Drawing
- Software design

- Design data
- Database schema and file structure
- Test plan and test cases
- Product specific documents
- Project plan
- Standards procedures
- Process description

Definition of Baseline: A baseline is an approved software configuration item that has been reviewed and finalised. An example of a baseline is an approved design document that is consistent with the requirements. The process of review and approval forms part of the formal technical review. The baseline serves as a reference for any change. Once the changes to a reference baseline is reviewed and approved, it acts as a baseline for the next change(s).

A baseline is a set of configuration items (hardware, documents and software components) that have been formally reviewed and agreed upon, thereafter serve as the basis for future development, and that can be changed only through formal change control procedures.

Figure 4.1 depicts a baseline for design specification.

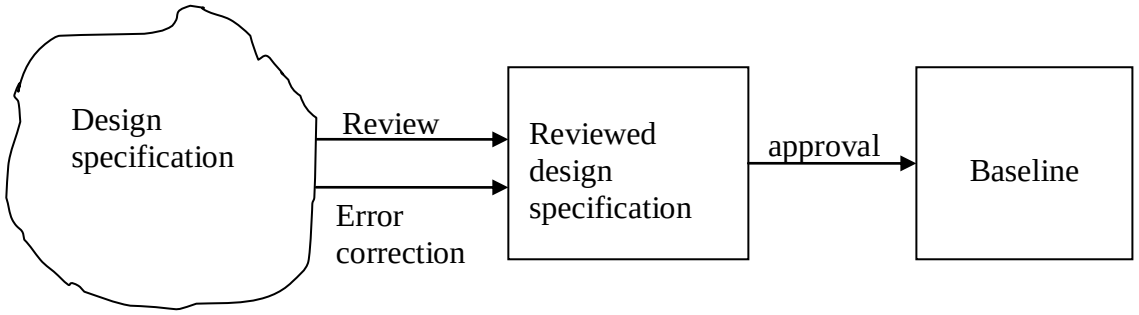

Figure 4.1 : A baseline for design specification

Figure 4.2 depicts the evolution of a baseline.

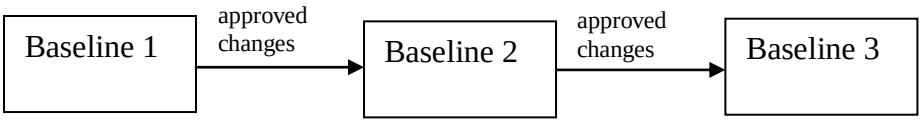

Figure 4.2: Evolution of a baseline

A baseline is functionally complete, i.e., it has a defined functionality. The features of these functionalities are documented for reference for further changes. The baseline has a defined quality which has undergone a formal round of testing and reviews before being termed as a baseline. Any baseline can be recreated at any point of time.

The process of change management

The domain of software change management process defines how to control and manage changes.

A formal process of change management is acutely felt in the current scenario when the software is developed in a very complex distributed environment with many versions of a software existing at the same time, many developers involved in the development process using different technologies. The ultimate bottomline is to maintain the integrity of the software product while incorporating changes.

The following are the objectives of software change management process:

1. **Configuration identification**: The source code, documents, test plans, etc. The process of identification involves identifying each component name, giving them a version name (a unique number for identification) and a configuration identification.
2. **Configuration control**: Cntrolling changes to a product. Controlling release of a product and changes that ensure that the software is consistent on the basis of a baseline product.
3. **Review**: Reviewing the process to ensure consistency among different configuration items.
4. **Status accounting** : Recording and reporting the changes and status of the components.
5. **Auditing and reporting**: Validating the product and maintaining consistency of the product throughout the software life cycle.

Process of changes: As we have discussed, baseline forms the reference for any change. Whenever a change is identified, the baseline which is available in project database is copied by the change agent (the software developer) to his private area. Once the modification is underway the baseline is locked for any further modification which may lead to inconsistency. The records of all changes are tracked and recorded in a status accounting file. After the changes are completed and the changes go through a change control procedure, it becomes a approved item for updating the original baseline in the project database.

Figure 4.3 depicts the process of changes to a baseline.

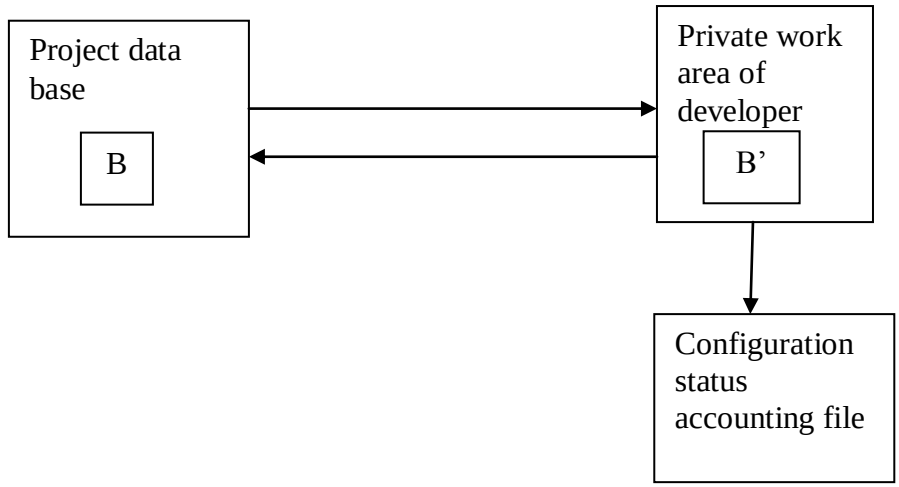

Figure 4.3: Process of changes to baseline

All the changes during the process of modification are recorded in the configuration status accounting file. It records all changes made to the previous baseline B to reach the new baseline B'. The status accounting file is used for configuration authentication which assures that the new baseline B' has all the required planned and approved changes incorporated. This is also known as auditing.

☞ **Check Your Progress 1**

1) _____ serves as reference for any change.

2) What is the aim of software change management process?

……...

……...

4.3 VERSION CONTROL

Version control is the management of multiple revisions of the same unit of item during the software development process. For example, a system requirement specification (SRS) is produced after taking into account the user requirements which change with time into account. Once a SRS is finalized, documented and approved, it is given a document number, with a unique identification number. The name of the items may follow a hierarchical pattern which may consist of the following:

- Project identifier
- Configuration item (or simply item, e.g. SRS, program, data model)
- Change number or version number

The identification of the configuration item must be able to provide the relationship between items whenever such relationship exists.

The identification process should be such that it uniquely identifies the configuration item throughout the development life cycle, such that all such changes are traceable to the previous configuration. An evolutionary graph graphically reflects the history of all such changes. The aim of these controls is to facilitate the return to any previous state of configuration item in case of any unresolved issue in the current unapproved version.

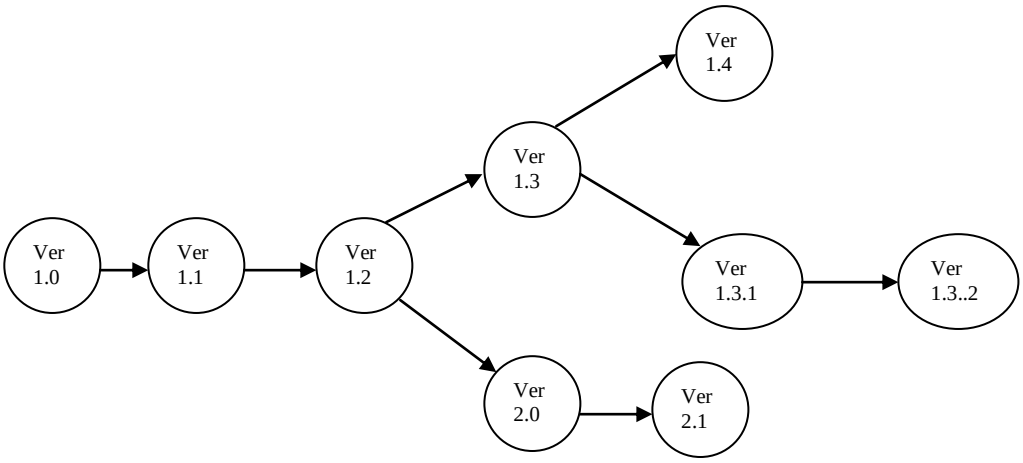

Figure 4.4 : An evolutionary graph for a different version of an item

The above evolutionary graph(*Figure 4.4*) depicts the evolution of a configuration item during the development life cycle. The initial version of the item is given version number Ver 1.0. Subsequent changes to the item which could be mostly fixing bugs or adding minor functionality is given as Ver 1.1 and Ver 1.2. After that, a major modification to Ver 1.2 is given a number Ver 2.0 at the same time, a parallel version of the same item without the major modification is maintained and given a version number 1.3.

Depending on the volume and extent of changes, the version numbers are given by the version control manager to uniquely identify an item through the software development lifecycle. It may be noted that most of the versions of the items are released during the software maintenance phase.

Software engineers use this version control mechanism to track the source code, documentation and other configuration items. In practice, many tools are available to store and number these configuration items automatically. As software is developed and deployed, it is common to expect that multiple versions of the same software are deployed or maintained for various reasons. Many of these versions are used by developers to privately work to update the software.

It is also sometimes desirable to develop two parallel versions of the same product where one version is used to fix a bug in the earlier version and other one is used to develop new functionality and features in the software. Traditionally, software developers maintained multiple versions of the same software and named them uniquely by a number. But, this numbering system has certain disadvantages like it does not give any idea about a nearly identical versions of the same software which may exist.

The project database maintains all copies of the different versions of the software and other items. It is quite possible that without each other's knowledge, two developers may copy the same version of the item to their private area and start working on it. Updating to the central project database after completing changes will lead to overwriting of each other's work. Most version control systems provide a solution to this kind of problem by locking the version for further modification.

Commercial tools are available for version control which performs one or more of following tasks;

- Source code control
- Revision control
- Concurrent version control

There are many commercial tools like Rational ClearCase, Microsoft Visual SourceSafe and a number of other commercial tools to help version control.

Managing change is an important part of computing. The programmer fixes bugs while producing a new version based on the feedback of the user. System administrator manages various changes like porting database, migrating to a new platform and application environment without interrupting the day to day operations. Revisions to documents are carried out while improving application.

An example of revision control

Let us consider the following simple HTML file in a web based application (welcome.htm)

```
<html>
<head>
<Title> A simple HTML Page</title>
</head>
<body>
<h1> Welcome to HTML Concepts</h1>
</body>
</html>
```

Once the code is tested and finalized, the first step is to register the program to he project database. The revision is numbered and this file is marked read-only to prevent any further undesirable changes. This forms the building block of source control. Each time the file is modified, a new version is created and a new revision number is given.

The first version of the file is numbered as version 1.0. Any further modification is possible only in the developer's private area by copying the file from the project

database. The process of copying the configuration object (the baseline version) is called check-out.

Figure 4.5 depicts the changes to a baselined file in project database.

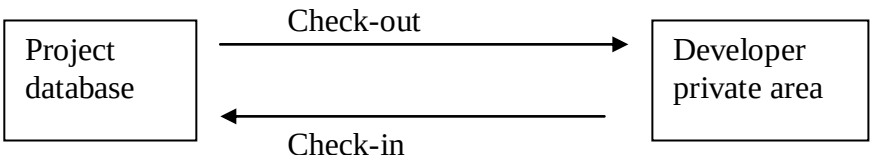

Figure 4.5: Changes to a baselined file in project database

The version (revision) control process starts with registering the initial versions of the file. This essentially enforces a check on the changes which ensure that the file can't be changed unless it is checked-out from the project database.

When a change is required to be made, allowing an email address to be added to the above html file, the developer will extract the latest version of the file welcome.htm from the project database. Once it is checked-out from the project database for modification, the file is locked for further modifications by any other developer. Other developers can check-out the file for read-only purpose.

Now, the following lines are added to the file welcome.htm.

```
<hr>
a href=mailto:webmaster@xyz.com> webmaster</a>
<hr>
```

The revised version of the file welcome.htm become

```
<html>
<head>
<Title> A simple HTML Page</title>
</head>
<body>
<h1> Welcome to HTML Concepts</h1>
<hr>
a href=mailto:webmaster@xyz.com> webmaster</a>
<hr>
</body>
</html>
```

Then the developer check-in's the revised version of the file to the project database with a new version (revision) number version 1.1 i.e. the first revision along with the details of the modification done.

Suppose another modification is done by adding a graphic to the html file welcome.htm. This becomes version 1.2. The version tree after two modifications looks as shown below (*Figure 4.6*).

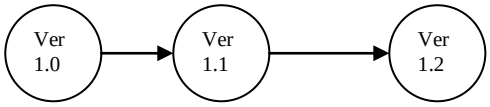

Figure 4.6: Version tree of welcome.htm

Suppose further modification is required for text-based browser as graphic will not be supported by text-based browser. Then the version 1.1 will be selected from the project database. This shows the necessity of storing all versions of the file in the project database.

☞ **Check Your Progress 2**

1) _____ is an example of source code control tool.

2) Version control mechanism allows multiple versions of an item to be maintained at the same time. (Yes/No)

3) How do version control systems ensure that two software developers do not attempt the same change at the same time?

……

……...

4.4 CHANGE CONTROL

Change is a fact of life, and the same applies to software development. Although, all changes requested by the user are not justified changes, but most of them are. The real challenge of change manager and project leader is to accept and accommodate all justifiable changes without affecting the integrity of product or without any side effect. The central to change management process is change control which deals with the formal process of change control.

The adoption and evolution of changes are carried out in a disciplined manner. In a large software environment where, as changes are done by a number of software developers, uncontrolled and un-coordinated changes may lead to havoc grossly diverting from the basic features and requirements of the system. For this, a formal change control process is developed.

Change control is a management process and is to some extent automated to provide a systematic mechanism for change control. Changes can be initiated by the user or other stake holder during the maintenance phase, although a change request may even come up during the development phase of the software.

A change request starts as a beginning of any change control process. The change request is evaluated for merits and demerits, and the potential side effects are evaluated. The overall impact on the system is assessed by the technical group consisting of the developer and project manager. A change control report is generated by the technical team listing the extent of changes and potential side effects. A designated team called change control authority makes the final decision, based on the change control report, whether to accept or reject the change request.

A change order called engineering change order is generated after the approval of the change request by the change control authority. The engineering change order forms the starting point of effecting a change in the component. If the change requested is

not approved by the change control authority, then the decision is conveyed to the user or the change request generator.

Once, change order is received by the developers, the required configuration items are identified which require changes. The baseline version of configuration items are copied from the project data base as discussed earlier.

The changes are then incorporated in the copied version of the item. The changes are subject to review (called audit) by a designated team before testing and other quality assurance activity is carried out. Once the changes are approved, a new version is generated for distribution.

The change control mechanisms are applied to the items which have become baselines. For other items which are yet to attain the stage of baseline, informal change control may be applied. For non- baseline items, the developer may make required changes as he feels appropriate to satisfy the technical requirement as long as it does not have an impact on the overall system.

The role of the change control authority is vital for any item which has become a baseline item. All changes to the baseline item must follow a formal change control process.

As discussed, change request, change report and engineering change order (change order) are generated as part of the change control activity within the software change management process. These documents are often represented inprinted or electronic forms. The typical content of these documents is given below:

Software Change Request Format

1.0 Change request Identification

1.1 Name, identification and description of software configuration item(s):
The name, version numbers of the software configuration is provided. Also, a brief description of the configuration item is provided.

1.2 Requester and contact details: The name of the person requesting the change and contact details

1.3 Date, location, and time when the change is requested

2.0 Description of the change

2.1 Description : This section specifies a detailed description of the change request.

- 2.1.1 Background Information, Background information of the request.
- 2.1.2 Examples: Supporting information, examples, error report, and screen shoots
- 2.1.3 The change : A detailed discussion of the change requested.

2.2 Justification for the change : Detailed justification for the request.

2.3 Priority : The priority of the change depending on critical effect on system functionalities.

Software Change Report Format

1.0 Change report Identification

1.1 Name, identification and description of software configuration item(s): The name, version numbers of the software configuration item and a brief description of it.

1.2 Requester: The name and contact details of the person requesting the change.

1.3 Evaluator : The name of the person or team who evaluated the change request.

1.4 Date and time : When change report was generated.

2.0 Overview of changes required to accommodate request

2.1 Description of software configuration item that will be affected

2.2 Change categorization : Type of change, in a generic sense

2.3 Scope of the change : The evaluator's assessment of the change.

2.3.1 Technical work required including tools required etc. A description of the work required to accomplish the change including required tools or other special resources are specified here

2.3.2 Technical risks : The risks associated with making the change are described.

3.0 Cost Assessment : Cost assessment of the requested change including an estimate of time required.

4.0 Recommendation

4.1 Evaluator’s recommendation : This section presents the evaluator's recommendation regarding the change

4.2 Internal priority: How important is this change in the light of the business operation and priority assigned by the evaluator.

Engineering Change Order Format

1.0 Change order Identification

1.1 Name, identification and description of software configuration item(s) : The name, version numbers including a brief description of software configuration items is provided.

1.2 Name of Requester

1.3 Name of Evaluator

2.0 Description of the change to be made

2.1 Description of software configuration(s) that is affected

2.2 Scope of the change required

The evaluator's assessment of scope of the change in the configuration item(s).

2.2.1 Technical work and tools required : A description of the work and tools required to accomplish the change.

2.3 Technical risks: The risks associated with making the change are described in this section.

3.0 Testing and Validation requirements

A description of the testing and review approach required to ensure that the change has been made without any undesirable side effects.

3.1 Review plan : Description of reviews that will be conducted.

3.2 Test plan

Description of the test plans and new tests that are required.

Benefits of change control management

The existence of a formal process of change management helps the developer to identify the responsibility of code for which a developer is responsible. An idea is achieved about the changes that affect the main product. The existence of such mechanism provides a road map to the development process and encourages the developers to be more involved in their work.

Version control mechanism helps the software tester to track the previous version of the product, thereby giving emphasis on testing of the changes made since the last approved changes. It helps the developer and tester to simultaneously work on multiple versions of the same product and still avoid any conflict and overlapping of activity.

The software change management process is used by the managers to keep a control on the changes to the product thereby tracking and monitoring every change. The existence of a formal process reassures the management. It provides a professional approach to control software changes.

It also provides confidence to the customer regarding the quality of the product.

4.5 AUDITING AND REPORTING

Auditing

Auditing and Reporting helps change management process to ensure whether the changes have been properly implemented or not, whether it has any undesired impact on other components. A formal technical review and software configuration audit helps in ensuring that the changes have been implemented properly during the change process.

A Formal Technical Review generally concentrates on technical correctness of the changes to the configuration item whereas software configuration audit complements it by checking the parameters which are not checked in a Formal Technical Review.

- Whether a formal technical review is carried out to check the technical accuracy of the changes made?
- Whether the changes as identified and reported in the change order have been incorporated?
- Have the changes been properly documented in the configuration items?
- Whether standards have been followed.
- Whether the procedure for identifying, recording and reporting changes has been followed.

As it is a formal process, it is desirable to conduct the audit by a separate team other than the team responsible for incorporating the changes.

Reporting: Status reporting is also called status accounting. It records all changes that lead to each new version of the item. Status reporting is the bookkeeping of each release. The process involves tracking the change in each version that leads the latest(new) version.

The report includes the following:

- The changes incorporated
- The person responsible for the change
- The date and time of changes
- The effect of the change
- The reason for such changes (if it is a bug fixing)

Every time a change is incorporated it is assigned a unique number to identify it from the previous version. Status reporting is of vital importance in a scenario where a large number of developers work on the same product at the same time and have little idea about the work of other developers.

For example, in source code, reporting the changes may be as below:

****
****
****

Title : Sub routine Insert to Employee Data

Version : Ver 1.1.3

Purpose : To insert employee data in the master file

Author : John Wright

Date : 23/10/2001

Auditor : J Waltson

Modification History:

12/12/2002 : by D K N
To fix bugs discovered in the first release

4/5/2003 : by S K G
to allow validation in date of birth data

6/6/2004 : by S S P
To add error checking module as requested by the customer

☞ Check Your Progress 3

1) Who decides the acceptance of a change request?.

……

……

2) How auditing is different from a Formal Technical Review (FTR)?

……

……

4.6 SUMMARY

Software change management is an activity that is applied throughout the software development life cycle. The process of change management includes configuration identification, configuration control, status reporting and status auditing. Configuration identification involves identification of all configurations that require changes. Configuration or change control is the process of controlling the release of the product and the changes. Status accounting or reporting is the process of recording the status of each component. Review involves auditing consistency and completeness of the component.

4.7 SOLUTIONS/ANSWERS

Check Your Progress 1

1) Baseline.
2) The domain of software change management process defines how to control and manage changes. The ultimate aim is to maintain the integrity of the software product while incorporating changes.

Check Your Progress 2

1) Microsoft Visual SourceSafe
2) Yes
3) Version control system locks the configuration item once it is copied from project database for modification by a developer.

Check Your Progress 3

1) Change control authority
2) Formal Technical Review is a formal process to evaluate the technical accuracy of any process or changes. Whereas software change or audit is carried out by a separate team to ensure that proper change management procedure has been followed to incorporate the changes.

4.8 FURTHER READINGS

1) *Software Engineering, Sixth Edition, 2001*, Ian Sommerville; Pearson Education.
2) *Software Engineering – A Practitioner's Approach*, Roger S. Pressman; McGraw-Hill International Edition.

Reference websites

http://www.rspa.com
http://www.ieee.org