UNIT 5 SOFTWARE PROJECT PLANNING

Structure **Page Nos.**

5.0 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.

5.1 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.

5.2 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.

5.3 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 5.1 depicts the software project estimation.

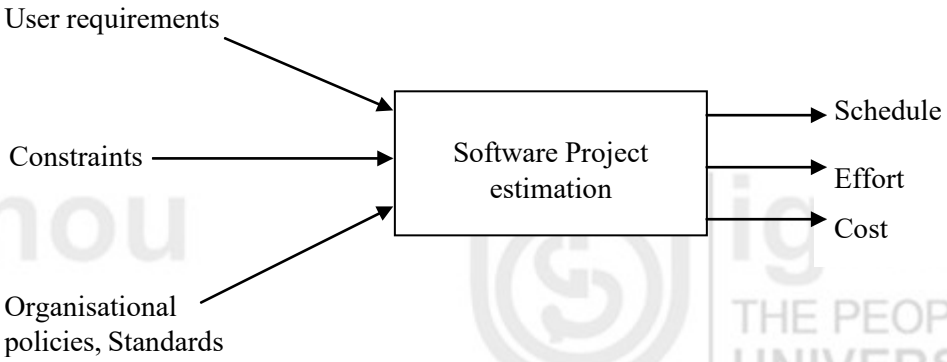

Figure 5.1: Software project estimation

5.3.1 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.

5.3.2 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.

5.3.3 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.

5.3.4 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 5.2 depicts the cost estimation process.

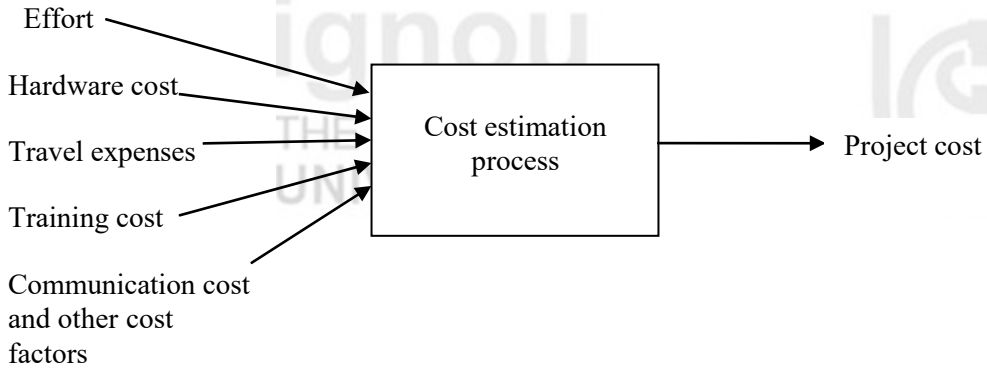

Figure 5.2 : Cost estimation process

Figure 5.3 depicts project estimation process.

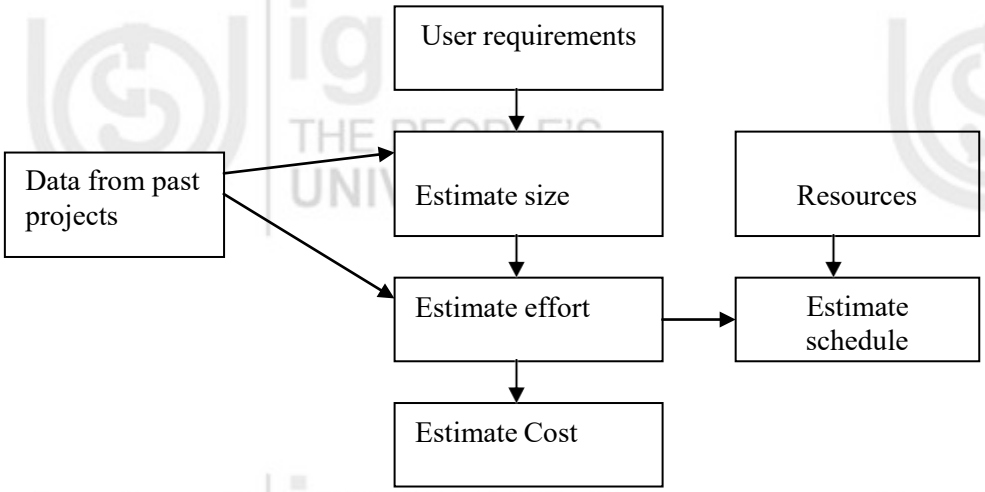

Figure 5.3: Project estimation process

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?

……

5.4 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

5.4.1 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

5.4.2 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.

5.4.3 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

5.4.4 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.

5.5 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 5.4 depicts a typical structure of estimation tools.

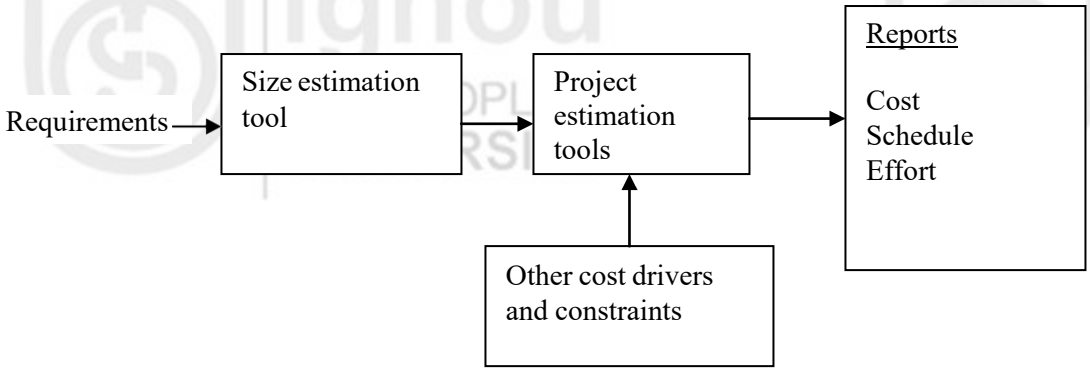

Figure 5.4: Typical structure of estimation tools

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/	

5.6 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.

5.7 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.

5.8 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 6 RISK MANAGEMENT AND PROJECT SCHEDULING

Structure **Page Nos.**

6.0 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.

6.1 OBJECTIVES

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

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

6.2 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.

6.3 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 6.1* depicts a risk table monitor. It depicts the risks that are being monitored.

Table 6.1: 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.

……

……

6.4 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:

6.4.1 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 6.1 depicts a risk manager tool.

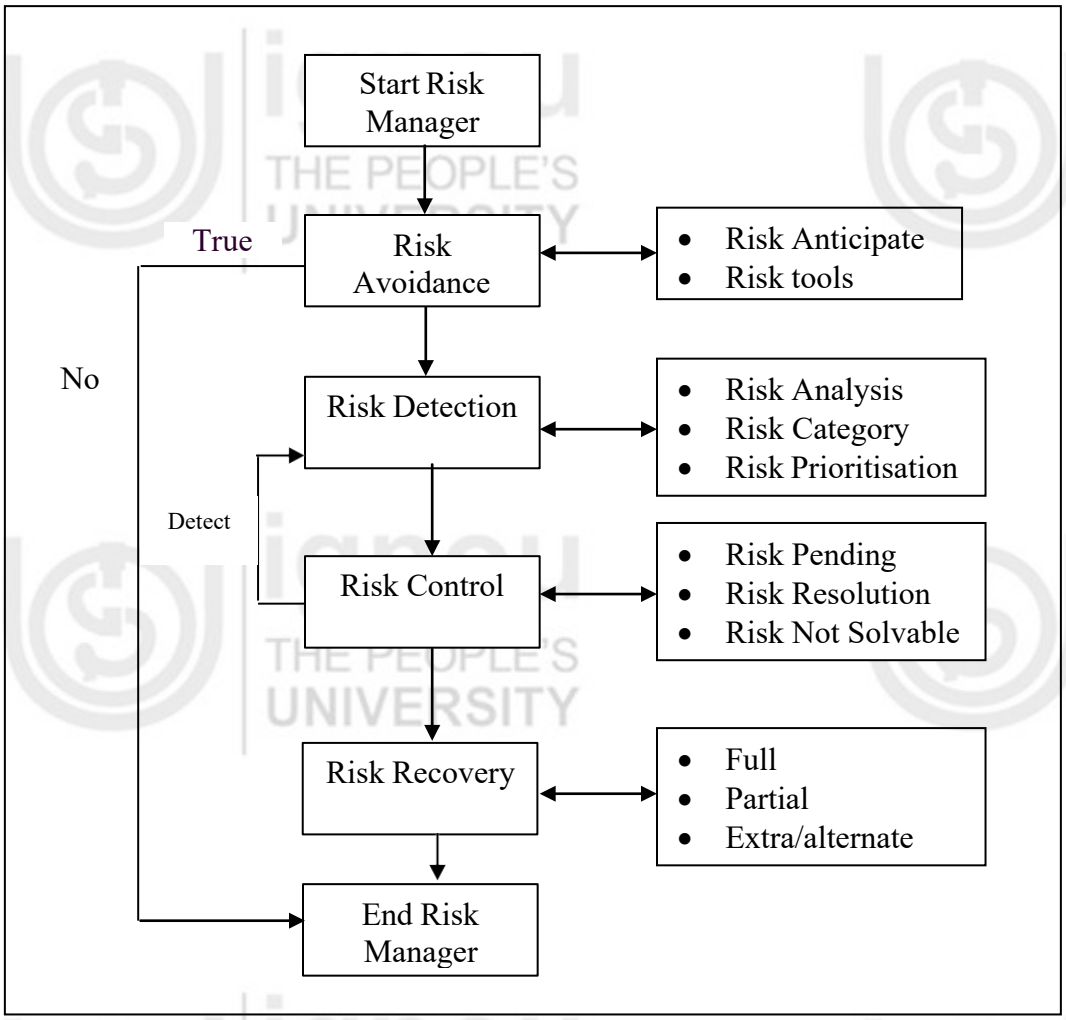

Figure 6.1 : Risk Manager Tool

From the *Figure 6.1*, 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.

6.4.2 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.

6.4.3 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 6.2 depicts a risk analysis table.

Table 6.2: 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.

6.5 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.

6.6 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?

……

……

6.7 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.

6.8 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.

6.9 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 6.2* depicts an example of a work breakdown structure.

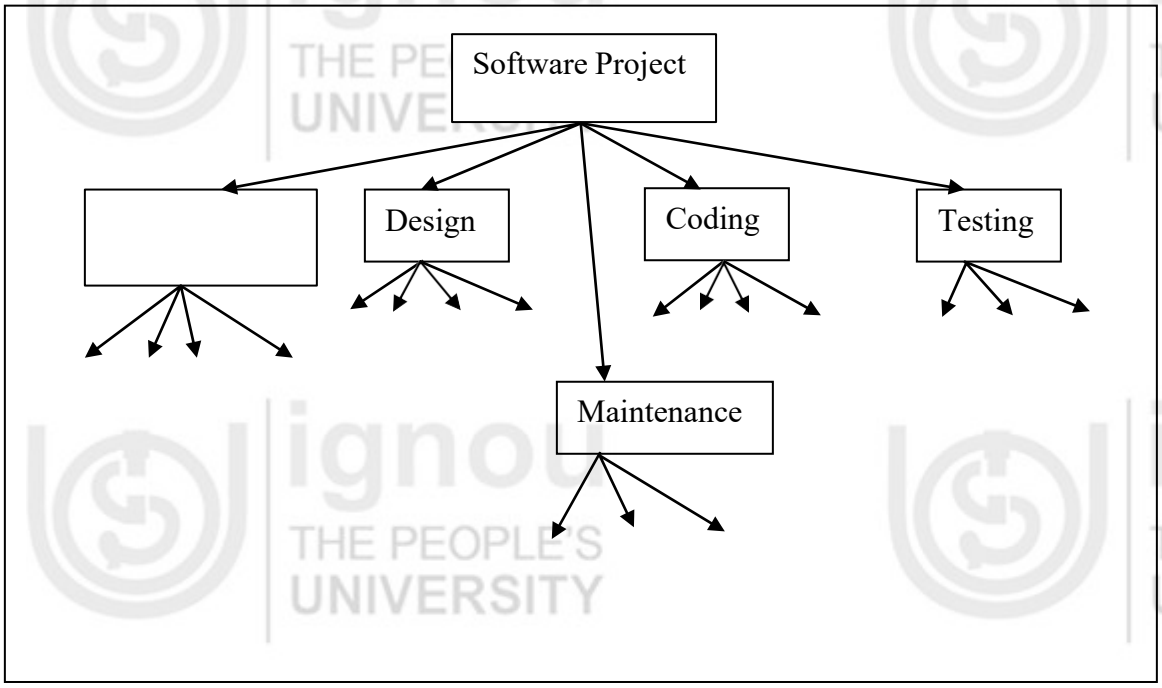

Figure 6.2: An example WBS

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 6.3* depicts a flow graph.

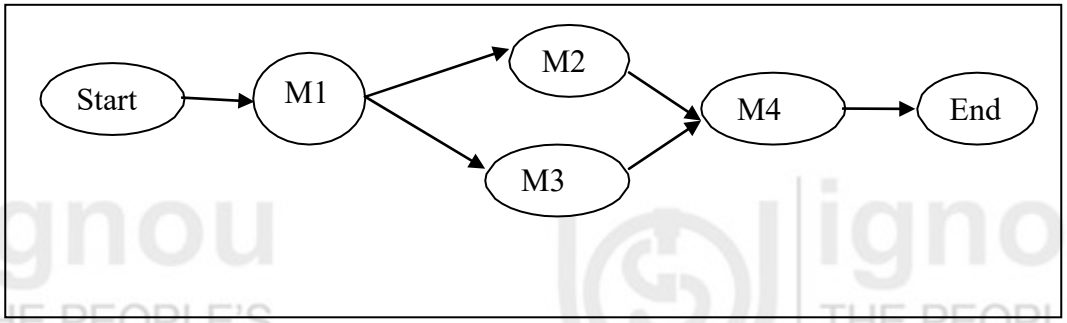

Figure 6.3 : Flow Graph

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 6.4* depicts a Gantt Chart. The circles indicate the milestones.

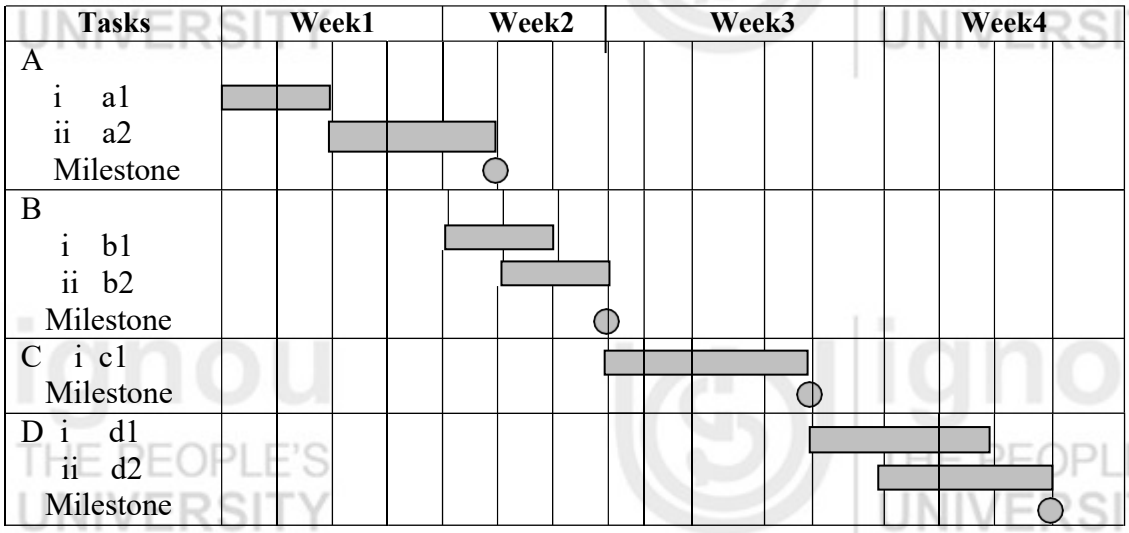

Figure 6.4: Gantt Chart

- **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.

6.10 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?

……

……

6.11 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.

6.12 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 .

6.13 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 7 SOFTWARE TESTING

Structure **Page Nos.**

7.0 INTRODUCTION

Testing means executing a program in order to understand its behaviour, that is, whether or not the program exhibits a failure, its response time or throughput for certain data sets, its mean time to failure, or the speed and accuracy with which users complete their designated tasks. In other words, it is a process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component. Testing can also be described as part of the process of Validation and Verification.

Validation is the process of evaluating a system or component during or at the end of the development process to determine if it satisfies the requirements of the system, or, in other words, are we building the correct system?

Verification is the process of evaluating a system or component at the end of a phase to determine if it satisfies the conditions imposed at the start of that phase, or, in other words, are we building the system correctly?

Software testing gives an important set of methods that can be used to evaluate and assure that a program or system meets its non-functional requirements.

To be more specific, software testing means that executing a program or its components in order to assure:

- The correctness of software with respect to requirements or intent;
- The performance of software under various conditions;
- The robustness of software, that is, its ability to handle erroneous inputs and unanticipated conditions;
- The usability of software under various conditions;
- The reliability, availability, survivability or other dependability measures of software; or
- Installability and other facets of a software release.

The purpose of testing is to show that the program has errors. The aim of most testing methods is to systematically and actively locate faults in the program and repair them. Debugging is the next stage of testing. Debugging is the activity of:

- Determining the exact nature and location of the suspected error within the program and
- Fixing the error. Usually, debugging begins with some indication of the existence of an error.

The purpose of debugging is to locate errors and fix them.

7.1 OBJECTIVES

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

- know the basic terms using in testing terminology;
- black box and White box testing techniques;
- other testing techniques; and
- some testing tools.

7.2 BASIC TERMS USED IN TESTING

Failure: A failure occurs when there is a deviation of the observed behavior of a program, or system, from its specification. A failure can also occur if the observed behaviour of a system, or program, deviates from its intended behavior.

Fault: A fault is an incorrect step, process, or data definition in a computer program. Faults are the source of failures. In normal language, software faults are usually referred to as "bugs".

Error: The difference between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition.

Test Cases: Ultimately, testing comes down to selecting and executing test cases. A test case for a specific component consists of three essential pieces of information:

- A set of test inputs;
- The expected results when the inputs are executed; and
- The execution conditions or environments in which the inputs are to be executed.

Some Testing Laws

- Testing can only be used to show the presence of errors, but never the absence or errors.
- A combination of different verification & validation (V&V) methods outprform any single method alone.
- Developers are unsuited to test their own code.
- Approximately 80% of the errors are found in 20% of the code.
- Partition testing, that is, methods that partition the input domain or the program and test according to those partitions. This is better than random testing.
- The adequacy of a test suite for coverage criterion can only be defined intuitively.

7.2.1 Input Domain

To conduct an analysis of the input, the sets of values making up the input domain are required. There are essentially two sources for the input domain. They are:

1. The software requirements specification in the case of black box testing method; and
2. The design and externally accessible program variables in the case of white box testing.

In the case of white box testing, input domain can be constructed from the following sources.

- Inputs passed in as parameters; Variables that are inputs to function under test can be: (i) Structured data such as linked lists, files or trees, as well as atomic data such as integers and floating point numbers;

(ii) A reference or a value parameter as in the *C* function declaration
int P(int *power, int base) {
...}

- Inputs entered by the user via the program interface;
- Inputs that are read in from files;
- Inputs that are constants and precomputed values; Constants declared in an enclosing scope of function under test, for example,

```
#define PI 3.14159
double circumference(double radius)
{
return 2*PI*radius;
}
```

In general, the inputs to a program or a function are stored in program variables. A program variable may be:

- A variable declared in a program as in the *C* declarations

For example: int base; char s[];

- Resulting from a read statement or similar interaction with the environment, For example: scanf(„„%d\n‟‟‟, &x);

7.2.2 Black Box and White Box Test Case Selection Strategies

- **Black box Testing**: In this method, where test cases are derived from the functional specification of the system; and
- **White box Testing**: In this method, where test cases are derived from the internal design specifications or actual code (Sometimes referred to as Glass-box).

Black box test case selection can be done without any reference to the program design or the program code. Test case selection is only concerned with the functionality and features of the system but not with its internal operations.

- The real advantage of black box test case selection is that it can be done *before* the design or coding of a program. Black box test cases can also help to get the design and coding correct with respect to the specification. Black box testing methods are good at testing for missing functions or program behavior that deviates from the specification. Black box testing is ideal for evaluating products that you intend to use in your systems.

- The main disadvantage of black box testing is that black box test cases cannot detect additional functions or features that have been added to the code. This is especially important for systems that need to be safe (additional code may interfere with the safety of the system) or secure (additional code may be used to break security).

White box test cases are selected using the specification, design and code of the program or functions under test. This means that the testing team needs access to the internal designs or code for the program.

- The chief advantage of white box testing is that it tests the internal details of the code and tries to check all the paths that a program can execute to determine if a problem occurs. White box testing can check additional functions or code that has been implemented, but not specified.

- The main disadvantage of white box testing is that you must wait until after design and coding of the programs of functions under test have been completed in order to select test cases.

Methods for Black box testing strategies

A number of test case selection methods exist within the broad classification of black box and white box testing.

For Black box testing strategies, the following are the methods:

- Boundary-value Analysis;
- Equivalence Partitioning.

We will also study State Based Testing, which can be classified as opaque box selection strategies that is somewhere between black box and white box selection strategies.

Boundary-value-analysis

The basic concept used in Boundary-value-analysis is that if the specific test cases are designed to check the boundaries of the input domain then the probability of detecting an error will increase. If we want to test a program written as a function F with two input variables x and y., then these input variables are defined with some boundaries like $a1 \leq x \leq a2$ and $b1 \leq y \leq b2$. It means that inputs x and y are bounded by two intervals [a1, a2] and [b1, b2].

Test Case Selection Guidelines for Boundary Value Analysis

The following set of guidelines is for the selection of test cases according to the principles of boundary value analysis. The guidelines do not constitute a firm set of rules for every case. You will need to develop some judgement in applying these guidelines.

1. If an *input* condition specifies a range of values, then construct valid test cases for the ends of the range, and invalid input test cases for input points just beyond the ends of the range.
2. If an *input* condition specifies a number of values, construct test cases for the minimum and maximum values; and one beneath and beyond these values.
3. If an *output* condition specifies a range of values, then construct valid test cases for the ends of the output range, and invalid input test cases for situations just beyond the ends of the output range.
4. If an *output* condition specifies a number of values, construct test cases for the minimum and maximum values; and one beneath and beyond these values.
5. If the input or output of a program is an ordered set (e.g., a sequential file, linear list, table), focus attention on the first and last elements of the set.

Example 1: Boundary Value Analysis for the Triangle Program

Consider a simple program to classify a triangle. Its input consists of three positive integers (say x, y, z) and the data types for input parameters ensures that these will be integers greater than zero and less than or equal to 100. The three values are interpreted as representing the lengths of the sides of a triangle. The program then prints a message to the standard output that states whether the triangle, if it can be formed, is scalene, isosceles, equilateral, or right-angled.

Solution: Following possible boundary conditions are formed:

1. Given sides *(*A; B; C) for a scalene triangle, the sum of any two sides is greater than the third and so, we have boundary conditions $A + B > C$, $B + C > A$ and $A + C > B$.
2. Given sides *(*A; B; C) for an isosceles triangle two sides must be equal and so we have boundary conditions $A = B$, $B = C$ or $A = C$.
3. Continuing in the same way for an equilateral triangle the sides must all be of equal length and we have only one boundary where $A = B = C$.
4. For right-angled triangles, we must have $A^2+B^2 = C^2$.

On the basis of the above boundary conditions, test cases are designed as follows (*Table 7.1*):

Table 7.1: Test cases for Example-1

Test case	x	y	z	Expected Output
1	100	100	100	Equilatera*l* triangle
2	50	3	50	Isosceles triangle
3	40	50	40	Equilatera*l* triangle
4	3	4	5	Right-angled triangles
5	10	10	10	Equilatera*l* triangle
6	2	2	5	Isosceles triangle
7	100	50	100	Scalene triangle
8	1	2	3	Non-triangles
9	2	3	4	Scalene triangle
10	1	3	1	Isosceles triangle

Equivalence Partitioning

Equivalence Partitioning is a method for selecting test cases based on a partitioning of the input domain. The aim of equivalence partitioning is to divide the input domain of the program or module into classes (sets) of test cases that have a similar effect on the program. The classes are called Equivalence classes.

Equivalence Classes

An Equivalence *Class* is a set of inputs that the program treats identically when the program is tested. In other words, a test input taken from an equivalence class is representative of all of the test inputs taken from that class. Equivalence classes are determined from the specification of a program or module. Each equivalence class is used to represent certain conditions (or predicates) on the input domain. For equivalence partitioning it is usual to also consider valid and invalid inputs. The terms input condition, valid and invalid inputs, are not used consistently. But, the following definition spells out how we will use them in this subject. An input condition on the input domain is a predicate over the values of the input domain. A *Valid* input to a program or module is an element of the input domain that is expected to return a non-error value. An *Invalid* input is an input that is expected to return an error value. Equivalence partitioning is then a systematic method for identifying interesting input conditions to be tested. An input condition can be applied to a set of values of a specific input variable, or a set of input variables as well.

A Method for Choosing Equivalence Classes

The aim is to minimize the number of test cases required to cover all of the identified equivalence classes. The following are two distinct steps in achieving this goal:

Step 1: Identify the equivalence classes

If an input condition specifies a range of values, then identify one valid equivalence class and two invalid equivalence classes.
For example, if an input condition specifies a range of values from 1 to 99, then, three equivalence classes can be identified:

- One valid equivalence class: $1 < X < 99$
- Two invalid equivalence classes $X < 1$ and $X > 99$

Step 2: Choose test cases

The next step is to generate test cases using the equivalence classes identified in the previous step. The guideline is to choose test cases on the boundaries of partitions and test cases close to the midpoint of the partition. In general, the idea is to select at least one element from each equivalence class.

Example 2: Selecting Test Cases for the Triangle Program

In this example, we will select a set of test cases for the following triangle program based on its specification. Consider the following informal specification for the Triangle Classification Program. The program reads three integer values from the standard input. The three values are interpreted as representing the lengths of the sides of a triangle. The program then prints a message to the standard output that states whether the triangle, if it can be formed, is scalene, isosceles, equilateral, or right-angled. The specification of the triangle classification program lists a number of inputs for the program as well as the form of output. Further, we require that each of the inputs *"must be*" a positive integer. Now, we can determine valid and invalid equivalence classes for the input conditions. Here, we have a range of values. If the three integers we have called x, y and z are all greater than zero, then, they are valid and we have the equivalence class.

ECvalid = f(x,y, z) x > 0 and y > 0 and z > 0.

For the invalid classes, we need to consider the case where each of the three variables in turn can be negative and so we have the following equivalence classes:

ECInvalid1 = f(x, y, z) x < 0 and y > 0 and z > 0
ECInvalid2 = f(x, y, z) x > 0 and y <0 and z > 0
ECInvalid3 = f(x, y, z) x > 0 and y > 0 and z < 0

Note that we can combine the valid equivalence classes. But, we are not allowed to combine the invalid equivalence classes. The output domain consists of the text „strings", „isosceles", „scalene", „equilateral" and „right-angled". Now, different values in the input domain map to different elements of the output domain to get the equivalence classes in *Table 7.2*. According to the equivalence partitioning method we only need to choose one element from each of the classes above in order to test the triangle program.

Table 7.2: The equivalence classes for the triangle program

Equivalence class	Test Inputs	Expected Outputs
ECscalene	f(3, 5, 7), . . . g	"Scalene"
ECisosceles	f(2, 3, 3), . . . g f(2, 3, 3), . . g	"Isosceles"
ECequilateral	f(7, 7, 7), . . . g f(7, 7, 7), . . . g	"Equilateral"
ECright angled	f(3, 4, 5), . . .	"Right Angled"
ECnon _ triangle	f(1, 1, 3), . . .	"Not a Triangle"
ECinvalid1	f(-1, 2, 3), (0, 1, 3), . . .	"Error Value"
ECinavlid2	f(1, -2, 3), (1, 0, 3), . . .	"Error Value"
ECinavlid3	f(1, 2, -3), (1, 2, 0),. . .	"Error Value"

Methods for White box testing strategies

In this approach, complete knowledge about the internal structure of the source code is required. For *White-box testing strategies,* the methods are:

1. Coverage Based Testing
2. Cyclomatic Complexity
3. Mutation Testing

Coverage based testing

The aim of coverage based testing methods is to *'cover*" the program with test cases that satisfy some fixed coverage criteria. Put another way, we choose test cases to exercise as much of the program as possible according to some criteria.

Coverage Based Testing Criteria

Coverage based testing works by choosing test cases according to well-defined „coverage" criteria. The more common coverage criteria are the following.

- **Statement Coverage or Node Coverage:** Every statement of the program should be exercised at least once.
- **Branch Coverage or Decision Coverage:** Every possible alternative in a branch or decision of the program should be exercised at least once. For *if* statements, this means that the branch must be made to take on the values *true* or *false*.
- **Decision/Condition Coverage**: Each condition in a branch is made to evaluate to both true and false and each branch is made to evaluate to both true and false.
- **Multiple condition coverage**: All possible combinations of condition outcomes within each branch should be exercised at least once.
- **Path coverage**: Every execution path of the program should be exercised at least once.

In this section, we will use the control flow graph to choose white box test cases according to the criteria above. To motivate the selection of test cases, consider the simple program given in Program 7.1.

Example 3:

```
void main(void)
{
int x1, x2, x3;
scanf("%d %d %d", &x1, &x2, &x3);
if ((x1 > 1) && (x2 == 0))
x3 = x3 / x1;
if ((x1 == 2) || (x3 > 1))
x3 = x3 + 1;
while (x1 >= 2)
x1 = x1 - 2;
printf("%d %d %d", x1, x2, x3);
}
```

Program 7.1: A simple program for white box testing

The first step in the analysis is to generate the flow chart, which is given in Figure 7.1. Now what is needed for statement coverage? If all of the branches are true, at least once, we will have executed every statement in the flow chart. Put another way to execute every statement at least once, we must execute the path ABCDEFGF. Now, looking at the conditions inside each of the three *branches*, we derive a set of constraints on the values of x1, x2 and x3 such that all the three branches are extended. A test case of the form (x1; x2; x3) = (2; 0; 3) will execute all of the statements in the program.

Note that we need not make every branch evaluate to true and false, nor have we make every condition evaluate to true and false, nor we traverse every path in the program.

To make the first branch true, we have test input (2; 0; 3) that will make all of the branches true. We need a test input that will now make each one false. Again looking at all of the conditions, the test input (1; 1; 1) will make all of the branches false.

For any of the criteria involving condition coverage, we need to look at each of the five conditions in the program: C1 = (x1>1), C2 = (x2 == 0), C3 = (x1 == 2), C4 = (x3>1) and C5 = (x1 >= 2). The test input (1; 0; 3) will make C 1 false, C2 true, C3 false, C4 true and C5 false.

Examples of sets of test inputs and the criteria that they meet are given in Table 7.3. The set of test cases meeting the multiple condition criteria is given in Table 7.4. In the table, we let the branches B1 = C1&&C2, B2 = C3||C4 and B3 = C5.

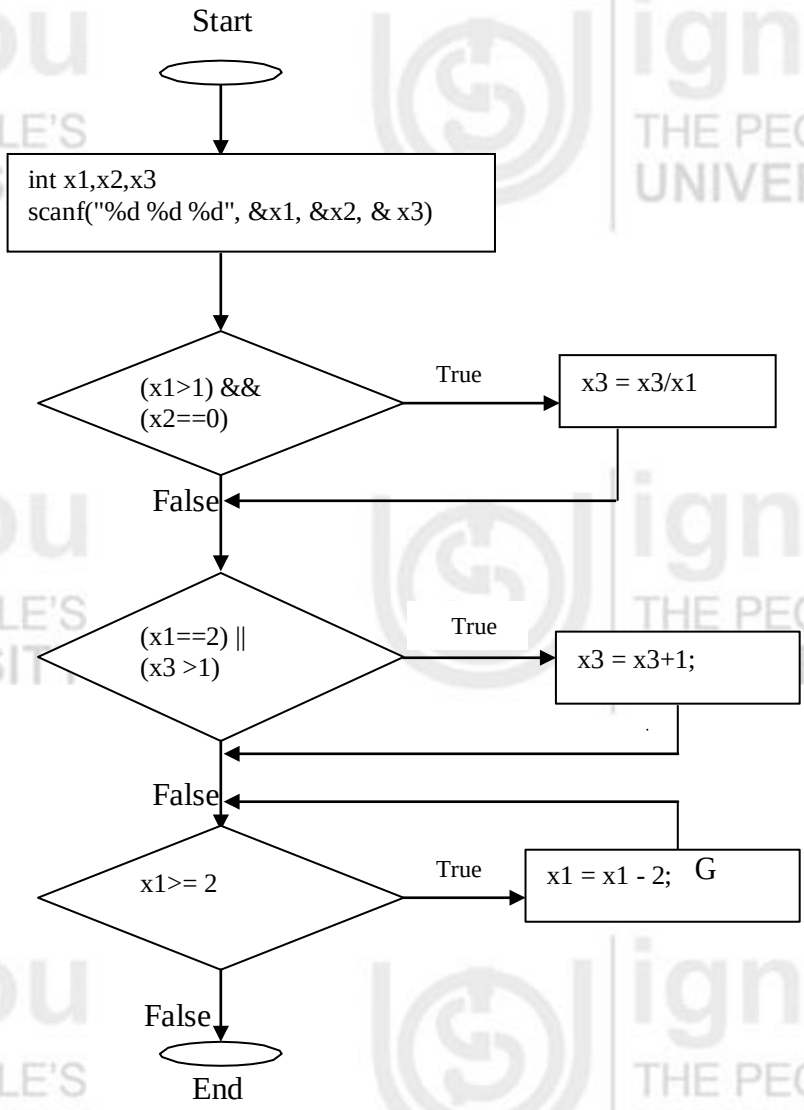

Figure 7.1: The flow chart for the program 7.1

Table 7.3: Test cases for the various coverage criteria for the program 7.1

Coverage Criteria	**Test Inputs (x1, x2, x3)**	**Execution Paths**
Statement	(2, 0, 3)	ABCDEFGF
Branch	(2, 0, 3), (1, 1, 1)	ABCDEFGF ABDF
Condition	(1, 0, 3), (2, 1, 1)	ABDEF ABDFGF
Decision/ Condition	(2, 0, 4), (1, 1, 1)	ABCDEFGF ABDF
Multiple Condition	(2, 0, 4), (2, 1, 1), (1, 0, 2), (1, 1, 1)	ABCDEFGF ABDEFGF ABDEF ABDF
Path	(2, 0, 4), (2, 1, 1), (1, 0, 2), (4, 0, 0),	ABCDEFGF ABDFGF ABDEF ABCDFGFGF

Table 4.4: Multiple condition coverage for the program in Figure 7.1

Test cases	C1 x1 > 1	C2 x2==0	B1	C3 x1==2	C4 x3 > 1	B2	B3 C5 x1 ≥ 2
(1,0,3)	F	T	F	F	T	T	F
(2,1,1)	T	F	F	T	F	F	T
(2,0,4)	T	T	T	T	T	T	T
(1,1,1)	F	F	F	F	F	F	F
(2,0,4)	T	T	T	T	T	T	T
(2,1,1)	T	F	F	T	F	T	T
(1,0,2)	F	T	F	F	T	T	F
(1,1,1)	F	F	F	F	F	F	F

7.2.3 Cyclomatic Complexity

Control flow graph (CFG)

A control flow graph describes the sequence in which different instructions of a program get executed. It also describes how the flow of control passes through the program. In order to draw the control flow graph of a program, we need to first number all the statements of a program. The different numbered statements serve as nodes of the control flow graph. An edge from one node to another node exists if the execution of the statement representing the first node can result in the transfer of control to the other node. Following structured programming constructs are represented as CFG:

Figure 7.2 : sequence

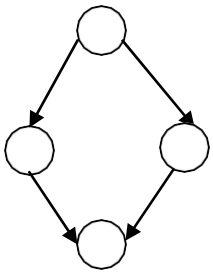

Figure 7.3 : if -else

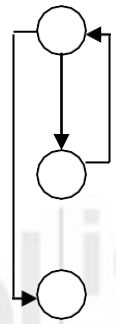

Figure 7.4 : while-loop

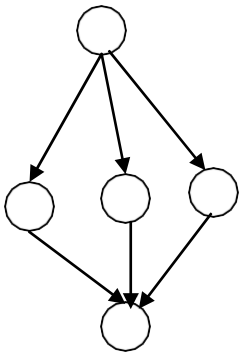

Figure 7.5: case

Example 4 : Draw CFG for the program given below.

```
int sample (a,b)
int a,b;
{
1       while (a!= b) {
2       if (a > b)
3       a = a-b;
4       else b = b-a;}
5       return a;
}
```

Program 7.2: A program

In the above program, two control constructs are used, namely, while-loop and if-then-else. A complete CFG for the program of Program 7.2 is given below: (*Figure 4.6*).

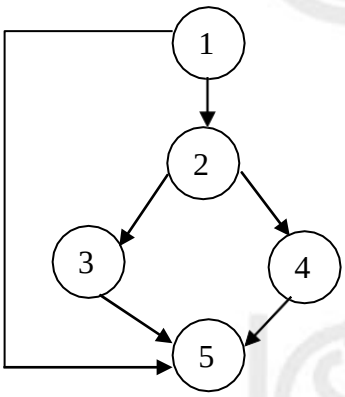

Figure 7.6: CFG for program 7.2

Cyclomatic Complexity: This technique is used to find the number of independent paths through a program. If CFG of a program is given, the Cyclomatic complexity V(G) can be computed as: V(G) = E – N + 2, where N is the number of nodes of the CFG and E is the number of edges in the CFG. For the example 4, the Cyclomatic Complexity = 6 – 5 + 2 = 3.

The following are the properties of Cyclomatic complexity:

- V(G) is the maximum number of independent paths in graph G
- Inserting and deleting functional statements to G does not affect V(G)
- G has only one path if and only if V(G) = 1.

Mutation Testing

Mutation Testing is a powerful method for finding errors in software programs. In this technique, multiple copies of programs are made, and each copy is altered; this altered copy is called a mutant. Mutants are executed with test data to determine whether the test data are capable of detecting the change between the original program and the mutated program. The mutant that is detected by a test case is termed "killed" and the goal of the mutation procedure is to find a set of test cases that are able to kill groups of mutant programs. Mutants are produced by applying mutant operators. An operator is essentially a grammatical rule that changes a single expression to another expression. It is essential that all mutants must be killed by the test cases or shown to be equivalent to the original expression. If we run a mutated program, there are two possibilities:

1. The results of the program were affected by the code change and the test suite detects it. We assumed that the test suite is perfect, which means that it must detect the change. If this happens, the mutant is called a killed mutant.
2. The results of the program are not changed and the test suite does not detect the mutation. The mutant is called an equivalent mutant.

If we take the ratio of killed mutants to all the mutants that were created, we get a number that is smaller than 1. This number gives an indication of the sensitivity of program to the changes in code. In real life, we may not have a perfect program and we may not have a perfect test suite. Hence, we can have one more scenario:

3. The results of the program are different, but the test suite does not detect it because it does not have the right test case.

Consider the following program 4.3:

```
main(argc, argv)                                    /* line 1 */
int argc;                                         /* line 2 */
char *argv[];                                       /* line 3 */
{                                                   /* line 4 */
  int c=0;                                          /* line 5 */
                                                    /* line 6 */
    if(atoi(argv[1]) < 3){                        /* line 7 */
      printf("Got less than 3\n");                  /* line 8 */
      if(atoi(argv[2]) > 5)                         /* line 9 */
        c = 2;                                    /* line 10 */
    }                                               /* line 11 */
    else                                            /* line 12 */
      printf("Got more than 3\n");                /* line 13 */
  exit(0);                                          /* line 14 */
}                                                   /* line 15 */
```

Program 7.3: A program

The program reads its arguments and prints messages accordingly.

Now let us assume that we have the following test suite that tests the program:

Test case 1:
Input: 2 4
Output: Got less than 3
Test case 2:
Input: 4 4
Output: Got more than 3
Test case 3:
Input: 4 6
Output: Got more than 3
Test case 4:
Input: 2 6
Output: Got less than 3
Test case 5:
Input: 4
Output: Got more than 3

Now, let‟s mutate the program. We can start with the following simple changes:

Mutant 1: change line 9 to the form
`if(atoi(argv[2]) <= 5)`

Mutant 2: change line 7 to the form
`if(atoi(argv[1]) >= 3)`

Mutant 3: change line 5 to the form
`int c=3;`

If we take the ratio of all the killed mutants to all the mutants generated, we get a number smaller than 1 that also contains information about accuracy of the test suite. In practice, there is no way to separate the effect that is related to test suite inaccuracy and that which is related to equivalent mutants. In the absence of other possibilities, one can accept the ratio of killed mutants to all the mutants as the measure of the test suite accuracy. The manner by which a test suite is evaluated via mutation testing is as follows: For a specific test suite and a specific set of mutants, there will be three types of mutants in the code (i) killed or dead (ii) live (iii) equivalent. The score (evaluation of test suite) associated with a test suite T and mutants M is simply computed as follows:

$$\frac{\text{\# killed Mutants}}{\text{\# total mutants - \# equivalent mutants}} \times 100$$

Check Your Progress 1

1) What is the use of Cyclomatic complexity in software development?

……

……

……

7.3 TESTING ACTIVITIES

Although testing varies between organisations, there is a cycle to testing:

Requirements Analysis: Testing should begin in the requirements phase of the software devlopment life cycle (SDLC).

Design Analysis: During the design phase, testers work with developers in determining what aspects of a design are testable and under what parameters should the testers work.

Test Planning: Test Strategy, Test Plan(s).

Test Development: Test Procedures, Test Scenarios, Test Cases, Test Scripts to use in testing software.

Test Execution: Testers execute the software based on the plans and tests and report any errors found to the development team.

Test Reporting: Once testing is completed, testers generate metrics and make final reports on their test effort and whether or not the software tested is ready for release.

Retesting the Defects: Defects are once again tested to find whether they got eliminated or not.

Levels of Testing:

Mainly, Software goes through three levels of testing:

- Unit testing
- Integration testing
- System testing.

Unit Testing

Unit testing is a procedure used to verify that a particular segment of source code is working properly. The idea about unit tests is to write test cases for all functions or methods. Ideally, each test case is separate from the others. This type of testing is mostly done by developers and not by end users.

The goal of unit testing is to isolate each part of the program and show that the individual parts are correct. Unit testing provides a strict, written contract that the piece of code must satisfy. Unit testing will not catch every error in the program. By definition, it only tests the functionality of the units themselves. Therefore, it will not catch integration errors, perfomance problems and any other system-wide issues. A unit test can only show the presence of errors; it cannot show the absence of errors.

Integration Testing

Integration testing is the phase of software testing in which individual software modules are combined and tested as a group. It follows unit testing and precedes system testing. Integration testing takes as its input, modules that have been checked out by unit testing, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output, the integrated

system ready for system testing.The purpose of Integration testing is to verify functional, performance and reliability requirements placed on major design items.

System Testing

System testing is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. As a rule, system testing takes, as its input, all of the "integrated" software components that have successfully passed integration testing and also the software system itself integrated with any applicable hardware system(s). In system testing, the entire system can be tested as a whole against the software requirements specification (SRS). There are *rules* that describe the functionality that the vendor (developer) and a customer have agreed upon. System testing tends to be more of an *investigatory* testing phase, where the focus is to have a destructive attitude and test not only the design, but also the behavior and even the believed expectations of the customer. System testing is intended to test up to and beyond the bounds defined in the software requirements specifications.

Acceptance tests are conducted in case the software developed was a custom software and not product based. These tests are conducted by customer to check whether the software meets all requirements or not. These tests may range from a few weeks to several months.

7.4 DEBUGGING

Debugging occurs as a consequence of successful testing. Debugging refers to the process of identifying the cause for defective behavior of a system and addressing that problem. In less complex terms - fixing a bug. When a test case uncovers an error, debugging is the process that results in the removal of the error. The debugging process begins with the execution of a test case. The debugging process attempts to match symptoms with cause, thereby leading to error correction. The following are two alternative outcomes of the debugging:

1. The cause will be found and necessary action such as correction or removal will be taken.
2. The cause will not be found.

Characteristics of bugs

1. The symptom and the cause may be geographically remote. That is, the symptom may appear in one part of a program, while the cause may actually be located at a site that is far removed. Highly coupled program structures exacerbate this situation.
2. The symptom may disappear (temporarily) when another error is corrected.
3. The symptom may actually be caused by non errors (e.g., round-off inaccuracies).
4. The symptom may be caused by human error that is not easily traced.
5. The symptom may be a result of timing problems, rather than processing problems.
6. It may be difficult to accurately reproduce input conditions (e.g., a real-time application in which input ordering is indeterminate).
7. The symptom may be intermittent. This is particularly common in embedded systems that couple hardware and software inextricably.
8. The symptom may be due to causes that are distributed across a number of tasks running on different processors.

Life Cycle of a Debugging Task

The following are various steps involved in debugging:

a) Defect Identification/Confirmation

- A problem is identified in a system and a defect report created
- Defect assigned to a software engineer
- The engineer analyzes the defect report, performing the following actions:
  - What is the expected/desired behaviour of the system?
  - What is the actual behaviour?
  - Is this really a defect in the system?
  - Can the defect be reproduced? (While many times, confirming a defect is straight forward. There will be defects that often exhibit quantum behaviour.)

b) Defect Analysis

Assuming that the software engineer concludes that the defect is genuine, the focus shifts to understanding the root cause of the problem. This is often the most challenging step in any debugging task, particularly when the software engineer is debugging complex software.

Many engineers debug by starting a debugging tool, generally a debugger and try to understand the root cause of the problem by following the execution of the program step-by-step. This approach may eventually yield success. However, in many situations, it takes too much time, and in some cases is not feasible, due to the complex nature of the program(s).

c) Defect Resolution

Once the root cause of a problem is identified, the defect can then be resolved by making an appropriate change to the system, which fixes the root cause.

Debugging Approaches

Three categories for debugging approaches are:

- Brute force
- Backtracking
- Cause elimination.

Brute force is probably the most popular despite being the least successful. We apply brute force debugging methods when all else fails. Using a "let the computer find the error" technique, memory dumps are taken, run-time traces are invoked, and the program is loaded with WRITE statements. *Backtracking* is a common debugging method that can be used successfully in small programs. Beginning at the site where a symptom has been uncovered, the source code is traced backwards till the error is found. In *Cause elimination,* a list of possible causes of an error are identified and tests are conducted until each one is eliminated.

☞ Check Your Progress 2

1) What are the different levels of testing and their goals? For each level specify which of the testing approaches are most suitable.

   ……..

   ……..

2) Mention the steps involved in the process of debugging.

   ……

   ……

7.5 TESTING TOOLS

The following are different categories of tools that can be used for testing:

- **Data Acquisition**: Tools that acquire data to be used during testing.
- **Static Measurement**: Tools that analyse source code without executing test cases.
- **Dynamic Measurement**: Tools that analyse source code during execution.
- **Simulation**: Tools that simulate functions of hardware or other externals.
- **Test Management**: Tools that assist in planning, development and control of testing.
- **Cross-Functional tools**: Tools that cross the bounds of preceding categories.

The following are some of the examples of commercial software testing tools:

Rational Test Real Time Unit Testing

- **Kind of Tool**

  Rational Test RealTime's Unit Testing feature automates C, C++ software component testing.

- **Organisation**

  IBM Rational Software

- **Software Description**

Rational Test RealTime Unit Testing performs black-box/functional testing, i.e., verifies that all units behave according to their specifications without regard to how that functionality is implemented. The Unit Testing feature has the flexibility to naturally fit any development process by matching and automating developers' and testers' work patterns, allowing them to focus on value-added tasks. Rational Test RealTime is integrated with native development environments (Unix and Windows) as well as with a large variety of cross-development environments.

- **Platforms**

Rational Test RealTime is available for most development and target systems including Windows and Unix.

AQtest

- **Kind of Tool**

  Automated support for functional, unit, and regression testing

- **Organisation**

  AutomatedQA Corp.

- **Software Description**

AQtest automates and manages functional tests, unit tests and regression tests, for applications written with VC++, VB, Delphi, C++Builder, Java or VS.NET. It also supports white-box testing, down to private properties or methods. External tests can be recorded or written in three scripting languages (VBScript, JScript, DelphiScript). Using AQtest as an OLE server, unit-test drivers can also run it directly from application code. AQtest automatically integrates AQtime when it is on the machine. Entirely COM-based, AQtest is easily extended through plug-ins using the complete IDL libraries supplied. Plug-ins currently support Win32 API calls, direct ADO access, direct BDE access, etc.

- **Platforms**

Windows 95, 98, NT, or 2000.

csUnit

- **Kind of Tool**

  "Complete Solution Unit Testing" for Microsoft .NET (freeware)

- **Organisation**

  csUnit.org

- **Software Description**

  csUnit is a unit testing framework for the Microsoft .NET Framework. It targets test driven development using .NET languages such as C#, Visual Basic .NET, and managed C++.

- **Platforms**

  Microsoft Windows

Sahi

http://sahi.sourceforge.net/

Software Description

Sahi is an automation and testing tool for web applications, with the facility to record and playback scripts. Developed in Java and JavaScript, it uses simple JavaScript to execute events on the browser. Features include in-browser controls, text based scripts, Ant support for playback of suites of tests, and multi-threaded playback. It supports HTTP and HTTPS. Sahi runs as a proxy server and the browser needs to use the Sahi server as its proxy. Sahi then injects JavaScript so that it can access elements in the webpage. This makes the tool independant of the website/ web application.

- **Platforms**

OS independent. Needs at least JDK1.4

7.6 SUMMARY

The importance of software testing and its impact on software is explained in this unit. Software testing is a fundamental component of software development life cycle and represents a review of specification, design and coding. The objective of testing is to have the highest likelihood of finding most of the errors within a minimum amount of time and minimal effort. A large number of test case design methods have been developed that offer a systematic approach to testing to the developer.

Knowing the specified functions that the product has been designed to perform, tests can be performed that show that each function is fully operational. A strategy for software testing may be to move upwards along the spiral. Unit testing happens at the vortex of the spiral and concentrates on each unit of the software as implemented by the source code. Testing happens upwards along the spiral to integration testing, where the focus is on design and production of the software architecture. Finally, we perform system testing, where software and other system elements are tested together.

Debugging is not testing, but always happens as a response of testing. The debugging process will have one of two outcomes:

1) The cause will be found, then corrected or removed, or

2) The cause will not be found. Regardless of the approach that is used, debugging has one main aim: to determine and correct errors. In general, three kinds of debugging approaches have been put forward: Brute force, Backtracking and Cause elimination.

7.7 SOLUTIONS / ANSWERS

Check Your Progress 1

1) Cyclomatic Complexity is asoftware metric that provides a quantitative measure of the logical complexity of a program. When it is used in the context of the basis path testing method, the value computed for Cyclomatic complexity defines the number of independent paths in the basis set of a program. It also provides an upper bound for the number of tests that must be conducted to ensure that all statements have been executed at least once.

Check Your Progress 2

1) The basic levels of testing are: unit testing, integration testing, system testing and acceptance testing.

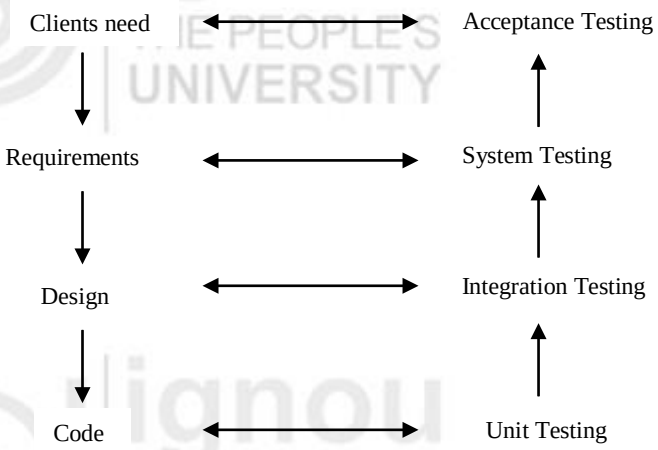

Figure 7.7: Testing levels

For unit testing, structural testing approach is best suited because the focus of testing is on testing the code. In fact, structural testing is not very suitable for large programs. It is used mostly at the unit testing level. The next level of testing is integration testing and the goal is to test interfaces between modules. With integration testing, we move slowly away from structural testing and towards functional testing. This testing activity can be considered for testing the design. The next levels are system and acceptance testing by which the entire software system is tested. These testing levels focus on the external behavior of the system. The internal logic of the program is not emphasized. Hence, mostly functional testing is performed at these levels.

2) The various steps involved in debugging are:

- Defect Identification/Confirmation
- Defect Analysis
- Defect Resolution

7.8 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.

3) An *Integrated approach to Software Engineering*, Pankaj Jalote; Narcosis Publishing House.

Reference websites

http://www.rspa.com
http://www.ieee.org
http://standards.ieee.org
http://www.ibm.com
http://www.opensourcetesting.org

UNIT 8 SOFTWARE CHANGE MANAGEMENT

Structure **Page Nos.**

8.0 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.

8.1 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.

8.2 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 8.1 depicts a baseline for design specification.

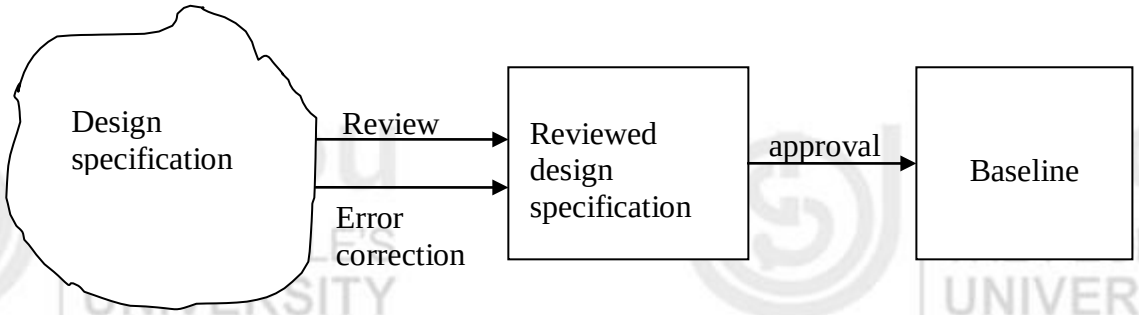

Figure 8.1 : A baseline for design specification

Figure 8.2 depicts the evolution of a baseline.

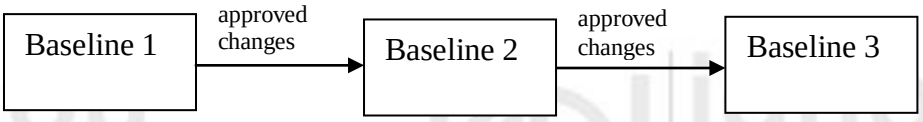

Figure 8.2: Evolution of a baseline

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 8.3 depicts the process of changes to a baseline.

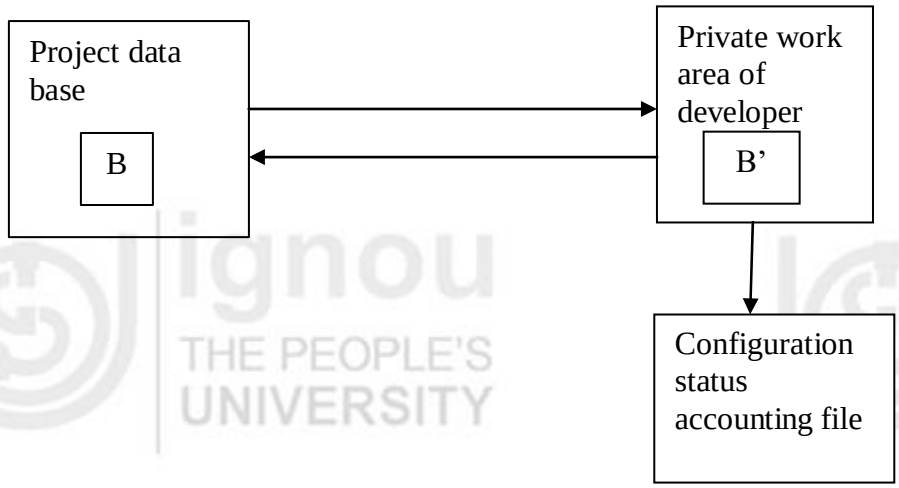

Figure 8.3: Process of changes to baseline

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?

……...

……...

8.3 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.

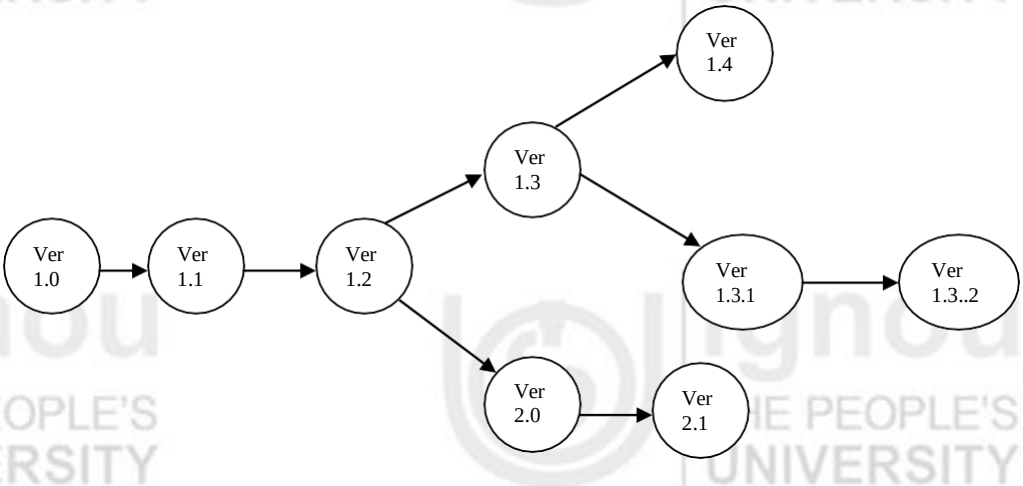

Figure 8.4 : An evolutionary graph for a different version of an item

The above evolutionary graph(*Figure 8.4*) 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 8.5 depicts the changes to a baselined file in project database.

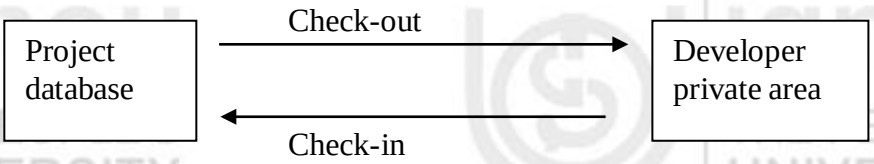

Figure 8.5: Changes to a baselined file in project database

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 8.6*).

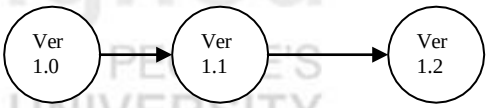

Figure 8.6: Version tree of welcome.htm

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?

……

……...

8.4 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.

8.5 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.

A check list for software configuration audit

- 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)?

……

……

8.6 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.

8.7 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.

8.8 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