

RAGHU INSTITUTE OF TECHNOLOGY

Dakamarri (v), Bheemunipatnam (M)

Visakhapatnam Dist, Andhra Pradesh, PIN-531162

(Approved by AICTE, New Delhi, and Affiliated to Jawaharlal Nehru Technological University: Kakinada (AP))



III B. Tech., CSE II - Semester

FACULTY LABORATORY MANUAL

For

SOFTWARE ENGINEERING

Prepared by

Ch NarayanaRao., Assistant Professor

DEPARTMENT OF

COMPUTER SCIENCE AND ENGINEERING

RAGHU INSTITUTE OF TECHNOLOGY

(Affiliated to JNTU-KAKINADA)
Visakhapatnam-531162



CERTIFICATE

Name of the Laboratory : SOFTWARE ENGINEERING

Name of the Faculty : Ch NarayanaRao

Department : CSE

Program : B.TECH

Year : III

Semester : II

IQAC Members:

Name(s):

Signature(s):

HOD

CONTENTS

S.NO	PROGRAM NO.	DESCRIPTION	Page No.
1		LAB OBJECTIVE	4
2		UNIVERSITY / JNTU Syllabus	5
3		List of Programs	5
4		SCHEDULE/CYCLE CHART	5
5	1	Do the Requirement Analysis and Prepare SRS.	6
6	2	Using COCOMO model estimate effort.	15
7	3	Calculate effort using FP (Function Point) oriented estimation model.	23
8	4	Analyze the risk related to the project and prepare RMMM (Risk Mitigation, Monitoring, and Management) plan.	25
9	5	Develop time line chart and project table using PERT (Program Evaluation Review Technique) or CPM (critical path method) project scheduling methods	27
10	6	Draw E-R Diagrams, DFD, CFD and structured charts for the project	28
11	7	Design of test cases based on requirement and design	36
12	8	Prepare FTR(formal technical reviews)	44
13	9	Prepare Version control and change control for software configuration items	44
14	10	REFERENCE	46

COURSE OBJECTIVE

The software Engineering lab will facilitate the students to develop a preliminary yet practical understanding of software development process and tools

At the end of this lab session:

1. To be able to understand different software methodologies.
2. Understanding of Requirement analysis and SRS.
3. Able to study ER Diagrams, DFD, CFD.
4. Design of test cases based on requirement and design.

CO to PO mapping:

Course Name	CO	PO											
		a	b	c	d	e	f	g	h	i	j	k s	l
Software Engg.	Identify and build an appropriate process model for a given project.	✓				✓							
	Analyze the principles at various phases of software development.		✓									✓	
	Translate a specification into a design, and identify the components to build The architecture for a given problem, all using an appropriate software engineering Methodology.			✓									
	Define a project management plan and tabulate appropriate testing plans at different levels during the development of the software.												✓

LIST OF EXPERIMENTS

1. Do the Requirement Analysis and Prepare SRS (Software Requirement Specification).
2. Using COCOMO (Constructive Cost Model) model estimate effort.
3. Calculate effort using FP (Function point) oriented estimation model.
4. Analyze the risk related to the project and prepare RMMM (Risk Mitigation, Monitoring, and Management) plan.
5. Develop time line chart and project table using PERT (Program Evaluation Review Technique) or CPM (critical path method) project scheduling methods.
6. Draw E-R (Entity Relationship) Diagrams, DFD(Dataflow Diagrams), CFD(Control Flow Diagrams) and structured charts for the project
7. Design of test cases based on requirement and design
8. Prepare FTR(formal technical reviews)
9. Prepare Version control and change control for software configuration items

Lab Incharge

HOD.

LIST OF PRESCRIBED EXPERIMENTS

TASK1:

1. Do the Requirement Analysis and Prepare SRS.

Introduction:

Requirements identification is the first step of any software development project. Until the requirements of a client have been clearly identified, and verified, no other task (design, coding, testing) could begin. Usually business analysts having domain knowledge on the subject matter discuss with clients and decide what features are to be implemented.

In this experiment we will learn how to identify functional and non-functional requirements from a given problem statement. Functional and non-functional requirements are the primary components of a Software Requirements Specification.

Theory:

Objectives:

After completing this experiment you will be able to:

Identify ambiguities, inconsistencies and incompleteness from a requirements specification

Identify and state functional requirements

Identify and state non-functional requirements

Steps for conducting the experiment:

General Instructions: Follow are the steps to be followed in general to perform the experiments in Software Engineering Virtual Lab.

Read the theory about the experiment

View the simulation provided for a chosen, related problem

Take the self evaluation to judge your understanding (optional, but recommended)

Solve the given list of exercises

Experiment Specific Instructions

Following are the instructions specifically for this experiment:

From the given problem statement, try to figure out if there's any inconsistency with the requirement specification

Also, try to determine what are the functional and non-functional requirements are. Select the check boxes accordingly, and then click on the 'Submit' button.

Requirements:

Somerville defines "requirement" as a specification of what should be implemented. Requirements specify how the target system should behave. It specifies what to do, but not how to do. Requirements engineering refers to the process of understanding what a customer expects from the system to be developed, and to document them in a standard and easily readable and understandable format. This documentation will serve as reference for the subsequent design, implementation and verification of the system.

It is necessary and important that before we start planning, design and implementation of the software system for our client, we are clear about its requirements. If we don't have a clear vision of what is to be developed and what all features are expected, there would be serious problems and customer dissatisfaction as well.

Characteristics of Requirements:

Requirements gathered for any new system to be developed should exhibit the following three properties:

Unambiguity: There should not be any ambiguity what a system to be developed should do. For example, consider you are developing a web application for your client. The client requires that enough number of people should be able to access the application simultaneously. What's the "enough number of people"? That could mean 10 to you, but, perhaps, 100 to the client. There's an ambiguity.

Consistency: To illustrate this, consider the automation of a nuclear plant. Suppose one of the clients say that if the radiation level inside the plant exceeds R1, all reactors should be shut down. However, another person from the client side suggests that the threshold radiation level should be R2. Thus, there is an inconsistency between the two end users regarding what they consider as threshold level of radiation.

Completeness: A particular requirement for a system should specify what the system should do and also what it should not. For example, consider software to be developed for ATM. If a customer enters an amount greater than the maximum permissible withdrawal amount, the ATM should display an error message, and it should not dispense any cash.

Categorization of Requirements:

Based on the target audience or subject matter, requirements can be classified into different types, as stated below:

User requirements: They are written in natural language so that both customers can verify their requirements have been correctly identified

System requirements: They are written involving technical terms and/or specifications, and are meant for the development or testing teams

Requirements can be classified into two groups based on what they describe:

Functional requirements (FRs): These describe the functionality of a system -- how a system should react to a particular set of inputs and what should be the corresponding output.

Non-functional requirements (NFRs): They are not directly related what functionalities are expected from the system. However, NFRs could typically define how the system should behave under certain situations. For example, a NFR could say that the system should work with 128MB RAM. Under such condition, a NFR could be more critical than a FR.

Non-functional requirements could be further classified into different types like:

Product requirements: For example, a specification that the web application should use only plain HTML, and no frames

Performance requirements: For example, the system should remain available 24x7

Organizational requirements: The development process should comply with SEI CMM level 4.

Functional Requirements:

Identifying Functional Requirements

Given a problem statement, the functional requirements could be identified by focusing on the following points:

Identify the high level functional requirements simply from the conceptual understanding of the problem. For example, a Library Management System, apart from anything else, should be able to issue and return books.

Identify the cases where an end user gets some meaningful work done by using the system. For example, in a digital library a user might use the "Search Book" functionality to obtain information about the books of his interest.

If we consider the system as a black box, there would be some inputs to it, and some output in return. This black box defines the functionalities of the system. For example, to search for a book, user gives title of the book as input and gets the book details and location as the output.

Any high level requirement identified could have different sub-requirements. For example, "Issue Book" module could behave differently for different class of users, or for a particular user who has issued the book thrice consecutively.

Preparing Software Requirements Specifications:

Once all possible FRs and non-FRs have been identified, which are complete, consistent, and non-ambiguous, the Software Requirements Specification (SRS) is to be prepared. IEEE provides a template [iv], also available here, which could be used for this purpose. The SRS is prepared by the service provider, and verified by its client. This document serves as a legal agreement between the client and the service provider. Once the concerned system has been developed and deployed, and a proposed feature was not found to be present in the system, the client can point this out from the SRS. Also, if after delivery, the client says a new feature is

required, which was not mentioned in the SRS, the service provider can again point to the SRS. The scope of the current experiment, however, doesn't cover writing a SRS.

Simulation:

We show here how to extract functional requirements when a problem statement is given. The case under study is a online voting system.

Internet has led to discussion of e-democracy and online voting. Many people think that the internet could replace representative democracy, enabling everyone to vote on everything and anything by online voting. Online voting could reduce cost and make voting more convenient. This type of voting can be done for e-democracy, or it may be used for finalizing a solution, if many alternatives are present. Online voting makes use of authentication, hence it needs security, and the system must be able to address obtaining, marking, delivering and counting ballots via computer. Advantage of online voting is it could increase voter turnout because of convenience, and it helps to reduce fraud voting.

User registration: A candidate or a user at the internet wishes to vote online has to register with the system by providing his details. His registration has to be verified by admin.

User login: A user has to login in order to vote. A candidate can login and view candidate specific details.

Publish manifesto: A candidate can login and publish his election manifesto

Vote: A user can cast his vote in favour of a particular candidate

Internet has led to discussion of e-democracy and online voting. Many people think that the internet could replace representative democracy, enabling everyone to vote on everything and anything by online voting. Online voting could reduce cost and make voting more convenient. This type of voting can be done for a solution, if many alternatives are present. Online voting makes use of authentication, hence it needs security, and the system must be able to address obtaining, marking, delivering and counting ballots via computer. Advantage of online voting is it could increase voter turnout because of convenience, and helps to reduce fraud voting.

Count votes: System must be able to count votes received by each candidate based on polling

Internet has led to discussion of e-democracy and online voting. Many people think that the internet could replace representative democracy, enabling everyone to vote on everything and anything by online voting. Online voting could reduce cost and make voting more convenient. This type of voting can be done for a solution, if many alternatives are present. Online voting makes use of authentication, hence it needs security, and the system must be able to address obtaining, marking, delivering and counting ballots via computer. Advantage of online voting is it could increase voter turnout because of convenience, and helps to reduce fraud voting.

Publish results: Administrator can publish polling results at designated time

Internet has led to discussion of e-democracy and online voting. Many people think that the internet could replace representative democracy, enabling everyone to vote on everything and anything by online voting. Online voting could reduce cost and make voting more convenient. This type of voting can be done for a solution, if many alternatives are present. Online voting makes use of authentication, hence it needs security, and the system must be able to address obtaining, marking, delivering and counting ballots via computer. Advantage of online voting is it could increase voter turnout because of convenience, and helps to reduce fraud voting.

Only one vote: A valid user can vote only once

Only valid users: Only valid users can participate in

Case Study:

A Library Information System

As the size and capacity of the institute is increasing with the time, it has been proposed to develop a Library Information System (LIS) for the benefit of students and employees of the institute. LIS will enable the members to borrow a book (or return it) with ease while sitting at his desk/chamber. The system also enables a member to extend the date of his borrowing if no other booking for that particular book has been made. For the library staff, this system aids them to easily handle day-to-day book transactions. The librarian, who has administrative privileges and complete control over the system, can enter a new record into the system when a new book has been purchased, or remove a record in case any book is taken off the shelf. Any non-member is free to use this system to browse/search books online. However, issuing or returning books is restricted to valid users (members) of LIS only.

The final deliverable would be a web application (using the recent HTML 5), which should run only within the institute LAN. Although this reduces security risk of the software to a large extent, care should be taken no confidential information (eg., passwords) is stored in plain text.

Identification of functional requirements

The above problem statement gives a brief description of the proposed system. From the above, even without doing any deep analysis, we might easily identify some of the basic functionality of the system:

- **New user registration:** Any member of the institute who wishes to avail the facilities of the library has to register himself with the Library Information System. On successful registration, a user ID and password would be provided to the member. He has to use this credentials for any future transaction in LIS.
- **Search book:** Any member of LIS can avail this facility to check whether any particular book is present in the institute's library. A book could be searched by its:
 - Title
 - Authors name
 - Publisher's name

User login: A registered user of LIS can login to the system by providing his employee ID and password as set by him while registering. After successful login, "Home" page for the user is shown from where he can access the different functionalities of LIS: search book, issue book, return book, reissue book. Any employee ID not registered with LIS cannot access the "Home" page -- a login failure message would be shown to him, and the login dialog would appear again. This same thing happens when any registered user types in his password wrong. However, if incorrect password has been provided for three time consecutively, the security question for the user (specified while registering) with an input box to answer it are also shown. If the user can answer the security question correctly, a new password would be sent to his email address. In case the user fails to answer the security question correctly, his LIS account would be blocked. He needs to contact with the administrator to make it active again.

Issue book: Any member of LIS can issue a book against his account provided that:

- The book is available in the library i.e. could be found by searching for it in LIS
- No other member has currently issued the book
- Current user has not issued the maximum number of books that can

If the above conditions are met, the book is issued to the member. Note that this FR would remain **incomplete** if the "maximum number of books that can be issued to a member" is not defined. We assume that this number has been set to four for students and research scholars, and to ten for professors. Once a book has been successfully issued, the user account is updated to reflect the same.

Return book: A book is issued for a finite time, which we assume to be a period of 20 days. That is, a book once issued should be returned within the next 20 days by the corresponding member of LIS. After successful return of a book, the user account is updated to reflect the same.

Reissue book: Any member who has issued a book might find that his requirement is not over by 20 days. In that case, he might choose to reissue the book, and get the permission to keep it for another 20 days. However, a member can reissue any book at most twice, after which he

has to return it. Once a book has been successfully reissued, the user account is updated to reflect the information.

In a similar way we can list other functionality offered by the system as well. However, certain features might not be evident directly from the problem system, but which, nevertheless, are required. One such functionality is "User Verification". The LIS should be able to judge between a registered and non-registered member. Most of the functionality would be available to a registered member. The "New User Registration" would, however, be available to non-members. Moreover, an already registered user shouldn't be allowed to register himself once again.

Having identified the (major) functional requirements, we assign an identifier to each of them for future reference and verification. Following table shows the list:

Table 01: Identifier and priority for software requirements

#	Requirement	Priority
R1	New user registration	High
R2	User Login	High
R3	Search book	High
R4	Issue book	High
R5	Return book	High
R6	Reissue book	Low

Identification of non-functional requirements

Having talked about functional requirements, let's try to identify a few non-functional requirements.

- **Performance Requirements:**

- This system should remain accessible 24x7
- At least 50 users should be able to access the system altogether at any given time

- **Security Requirements:**

- This system should be accessible only within the institute LAN
- The database of LIS should not store any password in plain text -- a hashed value has to be stored

- **Software Quality Attributes**

- **Database Requirements**

- **Design Constraints:**

- The LIS has to be developed as a web application, which should work with Firefox 5, Internet Explorer 8, Google Chrome 12, Opera 10
- The system should be developed using HTML 5

Once all the functional and non-functional requirements have been identified, they are documented formally in SRS, which then serves as a legal agreement

Viva Questions:

Q. What is feasibility study?

A. It is a measure to assess how practical and beneficial the software project development will be for an organization. The software analyzer conducts a thorough study to understand economic, technical and operational feasibility of the project.

Economic - Resource transportation, cost for training, cost of additional utilities and tools and overall estimation of costs and benefits of the project.

Technical - Is it possible to develop this system ? Assessing suitability of machine(s) and operating system(s) on which software will execute, existing developers' knowledge and skills, training, utilities or tools for project.

Operational - Can the organization adjust smoothly to the changes done as per the demand of project ? Is the problem worth solving?

Q. How can you gather requirements?

A. Requirements can be gathered from users via interviews, surveys, task analysis, brainstorming, domain analysis, prototyping, studying existing usable version of software, and by observation.

Q. What is SRS?

A. SRS or Software Requirement Specification is a document produced at the time of requirement gathering process. It can be also seen as a process of refining requirements and documenting them.

Q. What are functional requirements?

A. Functional requirements are functional features and specifications expected by users from the proposed software product.

Q. What are non-functional requirements?

A. Non-functional requirements are implicit and are related to security, performance, look and feel of user interface, interoperability, cost etc.

Q. What is computer software?

A. Computer software is a complete package, which includes software program, its documentation and user guide on how to use the software.

Q. Can you differentiate computer software and computer program?

A. A computer program is piece of programming code which performs a well defined task where as software includes programming code, its documentation and user guide.

Q. What is software engineering?

A. Software engineering is an engineering branch associated with software system development.

Q. When you know programming, what is the need to learn software engineering concepts?

A. A person who knows how to build a wall may not be good at building an entire house. Likewise, a person who can write programs may not have knowledge of other concepts of Software Engineering. The software engineering concepts guide programmers on how to assess requirements of end user, design the algorithms before actual coding starts, create programs by coding, testing the code and its documentation.

Q. What is software process or Software Development Life Cycle (SDLC)?

A. Software Development Life Cycle, or software process is the systematic development of software by following every stage in the development process namely, Requirement Gathering, System Analysis, Design, Coding, Testing, Maintenance and Documentation in that order.

Q. What are SDLC models available?

A. There are several SDLC models available such as Waterfall Model, Iterative Model, Spiral model, V-model and Big-bang Model etc.

Q. What are various phases of SDLC?

A. The generic phases of SDLC are: Requirement Gathering, System Analysis and Design, Coding, Testing and implementation. The phases depend upon the model we choose to develop software.

Q. Which SDLC model is the best?

A. SDLC Models are adopted as per requirements of development process. It may vary software-to-software to ensuring which model is suitable.

We can select the best SDLC model if following answers are satisfied -

Is SDLC suitable for selected technology to implement the software ?

Is SDLC appropriate for client's requirements and priorities ?

Is SDLC model suitable for size and complexity of the software ?

Is the SDLC model suitable for type of projects and engineering we do ?

Is the SDLC appropriate for the geographically co-located or dispersed developers ?

Task2:

Aim: Using COCOMO model estimate effort

Introduction:

After gathering the entire requirements specific to software project usually we need to think about different solution strategy for the project. Expert business analysts are analyzing their benefits and as well as their shortcomings by means of cost, time and resources require developing it.

In this experiment, we will learn how to estimate cost, effort and duration for a software project, and then select one solution approach which will be found suitable to fulfil the organizational goal.

Objectives:

Categorize projects using COCOMO, and estimate effort and development time required for a project

Estimate the program complexity and effort required to recreate it using Halstead's metrics.

Project Estimation Techniques:

A software project is not just about writing a few hundred lines of source code to achieve a particular objective. The scope of a software project is comparatively quite large, and such a project could take several years to complete. However, the phrase "quite large" could only give some (possibly vague) qualitative information. As in any other science and engineering discipline, one would be interested to measure how complex a project is. One of the major activities of the project planning phase, therefore, is to estimate various project parameters in order to take proper decisions. Some important project parameters that are estimated include:

Project size: What would be the size of the code written say, in number of lines, files, modules?

Cost: How much would it cost to develop a software? A software may be just pieces of code, but one has to pay to the managers, developers, and other project personnel.

Duration: How long would it be before the software is delivered to the clients?

Effort: How much effort from the team members would be required to create the software?

In this experiment we will focus on two methods for estimating project metrics: COCOMO and Halstead's method.

COCOMO:

COCOMO (Constructive Cost Model) was proposed by Boehm. According to him, there could be three categories of software projects: organic, semidetached, and embedded. The classification is done considering the characteristics of the software, the development team and environment. These product classes typically correspond to application, utility and system programs, respectively. Data processing programs could be considered as application programs. Compilers, linkers, are examples of utility programs. Operating systems, real-time system programs are examples of system programs. One could easily apprehend that it would take much more time and effort to develop an OS than an attendance management system.

The concept of organic, semidetached, and embedded systems are described below.

Organic: A development project is said to be of organic type, if

The project deals with developing a well understood application

The development team is small

The team members have prior experience in working with similar types of projects

Semidetached: A development project can be categorized as semidetached type, if

The team consists of some experienced as well as inexperienced staff

Team members may have some experience on the type of system to be developed

Embedded: Embedded type of development project are those, which

Aims to develop software strongly related to machine hardware

Team size is usually large

Boehm suggested that estimation of project parameters should be done through three stages: Basic COCOMO, Intermediate COCOMO, and Complete COCOMO.

Basic COCOMO Model

The basic COCOMO model helps to obtain a rough estimate of the project parameters. It estimates effort and time required for development in the following way:

$$\text{Effort} = a * (\text{KDSI})^b \text{ PM}$$

$$\text{Tdev} = 2.5 * (\text{Effort})^c \text{ Months}$$

where

KDSI is the estimated size of the software expressed in Kilo Delivered Source Instructions

a, b, c are constants determined by the category of software project

Effort denotes the total effort required for the software development, expressed in person months (PMs)

Tdev denotes the estimated time required to develop the software (expressed in months)

The value of the constants a, b, c are given below:

Software project	a	b	c
Organic	2.4	1.05	0.38

Semi-detached 3.0 1.12 0.35

Embedded 3.6 1.20 0.32

Intermediate COCOMO Model

The basic COCOMO model considers that effort and development time depends only on the size of the software. However, in real life there are many other project parameters that influence the development process. The intermediate COCOMO take those other factors into consideration by defining a set of 15 cost drivers (multipliers) as shown in the table below [i]. Thus, any project that makes use of modern programming practices would have lower estimates in terms of effort and cost. Each of the 15 such attributes can be rated on a six-point scale ranging from "very low" to "extra high" in their relative order of importance. Each attribute has an effort multiplier fixed as per the rating. The product of effort multipliers of all the 15 attributes gives the Effort Adjustment Factor (EAF).

Cost drivers for INTERmediate COCOMO (Source: <http://en.wikipedia.org/wiki/COCOMO>)

Cost Drivers Ratings

Very Low	Low	Nominal	High	Very High	Extra High
Product attributes					
Required software reliability	0.75	0.88	1.00	1.15	1.40
Size of application database		0.94	1.00	1.08	1.16
Complexity of the product	0.70	0.85	1.00	1.15	1.30 1.65
Hardware attributes					
Run-time performance constraints				1.00	1.11 1.30 1.66
Memory constraints		1.00	1.06	1.21	1.56
Volatility of the virtual machine environment					0.87 1.00 1.15 1.30
Required turnabout time		0.87	1.00	1.07	1.15
Personnel attributes					
Analyst capability	1.46	1.19	1.00	0.86	0.71
Applications experience	1.29	1.13	1.00	0.91	0.82
Software engineer capability	1.42	1.17	1.00	0.86	0.70
Virtual machine experience	1.21	1.10	1.00	0.90	
Programming language experience	1.14	1.07	1.00	0.95	
Project attributes					
Application of software engineering methods				1.24	1.10 1.00 0.91 0.82
Use of software tools	1.24	1.10	1.00	0.91	0.83
Required development schedule		1.23	1.08	1.00	1.04 1.10

EAF is used to refine the estimates obtained by basic COCOMO as follows:

Effort_{corrected} = Effort * EAF

Tdev_{corrected} = 2.5 * (Effort_{corrected})^c

both the basic and intermediate COCOMO models consider a software to be a single homogeneous entity -- an assumption, which is rarely true. In fact, many real life applications are made up of several smaller sub-systems. (One might not even develop all the sub-systems -- just use the available services). The complete COCOMO model takes these factors into account to provide a far more accurate estimate of project metrics

To illustrate this, consider a very popular distributed application: the ticket booking system of the Indian Railways. There are computerized ticket counters in most of the railway stations of our country. Tickets can be booked / cancelled from any such counter. Reservations for future tickets, cancellation of reserved tickets could also be performed. On a high level, the ticket booking system has three main components:

Database

Graphical User Interface (GUI)

Networking facilities

Advantages of COCOMO Model:

COCOMO is a simple model, and should help one to understand the concept of project metrics estimation.

Drawbacks of COCOMO

COCOMO uses KDSI, which is not a proper measure of a program's size. Indeed, estimating the size of software is a difficult task, and any slight miscalculation could cause a large deviation in subsequent project estimates. Moreover, COCOMO was proposed in 1981 keeping the waterfall model of project life cycle in mind [2]. It fails to address other popular approaches like prototype, incremental, spiral, agile models. Moreover, in present day a software project may not necessarily consist of coding of every bit of functionality. Rather, existing software components are often used and glued together towards the development of a new software. COCOMO is not suitable in such cases.

COCOMO II was proposed later in 2000 to many of address these issues.the concept of project metrics estimation.

Halstead's Complexity Metrics:

Halstead took a linguistic approach to determine the complexity of a program. According to him, a computer program consists of a collection of different operands and operators. The definition of operands and operators could, however, vary from one person to another and one programming language to other. Operands are usually the implementation variables or constants -- something upon which an operation could be performed. Operators are those symbols that affect the value of operands. Halstead's metrics are computed based on the operators and operands used in a computer program. Any given program has the following four parameters:

n1: Number of unique operators used in the program

n_2 : Number of unique operands used in the program

N_1 : Total number of operators used in the program

N_2 : Total number of operands used in the program

Using the above parameters one compute the following metrics:

Program Length: $N = N_1 + N_2$

Program Vocabulary: $n = n_1 + n_2$

Volume: $V = N * \lg n$

Difficulty: $D = (n_1 * N_2) / (2 * n_2)$

Effort: $E = D * V$

Time to Implement: $T = E / 18$ (in seconds) [vi]

The program volume V is the minimum number of bits needed to encode the program. It represents the size of the program while taking into account the programming language.

The difficulty metric indicates how difficult a program is to write or understand.

Effort denotes the "mental effort" required to develop the software, or to recreate the same in another programming language [iv]. COCOMO is a simple model, and should help one to understand the concept of project metrics estimation. Advantages of COCOMO

COCOMO is a simple model, and should help one to understand the concept of project metrics estimation.

Using Basic COCOMO model to estimate project parameters

Use the simulator on the right hand side to understand how project type and size affects the different parameters estimated.

Quick glance at the formulae:

- **Effort:** $a * (\text{Size})^b$ person-month
- **Time for development:** $2.5 * (\text{Effort})^c$ month

Drag the slider to change the project size. Note: select the nearest discrete value corresponding to the actual size.

Project Type	a	b	c
Organic	2.4	1.05	0.38
Project size (in KLOC)	2		
Effort (in PM)			
T_{dev} (in month)			
# of developers			

As evident from the simulation parameters, size of a semi-detached project is larger than that of an organic project, and size of an embedded project is larger than that of a semi-detached, and thereby affecting factors like effort and development time.

A Library Information System for SE VLabs Institute:

The SE VLabs Institute has been recently setup to provide state-of-the-art research facilities in the field of Software Engineering. Apart from research scholars (students) and professors, it also includes quite a large number of employees who work on different projects undertaken by the institution.

As the size and capacity of the institute is increasing with the time, it has been proposed to develop a Library Information System (LIS) for the benefit of students and employees of the institute. LIS will enable the members to borrow a book (or return it) with ease while sitting at

his desk/chamber. The system also enables a member to extend the date of his borrowing if no other booking for that particular book has been made. For the library staff, this system aids them to easily handle day-to-day book transactions. The librarian, who has administrative privileges and complete control over the system, can enter a new record into the system when a new book has been purchased, or remove a record in case any book is taken off the shelf. Any non-member is free to use this system to browse/search books online. However, issuing or returning books is restricted to valid users (members) of LIS only.

The final deliverable would be a web application (using the recent HTML 5), which should run only within the institute LAN. Although this reduces security risk of the software to a large extent, care should be taken no confidential information (eg., passwords) is stored in plain text.

The SE VLabs Institute has a IT management team of it's own. This team has been given the task to execute the Library Information System project. The team consists of a few experts from industry, and a batch of highly qualified engineers experienced with design and implementation of information systems. It is planned that the current project will be undertaken by a small team consisting of one expert and few engineers. Actual team composition would be determined in a later stage.

Using COCOMO and based on the team size (small) and experience (high), the concerned project could be categorized as "organic". The experts, based on their prior experience, suggested that the project size could roughly be around 10 KLOC. This would serve as the basis for estimation of different project parameters using basic COCOMO, as shown below:

$$\text{Effort} = a * (\text{KLOC})^b \text{ PM}$$

$$\text{Tdev} = 2.5 * (\text{Effort})^c \text{ Months}$$

For organic category of project the values of a, b, c are 2.4, 1.05, 0.38 respectively. So, the projected effort required for this project becomes

$$\text{Effort} = 2.4 * (10)^{1.05} \text{ PM}$$

$$= 27 \text{ PM (approx)}$$

So, around 27 person-months are required to complete this project. With this calculated value for effort we can also approximate the development time required:

$$\text{Tdev} = 2.5 * (27)^{0.38} \text{ Months}$$

$$= 8.7 \text{ Months (approx)}$$

So, the project is supposed to be complete by nine months. However, estimations using basic COCOMO are largely idealistic. Let us refine them using intermediate COCOMO. Before doing

so we determine the Effort Adjustment Factor (EAF) by assigning appropriate weight to each of the following attributes.

Cost Drivers Ratings

Very Low Low Nominal High Very High Extra High

Product attributes

Required software reliability 0.75 0.88 1.00 1.15 1.40

Size of application database 0.94 1.00 1.08 1.16

Complexity of the product 0.70 0.85 1.00 1.15 1.30 1.65

Hardware attributes

Run-time performance constraints 1.00 1.11 1.30 1.66

Memory constraints 1.00 1.06 1.21 1.56

Volatility of the virtual machine environment 0.87 1.00 1.15 1.30

Required turnabout time 0.87 1.00 1.07 1.15

Personnel attributes

Analyst capability 1.46 1.19 1.00 0.86 0.71

Applications experience 1.29 1.13 1.00 0.91 0.82

Software engineer capability 1.42 1.17 1.00 0.86 0.70

Virtual machine experience 1.21 1.10 1.00 0.90

Programming language experience 1.14 1.07 1.00 0.95

Project attributes

Application of software engineering methods 1.24 1.10 1.00 0.91 0.82

Use of software tools 1.24 1.10 1.00 0.91 0.83

Required development schedule 1.23 1.08 1.00 1.04 1.10

The cells with yellow backgrounds highlight our choice of weight for each of the cost drivers. EAF is determined by multiplying all the chosen weights. So, we get

$$\text{EAF} = 0.53 \text{ (approx)}$$

Using this EAF value we refine our estimates from basic COCOMO as shown below

$$\text{Effort}_{\text{corrected}} = \text{Effort} * \text{EAF}$$

$$= 27 * 0.53$$

$$= 15 \text{ PM (approx)}$$

$$\text{Tdev}_{\text{corrected}} = 2.5 * (\text{Effort}_{\text{corrected}})^{0.38}$$

$$= 2.5 * (15)^{0.38}$$

$$= 7 \text{ months (approx)}$$

After refining our estimates it seems that seven months would likely be sufficient for completion of this project. This is still a rough estimate since we have not taken the underlying components of the software into consideration. Complete COCOMO model considers such parameters to give a more realistic estimate.

Viva Questions:

Q. What are software project estimation techniques available?

A. There are many estimation techniques available. The most widely used are -

Decomposition technique (Counting Lines of Code and Function Points)

Empirical technique (Putnam and COCOMO).

Q. What is COCOMO model?

COCOMO (COConstructive COSt Model) has been designed in 1981 by Barry Boehm to give an estimate of the number of man-months it will take to develop a software product and it is referred as COCOMO 81

A new model called COCOMO II was designed in 1990 and the need for this model came up as software development technology moved from mainframe and overnight batch processing to desktop development, code re-usability and the use of off-the-shelf software components.

Q. What is software scope?

A. Software scope is a well-defined boundary, which encompasses all the activities that are done to develop and deliver the software product.

The software scope clearly defines all functionalities and artifacts to be delivered as a part of the software. The scope identifies what the product will do and what it will not do, what the end product will contain and what it will not contain.

Q. What is project estimation?

A. It is a process to estimate various aspects of software product in order to calculate the cost of development in terms of efforts, time and resources. This estimation can be derived from past experience, by consulting experts or by using pre-defined formulas.

Q. How can we derive the size of software product?

A. Size of software product can be calculated using either of two methods –

1. Counting the lines of delivered code
2. Counting delivered function points

Task3.

Aim: Calculate effort using FP oriented estimation model.

Objective: To study software estimation in early stages of software development.

References: Software Engineering Roger Pressman McGraw Hill Fifth edition Software Engineering Ian Somerville Pearson Education Sixth edition An Integrated Approach To Software Engineering Pankaj Jalote Narosa

Pre-requisite: Knowledge of project metrics and software measurement

Theory:

1. Compute the count-total which will be used to define the complexity of a project. (count_total)
2. Find the complexity adjustment values based on responses to the 14 questions ($\sum Fi$)
3. $FP = count_total[0.65 + 0.01 * \sum Fi]$

Sample Output:

count_total=462 , $\sum Fi=53.17$
 $FP = count_total[0.65 + 0.01 * \sum Fi]$
 $FP = 462 * (0.65 + 0.01 * 53.17)$
 $FP = 546$

Post Lab Assignment:

1. Explain Metrics for small organizations?
2. Explain Metrics for software quality?

Viva Questions:**Q. What are function points?**

A. Function points are the various features provided by the software product. It is considered as a unit of measurement for software size.

A **function point** is a "unit of measurement" to express the amount of business functionality an information system (as a product) provides to a user.

Function points measure software size. The cost (in dollars or hours) of a single unit is calculated from past projects.

Q. what is Functional point analysis:

It is a concept of finding out estimates in terms of points. It is one of the best practises of finding out estimates irrespective of any technology dependency.

FPA is divided into 4 attributes. All these 4 Attributes are nothing but functional key Ares of transaction.

For Example:

1. Input
2. Output
3. Input External Data
4. Output Internal Data

3. Can you explain steps in function points?

Below are the steps in function points:

First Count ILF, EIF, EI, EQ, RET, DET, FTR and use the rating tables. After you have counted all the elements you will get the unadjusted function points.

Put rating values 0 to 5 to all 14 GSC. Adding total of all 14 GSC to come out with total VAF.
Formula for VAF = $0.65 + (\text{sum of all GSC factor}/100)$.

Finally, make the calculation of adjusted function point. Formula: Total function point = VAF * Unadjusted function point.

Make estimation how many function points you will do per day. This is also called as "Performance factor". On basis of performance factor, you can calculate Man/Days.

Task4:

Aim: Analyze the Risk related to the project and prepare RMMM (Risk Mitigation, Monitoring, and Management) plan

Objective: To study types of risk and preparing RMMM plan.

References: Software Engineering Roger Pressman McGraw Hill Fifth edition Software Engineering Ian Somerville Pearson Education Sixth edition An Integrated Approach To Software Engineering Pankaj Jalote Narosa

Pre-requisite: Knowledge of software Analysis, Risk analysis.

Theory: Risk analysis and management are series steps that help a software team to understand uncertainty.

Types of Risks

1. Technical risks.
2. Business risks.
3. Project risks.

Risk Mitigation, Monitoring, and Management

- Risk mitigation (proactive planning for risk avoidance)
- Risk monitoring (assessing whether predicted risks occur or not, ensuring risk aversion steps are being properly applied, collect information for future risk analysis, attempt to determine which risks caused which problems)
- Risk management and contingency planning (actions to be taken in the event that mitigation steps have failed and the risk has become a live problem)

Sample Output:

Risk table

risk id	Category	Probability	Criticality	RMMM
1	TE	0.7	2	1
2	BU	0.4	3	2

Post Lab Assignment:

1. Explain Risk Identification?
2. Explain various risk strategies?

Viva Questions:**Q) What is RMMM?****A) Risk Mitigation, Monitoring, and Management**

- Risk mitigation (proactive planning for risk avoidance)
- Risk monitoring (assessing whether predicted risks occur or not, ensuring risk aversion steps are being properly applied, collect information for future risk analysis, attempt to determine which risks caused which problems)
- Risk management and contingency planning (actions to be taken in the event that mitigation steps have failed and the risk has become a live problem)

Q) What is Risk strategy?**A) Risk Strategies**

Reactive strategies - very common, also known as fire fighting, project team sets resources aside to deal with problems and does nothing until a risk becomes a problem

Proactive strategies - risk management begins long before technical work starts, risks are identified and prioritized by importance, then team builds a plan to avoid risks if they can or minimize them if the risks turn into problems

Q) What is Software Risks?**A) Project risks - threaten the project plan**

Technical risks - threaten product quality and the timeliness of the schedule

Business risks - threaten the viability of the software to be built (market risks, strategic risks, management risks, budget risks)

Known risks - predictable from careful evaluation of current project plan and those extrapolated from past project experience

Unknown risks - some problems simply occur without warning

Q) What is Risk Refinement?

A) Process of restating the risks as a set of more detailed risks that will be easier to mitigate, monitor, and manage.

CTC (condition-transition-consequence) format may be a good representation for the detailed risks (e.g. given that <condition> then there is a concern that (possibly) <consequence>).

Task5:

Aim: Develop Time-line chart and project table using PERT (Program Evaluation Review Technique) or CPM (Critical path method) project scheduling methods.

Objective: To study project scheduling and tracking

References: Software Engineering Roger Pressman McGraw Hill Fifth edition Software Engineering Ian Somerville Pearson Education Sixth edition An Integrated Approach To Software Engineering Pankaj Jalote Narosa

Prerequisite: Knowledge of Project and project scheduling, Task set for the software project, Adaptation criteria.

Theory: PERT and CPM are project scheduling methods that can be applied to software development. Both techniques are driven by information already developed in earlier project planning activities:

1. estimates of effort.
2. A decomposition of the product function.
3. The selection of the appropriate process model and task set .
4. Decomposition of tasks.

When creating a software project schedule, the planner begins with a set of tasks(the work breakdown structure).

Sample Output:

Time-line chart and work breakdown structure is prepared.

Post Lab Assignment:

1. Explain degree of Rigor?
2. Explain various Adaptation Criteria?

Viva Questions:

Q) Define PERT?

A) The Program Evaluation and Review Technique (PERT) is a network model that allows for randomness in activity completion times. PERT was developed in the late 1950's for the U.S. PERT (Program Evaluation and Review Technique) is one of the successful and proven methods among the many other techniques, such as, CPM, Function Point Counting, Top-Down Estimating, WAVE, etc.

Q) Define CPM?

A) The critical path itself represents the set or sequence of predecessor/successor activities which will take the longest time to complete. The duration of the critical path is the sum of the

activities' durations along the path. Thus, the critical path can be defined as the longest possible path through the "network" of project activities. The duration of the critical path represents the minimum time required to complete a project. Any delays along the critical path would imply that additional time would be required to complete the project.

Task6

Aim: Draw E-R (Entity Relationship) Diagrams, DFD (Dataflow Diagrams), CFD (Control Flow Diagrams) and structured charts for the project.

Introduction:

Developing databases is a very important task to develop a system. Before going to form exact database tables and establishing relationships between them, we conceptually or logically can model our database using ER diagrams.

In this experiment we will learn how to find the entities, its attributes and how the relationships between the entities can be established for a system.

Objectives:

After completing this experiment you will be able to:

- Identify entity sets, their attributes, and various relationships
- Represent the data model through ER diagram

Entity-Relationship model:

It is used to represent a logical design of a database to be created. In ER model, real world objects (or concepts) are abstracted as entities, and different possible associations among them are modelled as relationships.

For example, student and school -- they are two entities. Students study in school. So, these two entities are associated with a relationship "Studies in".

As another example, consider a system where some job runs every night, which updates the database. Here, job and database could be two entities. They are associated with the relationship "Updates"

Entity Set and Relationship Set

An entity set is a collection of all similar entities. For example, "Student" is an entity set that abstracts all students. Ram, John are specific entities belonging to this set. Similarly, a "Relationship" set is a set of similar relationships.

Attributes of Entity

Attributes are the characteristics describing any entity belonging to an entity set. Any entity in a set can be described by zero or more attributes.

For example, any student has got a name, age, an address. At any given time a student can study only at one school. In the school he would have a roll number, and of course a grade in which he studies. These data are the attributes of the entity set Student.

Keys

One or more attribute(s) of an entity set can be used to define the following keys:

Super key: One or more attributes, which when taken together, helps to uniquely identify an entity in an entity set. For example, a school can have any number of students. However, if we know grade and roll number, then we can uniquely identify a student in that school.

Candidate key: It is a minimal subset of a super key. In other words, a super key might contain extraneous attributes, which do not help in identifying an object uniquely. When such attributes are removed, the key formed so is called a candidate key.

Primary key: A database might have more than one candidate key. Any candidate key chosen for a particular implementation of the database is called a primary key.

Prime attribute: Any attribute taking part in a super key

Weak Entity

An entity set is said to be weak if it is dependent upon another entity set. A weak entity can't be uniquely identified only by its attributes. In other words, it doesn't have a super key.

For example, consider a company that allows employees to have travel allowance for their immediate family. So, here we have two entity sets: employee and family, related by "Can claim for". However, family doesn't have a super key. Existence of a family is entirely dependent on the concerned employee. So, it is meaningful only with reference to employee.

Entity Generalization and Specialization

Once we have identified the entity sets, we might find some similarities among them. For example, multiple person interacts with a banking system. Most of them are customers, and rest employees or other service providers. Here, customers, employees are persons, but with certain specializations. Or in other way, person is the generalized form of customer and employee entity sets.

ER model uses the "ISA" hierarchy to depict specialization (and thus, generalization).

Mapping Cardinalities

One of the main tasks of ER modeling is to associate different entity sets. Let's consider two entity sets E1 and E2 associated by a relationship set R. Based on the number of entities in E1 and E2 are associated with, we can have the following four type of mappings:

One to one: An entity in E1 is related to at most a single entity in E2, and vice versa

One to many: An entity in E1 could be related to zero or more entities in E2. Any entity in E2 could be related to at most a single entity in E1.

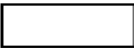

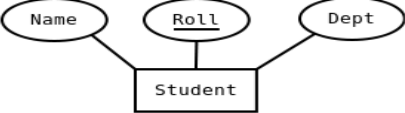
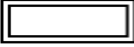
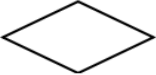

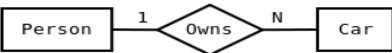

Many to one: Zero or more number of entities in E1 could be associated to a single entity in E2. However, an entity in E2 could be related to at most one entity in E1.

Many to many: Any number of entities could be related to any number of entities in E2, including zero, and vice versa.

ER Diagram

From a given problem statement we identify the possible entity sets, their attributes, and relationships among different entity sets. Once we have these information, we represent them pictorially, called an entity-relationship (ER) diagram.

Graphical Notations for ER Diagram:

Term	Notation	Remarks
Entity set		Name of the set is written inside the rectangle
Attribute		Name of the attribute is written inside the ellipse
Entity with attributes		Roll is the primary key; denoted with an underline
Weak entity set		
Relationship set		Name of the relationship is written inside the diamond
Related entity sets		
Relationship cardinality		A person can own zero or more cars but no two persons can own the same car
Relationship with weak entity set		

Importance of ER modelling

Figure - 01 shows the different steps involved in implementation of a (relational) database.



Figure - 01: Steps to implement a RDBMS

Given a problem statement, the first step is to identify the entities, attributes and relationships. We represent them using an ER diagram. Using this ER diagram, table structures are created, along with required constraints. Finally, these tables are normalized in order to remove redundancy and maintain data integrity. Thus, to have data stored efficiently, the ER diagram is to be drawn as much detailed and accurate as possible.

Simulation:

Draw an ER diagram for a School Management System based on the following information.

You have been asked to implement a database for a school management system (SMS). This primarily consists of maintaining students' information like name, address, date of birth, roll number, department, and so on. Details about the school to be stored includes school's name, location. Although it is unlikely that there would be two schools at the same place with same name, but our SMS would like to accommodate this possibility.

Faculty members work in the school. They teach the students. A faculty member normally teaches multiple students at a time. Also, he can teach multiple courses to the students.

ER Diagram for School Management System

We will follow the steps shown below to draw the ER diagram

1. Identify entity sets and their attributes
2. Draw the entities
3. Identify relationships
4. Draw the relationships
5. Identify mapping cardinalities and represent them

Case Study:

A Library Information System for SE VLabs Institute:

The SE VLabs Institute has been recently setup to provide state-of-the-art research facilities in the field of Software Engineering. Apart from research scholars (students) and professors, it also includes quite a large number of employees who work on different projects undertaken by the institution.

As the size and capacity of the institute is increasing with the time, it has been proposed to develop a Library Information System (LIS) for the benefit of students and employees of the institute. LIS will enable the members to borrow a book (or return it) with ease while sitting at his desk/chamber. The system also enables a member to extend the date of his borrowing if no other booking for that particular book has been made. For the library staff, this system aids them to easily handle day-to-day book transactions. The librarian, who has administrative privileges and complete control over the system, can enter a new record into the system when a new book

has been purchased, or remove a record in case any book is taken off the shelf. Any non-member is free to use this system to browse/search books online. However, issuing or returning books is restricted to valid users (members) of LIS only.

The final deliverable would be a web application (using the recent HTML 5), which should run only within the institute LAN. Although this reduces security risk of the software to a large extent, care should be taken no confidential information (eg., passwords) is stored in plain text.

A robust database backend is essential for a high-quality information system. Database schema should be efficiently modeled, refined, and normalized. In this section we would develop a simple ER model for the Library Information System.

The first step towards ER modeling is to identify the set of relevant entities from the given problem statement. The two primary, and obvious, entity sets in this context are "Member" and "Book". The entity set "Member" represents all students, professors, or employees who have registered themselves with the LIS. While registering with the LIS one has to furnish a lot of personal and professional information. This typically includes name (well, that is trivial), employee ID (roll # for students), email address, phone #, age, date of joining in this institute. The system may store some not-so-important information as well like, blood group, marital status, and so on. All these pieces of information that an user has to provide are sufficient to describe a particular member. These characteristics are the attributes of the entities belonging to the entity set "Member".

It is essential for an entity to have one or more attributes that help us to distinguish it from another entity. 'Name' can't help that -- two persons could have exactly the same name. However, ('Name', 'Phone #') combination seems to be okay. No two persons can have the same phone number. 'Employee ID', 'Email address' are other potential candidates. Here, 'Employee ID', 'Email address' and ('Name', 'Phone #') are super keys. We choose 'Employee ID' to uniquely identify an user in our implementation. So, 'Employee ID' becomes our primary key (PK) for the "Member" entity set. Figure 1 represents this set along with its attributes and the primary key.

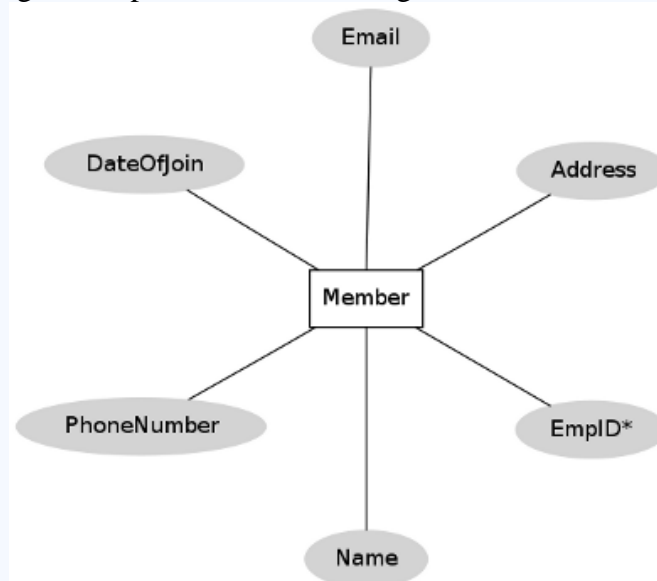


Figure 1: "Member" entity set

Let us now focus on the "Book" entity set. Typical attributes of a book are its title, name of author(s), publisher, date of publication, edition, language, ISBN-10, ISBN-13, price (of course!), date of purchase. The set of listed attributes for a book doesn't give a straight forward

choice of primary key. For instance, several books could have the same title. Again, ISBN numbers for a book are specific to its edition -- it can't distinguish between two books of the same edition. One might be tempted to use a combination of ('Title', 'Authors') as a primary key. This has some shortcomings. It is advisable not to use texts as a PK. Moreover, the number of authors that a book could have is not fixed, although it is a small, finite number. The rules of normalization (not covered here) would dictate to have a separate field for each author like 'Author1', 'Author2', and so on. Therefore, we assign an extra attribute, 'ID', to each book as its PK. Different databases available in the market provide mechanisms to generate such a unique ID, and automatically increment it whenever a new new entity is added. In fact, we could assign such an ID to the "Member" entity set as well. However, because of availability of the unique 'Employee ID' field, we skipped that. A graphical representation of the "Book" entity set is shown in figure 2.

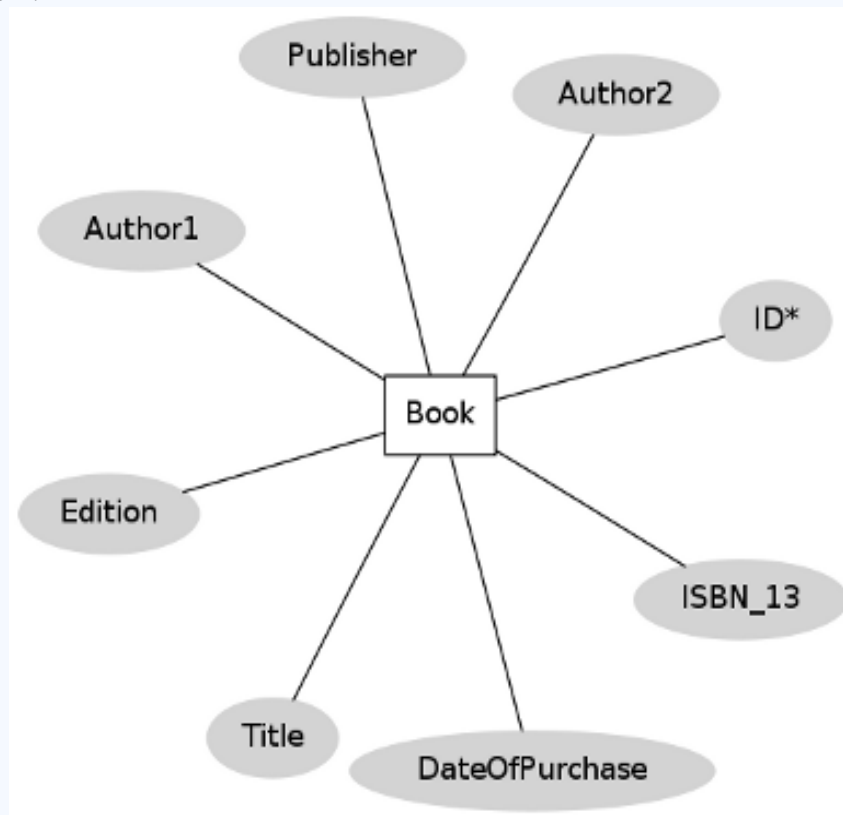


Figure 2: "Book" entity set

One point to note here is that a book is likely to have multiple copies in the library. Therefore, one might wish to have a '# of copies' attribute for the "Book" entity set. However, that won't allow us to differentiate among the different copies of book bearing same title by same author(s), edition, and publisher. The approach that we have taken is to uniquely identify each book even though they are copies of the same title.

To buy any new book an order is to be placed to the distributor. This task is done by the librarian. Therefore, "Librarian" and "Distributor" are two other entities playing roles in this system.

Having identified the key entities, we could now relate them with each other. Let us consider the entity sets "Member" and "Book". A member can issue books. In fact, he can issue multiple books up to a finite number say, N. A particular book, however, could be issued by a single

member only. Therefore, we have a one-to-many mapping from "Member" to "Book" entity sets. This relationship between "Member" and "Book" entity sets is pictorially depicted in figure 3.

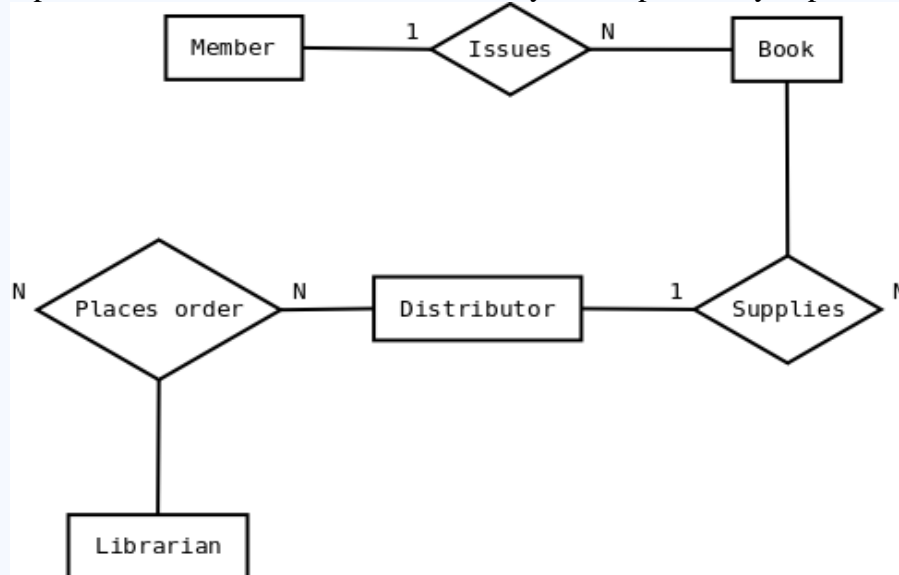


Figure 3: Relationships among different entity sets

Figure 3 also shows that the librarian can "place order" for books to the distributor. This is a many-to-many mapping since a librarian can purchase books from multiple distributors. Also, if the institute has more than one librarians (or any other staff having such authority), then each of them could place order to the same distributor. An order is termed as complete when distributor supplies the book(s) and invoice.

The design in figure 3 has a flaw. Librarian himself could be a member of the LIS. However, he is a "special" kind of member since he can place order for books. Our ER diagram doesn't reflect this scenario. Such special roles of an entity set could be represented using "ISA" relationship, which is not discussed here.

Any kind of designing couldn't be possibly done at one go. Therefore, the baseline ER model so prepared should be revised by considering the business model yet again to ensure that all necessary information could be captured. Once this has been finalized, the next logical step would be to create table structures for each identified entity set (and relationships in some cases) and normalize the relations

Webliography

Entity-relationship modelling

Entity Relationship Modelling – 2

Viva Questions:**Q) Define E-R Diagram?**

A) An entity relationship diagram (ERD) shows the relationships of entity sets stored in a database. An entity in this context is a component of data. In other words, ER diagrams illustrate the logical structure of databases.

Q) What is DFD?

A) A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modelling its process aspects. ADFD is often used as a preliminary step to create an overview of the system, which can later be elaborated.

Q) Define CFD?

A) A control flow diagram (CFD) is a diagram to describe the control flow of a business process, process or review

Control flow diagrams were developed in the 1950s, and are widely used in multiple engineering disciplines. They are one of the classic business process modelling methodologies, along with flow charts, data flow diagrams, functional flow block diagram, Gantt charts, PERT diagrams, and IDEF

Q) Define structured Charts?

A) A Structure Chart (SC) in software engineering and organizational theory, is a chart which shows the breakdown of a system to its lowest manageable levels. They are used in structured programming to arrange program modules into a tree. Each module is represented by a box, which contains the module's name.

Task7:

Aim: Design of Test cases based on requirements and design

Design of the test cases.

Objective: Development of new software, like any other product, remains incomplete until it subjected to exhaustive tests. The primary objective of testing is not only to verify that all desired features have been implemented correctly, but also includes the verification of the software behaviour in case of "bad inputs".

In this experiment we discuss in brief about different types of testing, and provide tools and mechanisms to have hands-on experience on unit testing.

Software Testing:

Testing software is an important part of the development life cycle of software. It is an expensive activity. Hence, appropriate testing methods are necessary for ensuring the reliability of a program. According to the ANSI/IEEE 1059 standard, the definition of testing is the process of analyzing a software item, to detect the differences between existing and required conditions i.e. defects/errors/bugs and to evaluate the features of the software item.

The purpose of testing is to verify and validate software and to find the defects present in software. The purpose of finding those problems is to get them fixed.

Verification is the checking or we can say the testing of software for consistency and conformance by evaluating the results against pre-specified requirements.

Validation looks at the systems correctness, i.e. the process of checking that what has been specified is what the user actually wanted.

Defect is a variance between the expected and actual result. The defect's ultimate source may be traced to a fault introduced in the specification, design, or development (coding) phases.

Test Cases and Test Suite

A test case describes input descriptions and an expected output descriptions. Input are of two types: preconditions (circumstances that hold prior to test case execution) and the actual inputs that are identified by some testing methods. The set of test cases is called a test suite. We may have a test suite of all possible test cases.

Types of Software Testing

Testing is done in every stage of software development life cycle, but the testing done at each level of software development is different in nature and has different objectives. There are different types of testing, such as stress testing, volume testing, configuration testing, compatibility testing, recovery testing, maintenance testing, documentation testing, and usability testing. Software testing are mainly of following types.

Unit Testing

Integration Testing

System Testing

Unit Testing

Unit testing is done at the lowest level. It tests the basic unit of software, that is the smallest testable piece of software. The individual component or unit of a program are tested in unit testing. Unit testing are of two types.

Black box testing: This is also known as functional testing, where the test cases are designed based on input output values only. There are many types of Black Box Testing but following are the prominent ones.

- Equivalence class partitioning: In this approach, the domain of input values to a program is divided into a set of equivalence classes. e.g. Consider a software program that computes whether an integer number is even or not that is in the range of 0 to 10. Determine the equivalence class test suite. There are three equivalence classes for this program. - The set of negative integer - The integers in the range 0 to 10 - The integer larger than 10
- Boundary value analysis : In this approach, while designing the test cases, the values at boundaries of different equivalence classes are taken into consideration. e.g. In the above given example as in equivalence class partitioning, a boundary values based test suite is { 0, -1, 10, 11 }

White box testing: It is also known as structural testing. In this testing, test cases are designed on the basis of examination of the code. This testing is performed based on the knowledge of how the system is implemented. It includes analyzing data flow, control flow, information flow, coding practices, exception and error handling within the system, to test the intended and unintended software behavior. White box testing can be performed to validate whether code implementation follows intended design, to validate implemented security functionality, and to uncover exploitable vulnerabilities. This testing requires access to the source code. Though white box testing can be performed any time in the life cycle after the code is developed, but it is a good practice to perform white box testing during the unit testing phase.

Integration Testing

Integration testing is performed when two or more tested units are combined into a larger structure. The main objective of this testing is to check whether the different modules of a program interface with each other properly or not. This testing is mainly of two types:

- Top-down approach
- Bottom-up approach

In bottom-up approach, each subsystem is tested separately and then the full system is tested. But the top-down integration testing starts with the main routine and one or two subordinate routines in the system. After the top-level 'skeleton' has been tested, the immediately subroutines of the 'skeleton' are combined with it and tested.

System Testing

System testing tends to affirm the end-to-end quality of the entire system. System testing is often based on the functional / requirement specification of the system. Non-functional quality attributes, such as reliability, security, and maintainability are also checked. There are three types of system testing

Alpha testing is done by the developers who develop the software. This testing is also done by the client or an outsider with the presence of developer or we can say tester.

Beta testing is done by very little number of end users before the delivery, where the change requests are fixed, if the user gives any feedback or reports any type of defect.

User Acceptance testing is also another level of the system testing process where the system is tested for acceptability. This test evaluates the system's compliance with the client requirements and assess whether it is acceptable for software delivery

An error correction may introduce new errors. Therefore, after every round of error-fixing, another testing is carried out, i.e. called regression testing. Regression testing does not belong to unit testing, integration testing, or system testing, instead, it is a separate dimension to these three forms of testing.

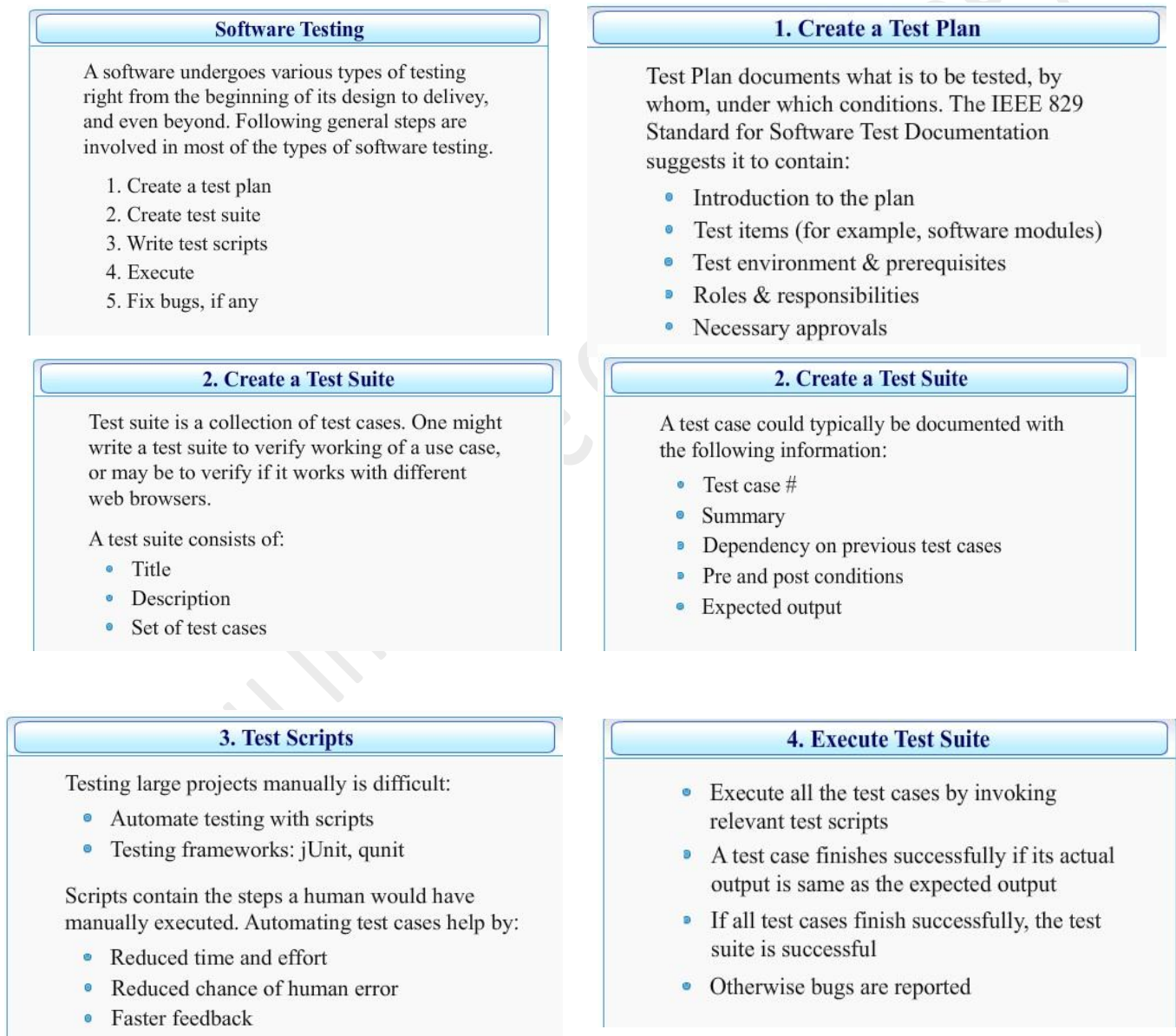
Regression Testing

The purpose of regression testing is to ensure that bug fixes and new functionality introduced in software do not adversely affect the unmodified parts of the program [2]. Regression testing is an important activity at both testing and maintenance phases. When a piece of software is modified, it is necessary to ensure that the quality of the software is preserved. To this end, regression testing is to retest the software using the test cases selected from the original test suite

Simulation:

Exhaustive testing of software is required to determine it is working as per expectations and requirements. Developers often do not have enough time (or at times interest) to test their codes thoroughly. To handle such scenarios, most projects usually have a dedicated testing team. However, unit testing, at least, is done by the developers.

Irrespective of who performs testing, or what is being tested, testing usually involves some generic steps. In this simulation we provide a broad overview of the testing process.

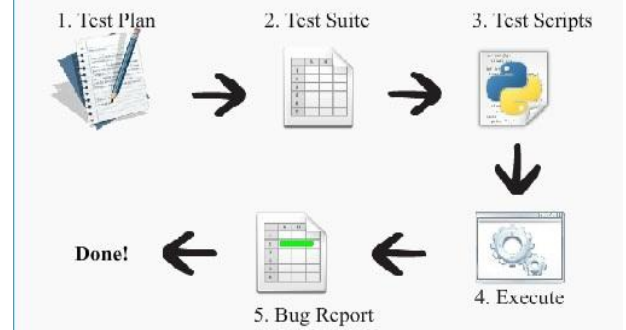


5. Bug fixing

- Developers act on the reported bugs, and fix them
- Repeat step #4 again
- If the reported bug is not encountered anymore, it is closed

Steps #4 and #5 are repeated until there is no open bug

A Graphical Representation



Steps for conducting the experiment

General Instructions:

Follow are the steps to be followed in general to perform the experiments in Software Engineering Virtual Lab

- 1 Read the theory about the experiment
- 2 View the simulation provided for a chosen, related problem
- 3 Take the self evaluation to judge your understanding (optional, but recommended)
- 4 Solve the given list of exercises

Following are the instructions specifically for this experiment:

- 1 Type in the code (in JavaScript) to be tested in the text area below the header "Code"
- 2 Once the code is ready, click on the "Create test suite" just below the code area. A small dialog box will appear just below the button.
- 3 Add a title and summary for the test suite to be created. (Both are optional.) When done, click on the "Add" button. If this test suite is not supposed to be added, click on the "Cancel" link.
- 4 After clicking on the "Add" button from the previous step, a dialog will display the new test suite. Every test suite is identified with a unique ID: an (auto-incrementing) integer prefixed with "TS". The first test suite will have the ID "TS0 ", and so on.
- 5 An already added test suite could be removed by clicking on the "Remove" link
- 6 Once a test suite has been created, click on the "Add test cases" button to add the test cases individually

7 After clicking on the "Add test cases" button a spreadsheet-type dialog will appear just below the button. The spreadsheet has six columns:

Summary: A brief description of the test case (mandatory)

Script: A (JavaScript) function to be called for execution (mandatory)

Expected Output: The value the above function call is expected to return (mandatory)

Actual Output: The value actually obtained after making the function call. This column would be populated automatically after the test suite is executed.

Manual Testing: Certain test cases could not be checked automatically. For example, a testing framework may not verify that an HTML element X is not overlapping with another HTML element Y. In such cases, manual intervention is required. To specify that a test case would be executed manually, select the "Yes" check box under this column.

Status: Indicates the status of a test case after it is executed. Possible values are:

No Run: The test suite has not been executed yet or the test case has been set for Manual Testing)

Pass: The concerned test cases expected and actual values are same

Fail: The concerned test cases expected and actual values are NOT same

8 The number of columns in the spreadsheet are fixed (six). The number of initial rows are 10. New rows could, however, be added by pressing Enter while the last column of the last row is being selected.

9 **NOTE:** Summary of a test case IS mandatory for every test case in the test suite. If a particular test case has no summary, the concerned test case, and all other subsequent test cases, would be ignored!

10 Once test cases have been written, click on the "Execute test suite" button to execute the test cases for the concerned test suite. Please note that this may not execute all the test cases in the test suite if the constraints as mentioned in the previous step are not met. Also, any test case set for manual testing would be skipped.

Viva Questions:

Q) Define Test case?

A) A test case is a document, which has a set of test data, preconditions, expected results and post conditions, developed for a particular test scenario in order to verify compliance against a specific requirement.

Test Case acts as the starting point for the test execution, and after applying a set of input values; the application has a definitive outcome and leaves the system at some end point or also known as execution post condition.

Q) Explain Testing types?

A) Black box testing – Internal system design is not considered in this type of testing. Tests are based on requirements and functionality.

White box testing – This testing is based on knowledge of the internal logic of an application's code. Also known as Glass box Testing. Internal software and code working should be known for this type of testing. Tests are based on coverage of code statements, branches, paths, conditions.

Unit testing – Testing of individual software components or modules. Typically done by the programmer and not by testers, as it requires detailed knowledge of the internal program design and code. may require developing test driver modules or test harnesses.

Incremental integration testing – Bottom up approach for testing i.e continuous testing of an application as new functionality is added; Application functionality and modules should be independent enough to test separately. done by programmers or by testers.

Integration testing – Testing of integrated modules to verify combined functionality after integration. Modules are typically code modules, individual applications, client and server applications on a network, etc. This type of testing is especially relevant to client/server and distributed systems.

Functional testing – This type of testing ignores the internal parts and focus on the output is as per requirement or not. Black-box type testing geared to functional requirements of an application.

System testing – Entire system is tested as per the requirements. Black-box type testing that is based on overall requirements specifications, covers all combined parts of a system.

Regression testing – Testing the application as a whole for the modification in any module or functionality. Difficult to cover all the system in regression testing so typically automation tools are used for these testing types.

Acceptance testing -Normally this type of testing is done to verify if system meets the customer specified requirements. User or customer does this testing to determine whether to accept application.

Performance testing – Term often used interchangeably with ‘stress’ and ‘load’ testing. To check whether system meets performance requirements. Used different performance and load tools to do this.

Alpha testing – In house virtual user environment can be created for this type of testing. Testing is done at the end of development. Still minor design changes may be made as a result of such testing.

Beta testing – Testing typically done by end-users or others. Final testing before releasing application for commercial purpose.

Q) Define Testing?

A) Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not.

Testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

Q) Who Does Testing?

A) It depends on the process and the associated stakeholders of the project(s). In the IT industry, large companies have a team with responsibilities to evaluate the developed software in context of the given requirements. Moreover, developers also conduct testing which is called Unit Testing. In most cases, the following professionals are involved in testing a system within their respective capacities:

Software Tester

Software Developer

Project Lead/Manager

End User

Task8:

Aim: Prepare FTR (Formal technical reviews) and Prepare version control and change control for software configuration items.

Objective: To understand Software configuration management

References: Software Engineering Roger Pressman McGraw Hill Fifth edition Software Engineering Ian Somerville Pearson Education Sixth edition An Integrated Approach to Software Engineering Pankaj Jalote Narosa.

Pre-requisite: Knowledge of **Software configuration management**

Theory: FTR-formal technical review serves as a training ground, enabling junior engineer to observe different approaches to software analysis. Version control-It combines procedures and tools to manage different versions of configuration objects that are created during software process Change control-It is a formal process used to ensure a product services or process is only modifies in line when identified necessary change.

Sample Output:

A document which contain software configuration items, details of change control and version control with respect to software configuration management.

Post Lab Assignment:

1. Explain software configuration management with baselines SCIs?
2. Explain Identification of objects in the software configuration?

Viva Questions:**Q) Define FTR?**

A) Formal Technical Reviews (FTR) are formal examinations of software products to identify faults (i.e. departures. from specifications and standards). They are a class of reviews that include: ϕ System Definition Reviews.

Formal Technical Review: Reviews that include walkthroughs, inspection,

Formal Technical Review

Reviews that include walkthroughs, inspection, round-robin reviews and other small group technical assessment of software. It is a planned and control meeting attended by the analyst, programmers and people involve in the software development.

Uncover errors in logic, function or implementation for any representation of software

To verify that the software under review meets the requirements

To ensure that the software has been represented according to predefined standards

To achieve software that is developed in a uniform manner.

To make project more manageable.

Early discovery of software defects, so that in the development and maintenance phase the errors are substantially reduced. " Serves as a training ground, enabling junior members to observe the different approaches in the software development phases (gives them helicopter view of what other are doing when developing the software).

Allows for continuity and backup of the project. This is because a number of people are become familiar with parts of the software that they might not have otherwise seen,

Greater cohesion between different developers.

Q) What are Objectives of FTR?

A) A formal technical review is a software quality assurance activity performed by software Engineers (and others). The objectives of the FTR are

- (1) To uncover errors in function, logic, or implementation for any representation of the software;
- (2) To verify that the software under review meets its requirements;
- (3) To ensure that the software has been represented according to predefined standards;
- (4) To achieve software that is developed in a uniform manner;
- (5) To make projects more manageable.

Q) Discuss FTR?

A) FTR is effective quality improvement

Reviews can find 60-100% of all defects.

Reviews are technical, not management.

Review data can assess/improve quality of:

- work product
- software development process

- review process

Reviews reduce total project cost, but have non-trivial cost (~15%)

Upstream defect removal is 10-100 times cheaper.

Reviews disseminate domain knowledge, development skills, and corporate culture.

REFERENCES:

- 1 - Software Engineering ? A Practitioner" s Approach, Roger S. Pressman, 1996, MGH.
- 2 - Software Engineering by Ian sommerville, Pearson Edu, 5th edition, 1999
- 3 - An Integrated Approach to software engineering by Pankaj jalote , 1991 Narosa
- 4 -Requirements Engineering: A Good Practice Guide, Ian Sommerville, Pete Sawyer, Wiley India Pvt Ltd, 2009
- 5 - Fundamentals of Software Engineering, Rajib Mall, Prentice-Hall of India, 3rd Edition, 2
- 6 - Database System Concepts, Henry F. Korth , Abraham Silberschatz, McGraw Hills, 5th Edition
7. - Fundamentals of Software Engineering, Rajib Mall, Prentice-Hall of India, 3rd Edition, 2009
- 8 - Software Engineering: A Practioner's Approach, Roger S. Pressman, McGraw Hills, 7th Edition, 2009