ਜਗਤ ਗੁਰੂ ਨਾਨਕ ਦੇਵ
ਪੰਜਾਬ ਸਟੇਟ ਓਪਨ ਯੂਨੀਵਰਸਿਟੀ
ਪਟਿਆਲਾ

JAGAT GURU NANAK DEV
PUNJAB STATE OPEN UNIVERSITY, PATIALA

(Established by Act No. 19 of 2019 of the Legislature of State of Punjab)

The Motto of the University

(SEWA)

SKILL ENHANCEMENT EMPLOYABILITY WISDOM ACCESSIBILITY

ਜਗਤ ਗੁਰੂ ਨਾਨਕ ਦੇਵ
ਪੰਜਾਬ ਸਟੇਟ ਓਪਨ ਯੂਨੀਵਰਸਿਟੀ, ਪਟਿਆਲਾ
JAGAT GURU NANAK DEV
PUNJAB STATE OPEN UNIVERSITY, PATIALA

Bachelor Of Computer Applications (BCA)
Course Name: SOFTWARE ENGINEERING
Course Code: BCA-4-01T

ADDRESS: C/28, THE LOWER MALL, PATIALA-147001
WEBSITE: www.psou.ac.in

JAGAT GURU NANAK DEV

PUNJAB STATE OPEN UNIVERSITY PATIALA

(Established by Act No.19 of 2019 of Legislature of the State of Punjab)

PROGRAMME & COURSE COORDINATOR :
Dr. Monika Pathak
Assistant Professor, School of Sciences and Emerging Technologies
Jagat Guru Nanak Dev Punjab State Open University, Patiala

PROGRAMME CO-COORDINATOR :
Dr. Gaurav Dhiman
Assistant Professor, School of Sciences and Emerging Technologies
Jagat Guru Nanak Dev Punjab State Open University, Patiala

JAGAT GURU NANAK DEV

PUNJAB STATE OPEN UNIVERSITY PATIALA

(Established by Act No.19 of 2019 of Legislature of the State of Punjab)

PREFACE

Jagat Guru Nanak Dev Punjab State Open University, Patiala was established in Decembas 2019 by Act 19 of the Legislature of State of Punjab. It is the first and only Open Universit of the State, entrusted with the responsibility of making higher education accessible to all especially to those sections of society who do not have the means, time or opportunity to pursue regular education.

In keeping with the nature of an Open University, this University provides a flexible education system to suit every need. The time given to complete a programme is double the duration of a regular mode programme. Well-designed study material has been prepared in consultation with experts in their respective fields.

The University offers programmes which have been designed to provide relevant, skill-based and employability-enhancing education. The study material provided in this booklet is self instructional, with self-assessment exercises, and recommendations for further readings. The syllabus has been divided in sections, and provided as units for simplification.

The Learner Support Centres/Study Centres are located in the Government and Government aided colleges of Punjab, to enable students to make use of reading facilities, and for curriculum-based counselling and practicals. We, at the University, welcome you to be a part of this institution of knowledge.

Prof. G. S. Batra,

Dean Academic Affairs

BCA-4-01T: Software Engineering

Total Marks: 100
External Marks: 70
Internal Marks: 30
Credits: 6
Pass Percentage:40%

INSTRUCTIONS FOR THE PAPER SETTER/EXAMINER

1. The syllabus prescribed should be strictly adhered to.
2. The question paper will consist of three sections: A, B, and C. Sections A and B will have four questions from the respective sections of the syllabus and will carry 10 marks each. The candidates will attempt two questions from each section.
3. Section C will have fifteen short answer questions covering the entire syllabus. Each question will carry 3 marks. Candidates will attempt any ten questions from this section.
4. The examiner shall give a clear instruction to the candidates to attempt questions only at one place and only once. Second or subsequent attempts, unless the earlier ones have been crossed out, shall not be evaluated.
5. The duration of each paper will be three hours.

INSTRUCTIONS FOR THE CANDIDATES

Candidates are required to attempt any two questions each from the sections A and B of the question paper and any ten short q questions from Section C. They have to attempt questions only at one place and only once. Second or subsequent attempts, unless the earlier ones have been crossed out, shall not be evaluated.

Course: Software Engineering	
Course Code: BCA-4-01T	
Course Outcomes (COs) After the completion of this course, the students will be able to:	
CO1	Understand the software development life cycle which increases the growth opportunity.
CO2	Learn the detail knowledge of software requirement analysis.
CO3	Understands the detailed knowledge of software design and coding.
CO4	Understand the software testing that is relevant to the industry.
CO5	Acquire the detail knowledge of the fundamentals, including terminology; the nature and need for maintenance; maintenance costs and software evolution

SECTION-A

Unit I: Introduction of Software Engineering: The Problem Domain, Software Engineering, Challenges, Software Engineering Approach. Software development life cycle and its phases, Software development process models: Waterfall, Prototyping, Iterative.

Unit II: Software Process: Characteristics of software process, Project management process, Software configuration management process.

Unit III: Project Planning: Activities, COCOMO model. Software Metrics – Definition, Importance, Categories of metrics. Software Quality – Attributes, Cyclomatic complexity metric.

Unit IV: Software Requirements Analysis: Need for SRS, Data flow diagrams, Data Dictionary, entity relationship diagram, Characteristics and components of SRS, validation, metrics.

SECTION-B

Unit V: Software Design: Design principles, Module-level concepts, Structure Chart and Structured Design methodology, verification, metrics: network metrics, information flow metrics.

Unit VI: Coding: Programming Principles and Guidelines, Verification- code inspections, static analysis.

Unit VII: Software Testing: Testing fundamentals, Black Box Testing: Equivalence class partitioning, Boundary value analysis, cause-effect graphing; White Box Testing: Control flow and Data flow based testing, mutation testing; levels of testing, test plan, test case specification, test case execution and analysis.

Unit VIII: Software Maintenance: Categories of maintenance. Software Reliability – Definition, uses of reliability studies

Reference Books:

- Pankaj Jalote, "An Integrated approach to Software Engineering", 3rd Edition 2005, Narosa Publications.
- K.K. Aggarwal, Yogesh Singh, "Software Engineering", Revised 2nd Edition, New Age International Publishers.
- Roger. S. Pressman, "Software Engineering – A Practitioner's Approach", 5th Edition, Tata McGraw Hill.

BACHELOR OF COMPUTER APPLICATIONS (BCA)
BCA-4-01T: SOFTWARE ENGINEERING

UNIT 1: INTRODUCTION OF SOFTWARE ENGINEERING

1.1 **INTRODUCTION**

Today, computers are everywhere .Almost all organizations are dependent on the computer. It helps in marketing, accounting, sales etc. Computer is a combination of hardware and software. Computer cannot work without software. Software helps hardware to perform various tasks. Software has changes the society and their work. There are many different software packages in the market. People choose software according to their need and facility. It helps people in many different ways, including saving time and money. For example, CRM (customer relationship management) software helps business to manage their customer relationships.

1.2 **PROGRAM**

1. A program is normally complete in itself and small in size.
2. It is used by the single user. Program lacks in proper documentation because single user is the author or single developer of the program. User makes little documentation according to his/her needs without using any format of documentation.
3. In a program, author is the only user of the program. If any problem occurs in the program and program crashes, then the author can fix the problem and can start using program again without any disturbance.
4. Program is the result of adhoc development where issues like portability and reliability are not considered.

1.3 SOFTWARE

Computers cannot perform any task/work without getting instructions from user through software. The combination of hardware and software is necessary if we want to get some task to be performed by the computer:

1. Software gives instructions to the computer what to do and how to do it.
2. Software is used for managing controlling and integrating the hardware components of the computer.
3. Software is not only the collection of computer programs. There is a difference between a program and software.
4. IEEE defines:***"Software is a collection of computer programs, procedures, rules and associated documentation and data."***

- ***Program:*** It is a list of organized instructions when executed behave in a predetermined manner.
- ***Procedures*****:** It is a step by step instruction to achieve a desired result. It explains how to prepare something.
- ***Rules:*** Rules are basically standards for actions/activities. Rule is basically a collection of guidelines to achieve something.
- ***Documentation:*** It is a set of documents provided in a particular format. It is written on paper, online or digital media.
- ***Data:*** Data is a raw material which is used in software. It is a set of values.

5. Software is basically developed by a team of developers who ensure the proper documentation.
6. Software developers ensure the well-designed interface because software are used by many people from different backgrounds.
7. Software is usually large in size and used by many people according to their needs.
8. Many resources and very many efforts are required for developing software.
9. Cost of developing software is very high because the software technology is still labor intensive and expensive.
10. Software projects are very large, involving many people and span over many years.
11. Software development is a systematic process where key issues like portability and reliability are considered.

1.4 DIFFERENCE BETWEEN PROGRAM AND SOFTWARE

Sr. No.	Concept	Program	Software
1.	Size	Size of program is small.	Size of software is large.
2.	Number of users	Single user	Large number of users.
3.	User interface	Program lacks in adequate user interface.	Software provide well designed and adequate user interface.
4.	Development	Program is the result of adhoc development.	Software is the result of systematic development.
5.	Number of developers	Single developer.	Team of developers.
6.	Documentation	User is the developer who makes little documentation according to his/her needs without using any format of documentation.	Software is the result of systematic work of team of developers. They provide proper documentation.
7.	Concentration on Issues	Issues like portability and reliability are not considered/concentrated.	Issues like portability and reliability are given proper concentration.
8.	Effort	Less effort is required.	More effort is required in the development of software as compared to program.
9.	Use of Resources	Less resources are required.	More resources are required as compared to program.
10.	Cost	Development cost of programme is less.	Development cost of software is approximately ten times more as compared to program.
11.	Complexity	Less complex.	More complex as compared to program.

Table 1.1 Difference Between Program and Software

1.5TYPES OF SOFTWARE

Software is basically of three types:

1. System software
2. Application software
3. Service or utility software

1. ***System Software:*** System software is a set of one more programs designed to control the operation and extend the processing capabilities of a computer system.

The efficient system software allows application software to be run on the computer with less time and effort. System software makes the operation of a computer system more effective and efficient.

System software helps the hardware components to work together and provides support for the development and execution of application software.

Some commonly known system software are as follows:

(i) **Operating System:** An operating system is software that runs on computer and manages the computer hardware. Operating system performs basic tasks such as recognizing input from keyboard, sending output to the display screen, keeping tracks of files and controlling peripheral devices like printers and responsible for security ensuring that unauthorized users do not access the system. Operating system acts as an interface between hardware and software.

(ii) **Language Processors (Translators):** Language processors or translators are system software which transfer (convert) the instructions written in a programming language, into a form which can be interpreted and executed by a computer system.

It is of three types:

- **Compilers:** A compiler is a software that accept the total program code (high level language) as input and then converts it into machine code (low level language). Compilers also detect and indicate certain types of errors in source programs automatically.

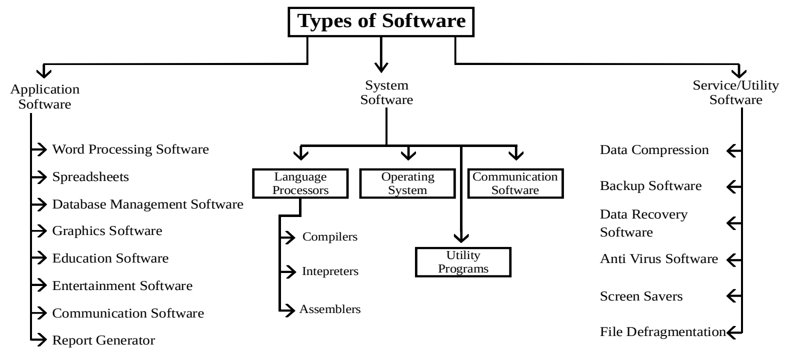


Fig. 1.1 Types of Software

- **Interpreters:** The interpreter takes one source program instruction, translates it into object code and executes it, then takes up the next instruction translates it and so on. Interpreters are slower than compilers.
- **Assemblers:** An Assemblers is a system software that takes as an input a program written in a assembly language and as an output generates the program written in machine level which can be directly executed on the computer.

(iii) **Communication Software:** Communication software enables to transfer the data and programs from one computer system to another computer system in a network environment where multiple computers are inter-connected together by communication networks.

(iv) **Utility Programs:** Utility programs is a set of programs which helps users in system maintenance and in performing tasks of routine nature like formatting of hard disks, taking backup of files, sorting records etc.

2. **Application Software:** Application software is a set of one or more programs, designed to solve a specific problem. There are wide ranges of applications ranging from simple applications like word processing, banking etc. to complex scientific/engineering applications like weather forecasting, design of aircrafts etc.

Some commonly known application software are as follows:

(i) **Word processing software:** Word processing software helps in day to day documentation work. It helps in creating texts, manipulating, formatting and printing of the text. It is considered as a computerized version of typewriter. It allows users to make changes and corrections in the text easily.

(ii) **Spreadsheets (Electronic Spreads Sheets):** Electronic spreadsheets are sheets of paper with rows and columns. Data is entered in a tabular form. The area where row and column meet is known as cell. User put text, number or formulae into cell. It also provides flexibility to user to make changes in spreadsheets. It provides various built in functions for the convenience of user.

(iii) **Database Management Software:** Database management system is a software package that allows a user to perform various operations like creating, deleting, updating, adding, modifying data in databases. Whereas database is an organized collection of logically related data. It makes retrieval of information easy and effective.

(iv) **Graphics Software:** Graphics software is a collection of programs which manipulate the images to any extent. These programs are referred to as point or draw programs in which user can draw illustrations from the scratch using an electronic pointing devices such as mouse, light pen or brush. Graphics programs are also known as point programs.

(v) **Education Software:** Education softwares are known as teaching and learning tool. These softwares are used to teach students. These softwares are very effective in mathematics, recognizing of alphabets, read whole words and sentences etc.

(vi) **Entertainment Software:** Entertainment software allows a computer system to be used as an entertainment tool. A good example of such an application is computer video games.

(vii) **Communication Software:** Communication software is used to share information and resources by connecting the computers. Communication software is also known as network software. A user can share information with another user with communication software.

(viii)**Report Generators:** Report generator helps a user to design his own report. This method is very easy and less time consumed. Manually design of these reports is very difficult task. Commonly available report generating software is oracle reports.

3. **Service or Utility Software:** Utility software is used to provide services to facilities the working of the operating system. It helps user to take backup of data, scanning virus and retrieval of deleted files.

Some commonly known utility software are as follows:

(i) **Data Compression:** Data compression is a ability to reduce the storage requirements of a file using mathematical algorithms. It helps to store more data on a disk. Commonly used data compression software areeasyip and WinZip.

(ii) **Backup Software:** The backup software helps the user to copy large group of files from hard disk to other storage media like floppy etc.

(iii) **Data Recovery Software:** It is also known as unerase software. It is required when user delete a file accidentally from a disk. After deleting a file, user realize that file is still needed and very important. This software gets back those deleted files which were deleted by the user.

(iv) **Antivirus Utilities:** The antivirus software track the virus, eradicate and prevent their spread. The antivirus scans the disk, identify any virus and attempts to remove them. Examples are McAfee, Norton antivirus etc.

(v) **Screen Savers:**Screen saver displays moving images on the screen if no input is received from the user for several minutes. A variety of screen savers are available ranging from flying windows to hollywood personalities and desktop themes.

(vi) **File defragmentation:** A file defragmentation is used to defragment the files to speed up the working of the hard disk. It is now part of the window operating system.

Comparison between system software and application software

Sr.No	System Software	Application Software
1	It is general purpose software.	It is specific purpose software.
2	System software is a superset of application software.	Application software is a subset of system software.
3	Examples: operating system, language processor, communication software.	Examples: word processing, software database management system, spreadsheets etc.
4	System software gets installed when the operating system is installed on the computer.	Application software gets installed according to the requirement of the user.
5	System software can run independently without the presence of the application software.	Application software cannot run without the presence of the system software.
6	It is essential for a computer.	It is not essential for a computer.
7	The number of system software is less than application software.	The number of application software is much more than system software.

Table 1.2 Comparison between system software and application software

1.6 COMPONENTS/PARTS OF SOFTWARE

The components of a software means those parts which complement a software product. Software needs other existing softwares for communication. These parts are necessary for the software development. The combination of these parts make a complete software.

The following are components of a software:

1. **Operating Environment:** Operating environment describe the suitable environment where software will operate on particular hardware platform in a peaceful manner.

2. **User classes and characteristics:** Various user classes are used to anticipate the software product. User classes are based on many aspects like frequency of use, technical expertise, security levels, educational level etc. The characteristics of user classes are also described in this section.

3. **Documentation:** Documentation is the text in the written form, which explains all the operating mechanisms/methods, functions and uses of the software. It provides the full-fledged information about the software. It explains complete information of the software in a written form. It is of three types:

(i) **Architecture/Design Documentation:** This document gives an overview of the software. This type of document gives the justification of the list things which are used in the development of the software. The design documentation also explains why such things are used in the software.

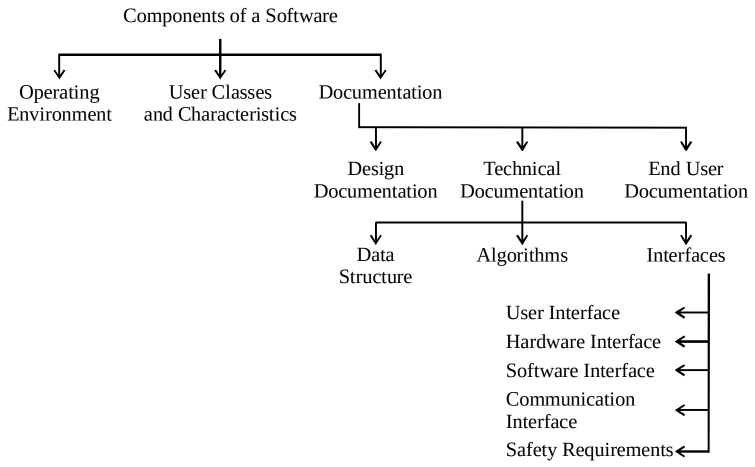


Fig. 1.2 Components of a Software

(ii) **Technical Documentation:** The technical documentation provides the coding part and various aspects of operations of the software. This documentation also explains the data structures, algorithms and interfaces:

- **Data Structure:** A data structure is a systematic way of organizing and accessing data. It explains how data are stored in a computer.
- **Algorithms:** Analgorithm is a procedure to solve a problem. The algorithm is the basic technique used to get the job done.
- **Interfaces:** It is a boundary across in which two independent system meet and act on to communicate with each other. The following are the types of interface:

(a) **User Interface:** It explains the characteristics of interface between the software product and users. It include the screen images, GUI standards, standard buttons and functions, keyboard shortcuts, error messages etc.

(b) **Hardware Interface:** It explains the characteristics of the software product and hardware components of the system. It include the supported device types, nature of data, communication protocols, control interactions between the software and hardware.

(c) **Software Interface:** It explains the connection between the software product with other specific software components, databases, tools, operating systems, messages coming into the system and going out. It also explains the communications nature and services needed.

(d) **Communication Interface:** It explains the communication functions of any product like email, communication protocols, electronic forms etc. It also describes the used communication standards like FTP or HTTP.

(e) **Safety requirements:** Safety requirements are concerned with damage, harm or loss which occurs during the usage of the product. It also explains some actions that prevents from these damages.

(iii) **End-User documentation:** This documentation helps the end users, system administrators and support staff in understanding the software. It provides all the features and steps required in the software. In short, it is a type of guide tutorials or a reference guide for a beginner. It should be very simple, manual and understandable so that end-user can easily understand it and use it.

1.7 **CHARACTERISTICS/FEATURES OF SOFTWARE**

The uses of computers are growing very rapidly. The computer industries are also growing very rapidly. The computer industries are very different from manufacturing industries in many aspects. Manufacturing industries are those industries who make/manufacture T.V.s, buildings, bikes etc. Computer industries develop the software not manufactured. Software is a logical entity not a physical entity. User cannot touch the software, cannot feel the quality of software.

The following are the characteristics of a software:

1. **Software is not manufactured or created, it is developed:** Software is first designed and then developed. Software is a logical entity, so it is developed not manufactured.Industries manufactured only those products which are in physical from, which can be touched. Software is not in a touchable form that is why it is developed.

Developers have number of automated case tools but quality of software depends upon the skills and creative abilities of the individual developers. Developers needs managing and controlling abilities to ensure the quality of the software product.

2. **Software is Highly Malleable:** Software is different and highly malleable as compared to other products. For example, we buy a new cabinet for old T.V. but old T.V. does not fit into this new cabinet. What we do? We call the carpenter to make necessary changes in the cabinet so that T.V. can fit into the new cabinet. On the other hand, if we want to change something in the software. We can directly modify the software itself rather than changing in the design of the software.

3. **Software does not wear out but deteriorates:** Software does not wear out means software is not influenced by the environmental changes. Software is a logical entity.

Software functions exactly the same way years after years without changing requirements over a period of time. If changes are necessary in the software to meet the user's new requirements, then there is a possibility of occurring of some defects which deteriorates the quality of the software.

Manufacturing companies replace the defected parts with the spare parts and maintain the quality of the product. But this cannot happens in the software industries. Software cannot replace with the spare parts.

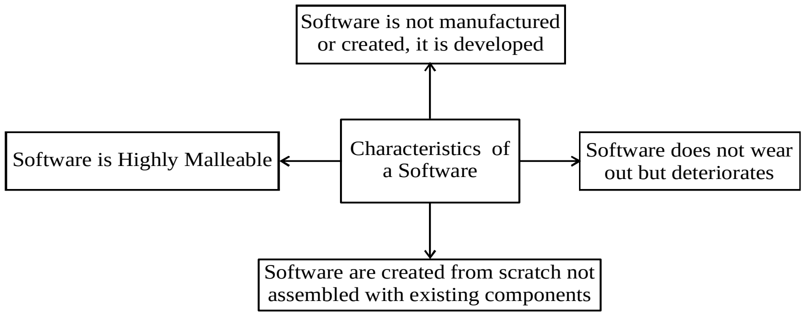

Software is not manufactured or created, it is developed
Software is Highly Malleable
Characteristics of a Software
Software does not wear out but deteriorates
Software are created from scratch not assembled with existing components

Fig. 1.3 Characteristics of a Software

4. **Software are created from scratch not assembled with existing components:** In manufacturing companies, various identifying components are put together to form the original masterpiece. Every component is flexible and independent in nature. But this approach is not applicable in software development. In software development approach, each module is inter-linked with another module. Each module is further divided into sub-modules. Different development teams are required for

different modules and sub-modules. Here modules are not assembled, modules are developed and inter-linked with each other to make a complete software.

1.8 APPLICATIONS OF A SOFTWARE

Software are used in many areas such as business, entertainment, medical, reservations etc. There is no area where software is not used.

The following are some areas where software are used:

1. **Business Softwares:** Softwares are used in business for fast, efficient and intelligent functions/tasks. With the help of softwares, companies make charts, excel sheets etc. which help in effective decision making. Companies also use software for visuals and advanced formatting.
2. **Entertainment Softwares:**Softwares are also used for entertainment purposes. There are number of entertainment softwares in the market. These softwares are extremely popular over the past decades. These softwaresincludes games, graphics, multimedia etc.
3. **Scientific Investigation and Engineering Software:** These softwares satisfy the requirements of scientists and engineers. These software are specially developed and designed for scientific and engineering purposes. These softwares are capable to perform complicated algorithms, programs etc. Astronomy and satellite launching etc. are related to scientific investigation and engineering softwares.
4. **Education Softwares:** Education softwares are known as teaching and learning tool. These software are used to teach students. These softwares are very effective in mathematics, recognizing of alphabets, read whole words and sentences etc. The primary purpose of education software is teaching or self-learning.

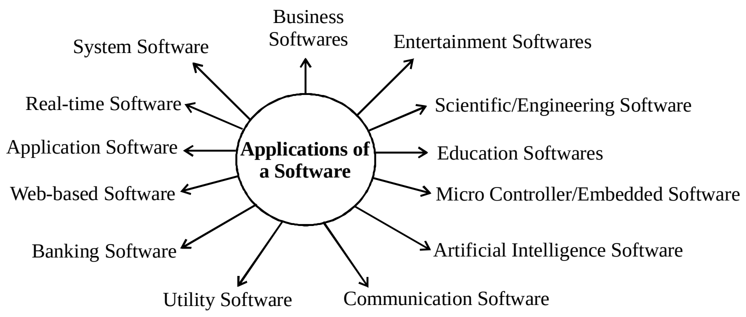


Fig. 1.4 Applications of a Software

5. **Micro Controller/Embedded Software:** Embedded software is designed for a particular kind of application service. It performs one or more predefined tasks. It is a combination of computer hardware and software. Embedded software refers to computer programs that directly interact with hardware. Example — Cars, Microwave, Watches etc.
6. **Artificial Intelligence Software:** Artificial intelligence software is an advance technique software. There softwares can understand a natural language, find out the reasons and their solutions automatically around the world. Artificial intelligence software requires human type of intelligence. Example—Expert system, Pattern recognition etc.
7. **Communication Software:** Communication software is used to share information and resources by connecting the computers. A user can share information with another user with communication software. With the help of modem and telephone line, a person can make contact with another person located at any place in the world.
8. **Utility Software:** Utility software is designed to help in management and tuning of computer hardware, operating system and application software. It is normally used to solve the common problems of software and hardware. The examples of Utility software are as follows:

- Disk defragmenters
- System profilers
- Virus scanners

- Application launchers
- Network managers
- Encryption utilities.

9. **Banking Software:** Banking software is an enterprise application solution that enables Banks to provide banking services and content to customers with efficieney and reliability within their bank premises. The banking software provides following features:

- Customer Management
- Transaction Management
- General Ledger & Accounting
- Accounting Transaction /Maintenance
- Rule Implementation / Maintenance
- Control / Configuration Management
- User-orientation
- Reporting
- Security
- Control / Configuration Management

10. **Web-based Software:** Web based software is an application package that can be accessed through the web browser. The software and database reside on a central server. Web based application gives us an opportunity to access the business information from anywhere in the world at anytime. It also facilitates us to save time, money and improve the interactivity with the customers and partners. It is also widely used for information exchange, entertainment and e-commerce.

11. **Application Software:** Application software is a computer software that makes use of the capabilities of a computer directly and thoroughly to a task that the user wishes to perform. Typical examples of software applications are word processors, spreadsheets and media players. Multiple applications bundled together as a package are sometimes referred to as an application suite.

12. **Real-time Software:** The term "real-time" refers to the ability to do or perform tasks right at the very moment they are said to be executed. Real-time software enables the user to execute various tasks and activities at the same time, as long as the programs are kept open. In computer systems, real-time operating systems contain a multitude of programs to run and operate even if the user is focused only on just one application. These software include media, building tools, computing and analysis applications, and system maintenance

13. **System Software:** System software is a type of computer program that is designed to run computer hardware and application programs. The operating system (OS) is the best-known example of system software. The OS manages all the other programs in a computer. Other examples of system software are BIOS (basic input/output system) program, the boot program, assembler, device driver, etc.

1.9**ATTRIBUTES/PROPERTIES QUALITIES OF A GOOD SOFTWARE**

Quality of a software depends upon different parameters. No single property fully describes the quality of a product. A software is called a good quality software if it possesses the following attributes:

1. **High Quality:** A good software product should possess the high quality means software should have minimum number of errors. The minimum number of errors make software more reliable.
2. **User Friendly:** Software should be user friendly. It should be usable by all types of user. Software should be easy in use. Software should provide all appropriate and adequate user interface along with the sufficient documentation.
3. **Secured and Safe:** Software should not cause any economic/physical damage. It should be secured and safe to work without any failure.
4. **Fit for intended job:** Software should fulfills all the requirements of the user. It should work according to the needs of the user and should be fit for the intended job. A quality software always fulfils the purpose of the user.
5. **Within the budget:**Software should be within the budget for the user. The cost of software should be within the packet of the user. Software should give profit to the computer industry.

6. **High Adherence (tolerance) power:** Software should respond to each and every type of input of the user. Software should behave reasonably even in the anticipated situations.
7. **Maintainable/Modifiable:**A good software should be maintainable and modifiable according to the situation. It should be changed to meet the new requirements of a user/customer.
8. **Efficiency:** Software should utilize resources in such a way that there should be no wastage of any resource. For example: memory and processor should be utilized up to the optimum level.

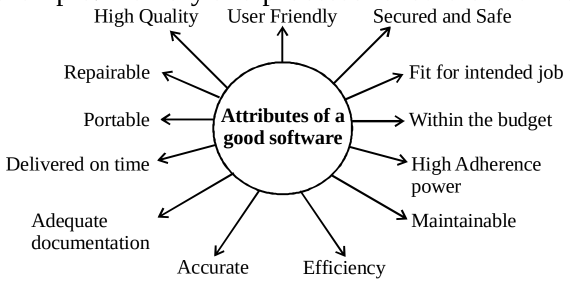


Fig. 1.5 Attributes of a good software

9. **Adequate documentation:** A software should have adequate documentation so that it can solve its purpose very well.
10. **Accurate:** Software should perform all tasks very accurately. Software should perform all operations according to the predefined specifications of the user and give accurate output to the user.
11. **Delivered on time:**Software should be developed and delivered within the specified time.
12. **Portable:**Software should be portable and have ability to work in different platforms.
13. **Repairable:**Software should be easily repairable. It should have privilege to correct the errors in a limited time period with the small amount of effort.

1.10 SOFTWARE **CRISIS**

Development of large software product with many essential features is the manual, error prone and complex process. The term software crisis is used in development of software. The term software crisis refers to the difficulty in developing a software like writing correct, understandable and verifiable computer programs. The main reasons of term software crisis are expectations, level of complexity and changeability. For example: every user demands large number of features in a required time and in a minimum budget. These expectations take the software into the software crisis.

"Software crisis means difficulty in writing correct, understandable and verifiable computer program."

Software crisis are the problems encountered in the development of the software. This is due to many issues like projects are not managed properly, late delivery of software, exceeds the budget for the development of software etc.

The following are the main reasons of software crisis:

1. Lack of communication between developers and users.
2. Lack of understanding of the problem.
3. Lack of adequate skills in the development staff.
4. Large and complex problems.
5. Increase in cost of software as compared to hardware.
6. Lack of adequate environment for developing the software.

The following are the some software crisis which are faced by many software industries:

1. Software is expensive.
2. Late delivery.
3. Software is not as per user requirements.
4. Difficult to modify/change and rework.

5. More chances of failures.
6. Brittleness.
7. Inadequate documentation.
8. Low quality of software.

In short, many software projects failed in late 1960's because these projects were late, over budget, complex, difficult to maintain and did not satisfy the client's requirements. This was known as software crisis.

Companies has been made many improvements and changes to solve these problems but problems are not decreasing. After using various methods and processes, companies are still facing new challenges and problems with the same old problems. Companies come with the conclusion that no single approach prevents the project/software over runs and failures in all cases.

In general, large complicated and poorly documented software projects have large, unanticipated problems. Whereas small projects have small problems. Researchers are still searching for the solution of the crisis.

1.11 **SOFTWARE MYTHS**

The term myth relates with mis-information or confusion. There are so many myths associated with software development. These myths affect the development of software indirectly. It means because of these myths, problems are arise in the initial stage of software development. There is no truth behind these myths. Some myths related to software development are as follows:

1. Software is good if it has many features.
2. Many software engineers cover up the delay.
3. Quality of software can be checked only after execution.
4. User requirements can be changed during any stage of software development process.
5. Reusing software increases safety.
6. Testing software can remove all the errors/bugs.
7. Software is easy to change.

1.12**SOFTWARE ENGINEERING**

1. Software engineering has different meaning and definition for many people. People explain software engineering according to their interest whereas experts give different definitions.

According to IEEE

"Software engineering is a process of systematic and disciplined approach for the development, operations and maintenance of the software."

According to Barry Bohem:

"Software engineering is the application of science and mathematics by which the capabilities of computer equipment we made useful to man via computer programs, procedures and associated documentation."

2. Software engineering ensures the development of good quality software in a minimum budget. It is the disciplined process which is concerned with all aspects of software production.

3. Software engineering follows quality standards like ISO 9001 and capability maturity model to maintain the quality in software. It focuses on various methods which are used in many software projects.

4. **Need of Software Engineering:** The goal of software engineering is to develop methods and procedures fear software development in a consistent manner to produce high quality software at low budget. Software engineering follows quality standards like ISO 9001 and capability maturity model to maintain quality in software. To achieve this goal, software development should be done in phases. Software processes should be designed and controlled properly to maintain consistency in development of software. The aim of the software engineering is to take the development nearer to the science and to concentrate on methods helpful in software development.

5. **Issues in software engineering:** The following are some issues/problems related to software engineering.

- **Technology and Management:** Developing a large software is very difficult as compared to small software. Large software needs adequate technology (methods, procedures and tools) and proper management so that software can be developed in a systematic manner in a required time without facing any problem.

Large projects are tightly managed to control the cost schedule and quality. Formal technology requirements of large software projects are high.

Whereas, small software projects can be developed with informal methods and technologies. For example: if we count the people in a room. It requires easy method and less time. But if we want to take a census of a country, then it requires proper formal methods and procedures for management and organization.

- **Consistency in terms of performance and quality:** Consistency of performance and quality is very important for all computer industries. The aim of computer industries is to develop a software with low budget and high quality in a consistent manner. Computer industries want consistent quality with improved productivity but it is very difficult task. A good quality software in a low cost is a challenge for computer industries.
- **Expensive:** Software development needs resources like hardware, software, manpower etc. in a low budget. But it is a challenge because software development is a labor intensive, time consuming and costly method.

1.13 EVOLUTION OF SOFTWARE ENGINEERING

Evolution of software engineering means take a look to check how software engineering been over the years. With the passage of years, scope and meaning of software engineering enhanced with number of issues.

During 1960, software was developed by a single programmer or by a small team of programmers. In this period, programmer can draw flow diagrams using language tools, write programming codesaccording to his/her choice. In 1970, team of programmers became larger on the basis of type of software project. Cost estimation was not considered but programming languages like FORTRAN and quality assurance got introduced. During the period of 1980's and 1990's, standards and methodologies were introduced for ensuring quality and reducing cost in software product.

Computer aided software engineering (CASE) companies use no of tools (Knowledge Based Systems and Total Quality Management) to improve the software development process.

1.14 SOFTWARE ENGINEERING — A LAYERED TECHNOLOGY

Software engineering follows the layered technology which are as follows:

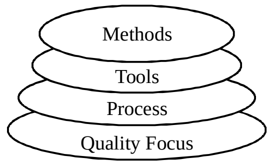


Fig. 1.6 Software Engineering — A Layered Technology

1. **Methods:**It provides technical support for building software. It encompasses many tasks including communication, requirement analysis, design modeling, program construction, testing and support.
2. **Tools:**Tools provide automated or semi-automated support for the process and methods.
3. **Process:** It is a framework with activities for effective delivery of software engineering technology. A process is a collection of activities, actions and tasks that are performed when some work product is to be created. The purpose of process is to deliver software in a timely manner and with sufficient quality to satisfy those who have sponsored its creation and those who will use it.

4. **Quality Focus:** Any engineering approach must rest on organizational commitment to quality which fosters a continuous process improvement culture.

1.15 ROLE AND RESPONSIBILITIES OF SOFTWARE ENGINEER

An engineer is someone who uses advanced knowledge of science, mathematics, and technology to build objects for use by others. Whereas software engineer is responsible for the complete life cycle of a new/modified software product from design to implementation. The roles and responsibilities of software engineer are as follows:

1. Software engineer is a good programmer and have a good knowledge of data structures and algorithms. Software engineer is expert in more than one programming language and have creative or logical thinking mind.
2. Software engineer is familiar with many design approaches and have ability to move among several levels of abstraction of different stages of the project, to a specific design for system and at the end to the detailed coding level.
3. Software engineer consult with systems analysts, programmers and others to collect the limitations, capabilities and performance requirements of software product.
4. Software engineer modify the existing software to correct errors, allow it to adapt to new hardware, or to improve its performance.
5. Software engineer analyze the user needs and software requirements to determine feasibility of design within time and cost constraints.
6. Software engineer coordinate software system installation and monitor functions to ensure specifications are met.
7. Software engineer analyze information to determine, recommend, and plan computer specifications and layouts, and peripheral equipment modifications.
8. Software engineer supervise the work of programmers, technologists and technicians.
9. Software engineer utilizes software engineering tools such as configuration management systems, build processes, and debuggers in the software development process.
10. Software engineer serve as a mentor to less experienced software engineers.
11. Software engineer also provides training to new users and handles support and feedback.
12. Software engineer produce well-organized, optimized, and documented source code. Contribute to technical design documentation.
13. Software engineer continuously learn and improve skills and can work independently when required.

1.16 SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)

Software development life cycle is also known as product development life cycle (PDLC). It is also called software development process (SDP). Software development life cycle has many phases and activities for software development. Each phase ends with a defined output which is the input for the next phase.

The goal of SDLC is to develop a high quality product in minimum cost by checking the progress at the end of each phase. In all the phases, complexity of tasks and project tracking etc. should be managed to maintain the quality of the software product. It should also help to solve the software crisis.

SDLC varies with large projects and small projects. Small projects have different phases, activities and issues as compared to large software projects. Large software projects should require proper documentation with user specifications whereas small softwares can be developed without written record of activities.

Key Points:

- In the SDLC, basically the major activities related to software development are performed.

- SDLC is a sequence of steps/phases to perform the well-defined activities to fulfill the desired goals. In SDLC output of one step/phase is the input of next step/phase.
- There is a criteria for each input/output phase. Input criteria specifies the conditions when to terminate the activities of the phase output criteria specifies the conditions when to terminate the activities of the phase.
- The goal of the SDLC is to detect the errors/defects in the starting of their origin in the phase. Validation and verification should be performed at the end of the phase to detect the errors.
- Due to importance of development process, various models have been proposed for developing a software which are being discussed in the this chapter. [2.3]
- Clear objective is defined in the starting of the development process to make it more effective and less expensive.

There are seven phase of SDLC:

1. Feasibility Analysis
2. Requirement Analysis
3. Design
4. Coding
5. Testing
6. Implementation
7. Maintenance

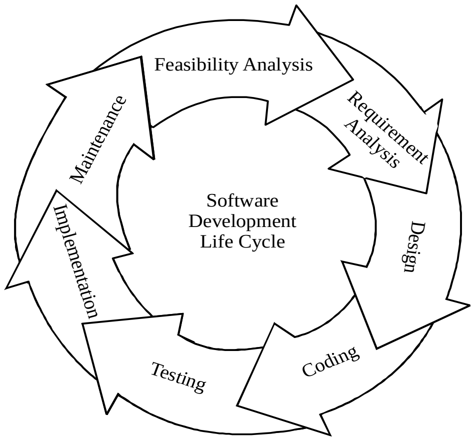


Fig. 1.7 Phases of SDLC

1. **Feasibility Analysis/Feasibility Study:**

- **Requirements Gathering:** The Business Analyst (BA) or a representator of a company collects all the predefined information of the company like what type of softwares a company can develop and in how many days etc. and goes to the client. Then BA collects all the basic requirements and specifications of the client and prepare a document known as BDD (business development design), BRS (business requirement specification), URS (user requirement specification), and CRS (customer requirement specification) etc. All these documents are same, only name is different. These documents are the output of the feasibility study phase and input of the requirement analysis phase.

- **Types of Feasibility Study:**

(a) **Technical feasibility study:** In the technical feasibility study, possibility of project development is measured. It is checked that whether the software project is technically sound or not.

(b) **Economic feasibility study:** In the economic feasibility study financial matters are discussed. For example : Is the software project is under budget or not? A software project should be under budget.

(c) **Schedule feasibility study:** In the schedule feasibility study, time duration is decided. The company finds the probability to complete the project in fixed allowed time period.

(d) **Motivational feasibility study:** In the motivational feasibility study, the sufficient motivation of organization is checked with necessary user participations, resources etc.

2. **Requirement Analysis:** The output of the feasibility study phase CRS (customer requirement specification) is used as input in the requirement analysis phase. Generally project manager analyze the requirements of the user and prepare the project plan.

Requirement analysis phase has 4 steps which are as follows:

(a) **Requirement Analysis:** All the requirements of client (mentioned in CRS) are studied and analyzed in detail. To understand the requirements of clients for large projects with many features, different tasks are analyzed. But it is very difficult because of communication gap between client and developer. This type of difficulty is based on basically two aspects. In one aspect, developer tries to develop a software which meets client's requirements without understanding the client's problem domain. In second aspect, client does not aware about the issues/problems in the development of the software.

(b) **Deciding Technology:** In this step, suitable technology is being declared for the development of software project. Companies select suitable technology based on many factors like requirements of clients, type of software project etc.

(c) **Estimating the Resources:** In this step, resources like manpower, time, money etc. are estimated for the development software project. All resources should be examined and improved in advance to reduce the difficulties in the software development.

(d) **Preparing SRS (Software requirement specification):**

- SRS (Software Requirement Specification) is a last and fourth step of requirement analysis phase. It is a document with the specific language to specify all the inputs, outputs, functional requirements, performance requirements with some constraints like economic issues, security etc.
- Definition: SRS is the specification for particular software product that perform certain functions in a specific environment.
- SRS is a complete description of the software which is going to be developed. It describes the complete scope of the product.
- SRS is prepared by Senior Analyst (SR).
- It is a type of agreement between the client and the company on what the software product is going to do.
- All functions/tasks of software development are based on the software requirement specification.

3. **Designing:** In the design phase, a plan is proposed for software development according to the software requirement specification. The quality of a software depends upon the design document. Design document is like a blue-print or plan for the software development which is used in later stages.

The design activity of a software development is divided into two levels.

(a) **High Level Designing/System Design**

- It is also known as top-level design. It is done by Technical Manager (TM).
- In this designing, software is divided into number of modules. Modules are identified along with their specification and interaction among them. In this type of designing, focus is on identifying the modules.

(b) **Low level designing/detailed design:**

- It is also known as detailed design. It is done by Team Lead (TL).
- In this designing, modules are further divided into number of sub-modules.
- Internal logic of sub-modules are specifies in the high level descriptive language.
- The output of this phase is detail design document (DDD) which is further used in later phases to develop a high quality software.

4. **Coding:**

- Once the designing of software is complete, the coding phase comes, in which developers follow some coding standards for writing the programs.
- In this phase, design of a software product translates into simple and understandable code in a particular language.
- Developers prepare the source code which is easy to write, read and understand.
- Coding should be done with simplicity, clarity and well-written format. Effective coding can reduce the testing and maintenance effort.
- Coding part is very time consuming and difficult task. Coding should be done in structured programming which linearizes the control flow in the programs.

5. **Testing:**

- The quality of a software product is measured and controlled by the testing phase.
- Testing is a process of executing a program with the intent of detecting or finding errors. Proper planning is required for the testing.
- Testing is very difficult and time-consumingprocess.

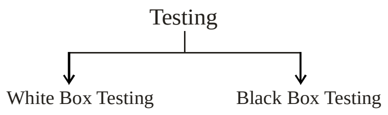


Fig. 1.8 Types of Testing

- Before starting this process, test engineers study the CRS (customer requirement specification) document prepared in the first phase and prepare a review report.
- In the review report, test engineers mentioned those points which are not clear and understandable. Test engineer send this review report to the Bussing Analyst (BA).
- Test Engineer prepare test plan before testing. Test plan specifies the scope, approach, resources and schedule of all testing activities.
- Test plan covers all the activities like items to be tested, types of testing to be performed, features to be tested, persons/resources responsible for testing, time scheduling issues/risks associated with the plan etc.
- Testing starts with the unit testing, in which a module is tested separately then integration testing is performed, which detect design errors and in last, system testing is performed to check the system performance according to the software requirement specification (SRS).
- The output of the testing phase is known as defect profile document (DPD). Test Engineers sends this document to the team of developers for removing the errors and controlling the quality of software.

6. **Implementation:**

- In this phase, after completing the software project, a mail is sent to the client. This mail is known as software delivery note.
- It after receiving this mail, client test the software project known as user acceptance testing. After this successful testing, software is installed in the client's environment.

7. **Maintenance:** During installation of software product in the client's environment, if any problem occurs or if client wants to make some changes in the software, then the maintenance people make changes to solve the problem and prepare the deployment document (DD) for further use.

1.17 SOFTWARE DEVELOPMENT PROCESS MODELS

1. A software development life cycle models describe the necessary sequence of different activities helpful in software development.
2. Software development starts with the request of client and undergoes many stages until software product is developed and delivered to the client.

3. Thses models show series of identifiable activities of software development for the successful delivery of the software product.
4. In the SDLC models, several activities and several documents are carried out. Documents are normally made to collect all the input/output information. Different phases of software development are shown graphically with the textual description for easy understanding.
5. In short, these models give descriptive and diagrammatic representation of software development phases.
6. There are so many software development life cycle models like waterfall model, spiral model etc. The common aim of these models is to develop a good quality software.
7. No matter which life cycle model is followed, all models have basic activities, carrying process of these models may be in different orders, but goal will remain same.
8. These models are helpful in developing a high quality software with low budget and time constraints.

There are so many life cycle models. The following are some life cycle models commonly used by computer organizations:

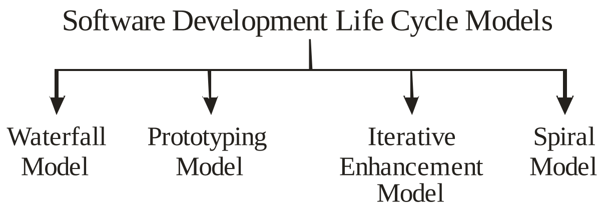

Software Development Life Cycle Models
Waterfall Model
Prototyping Model
Iterative Enhancement Model
Spiral Model

Fig. 1.9 Models of SDLC

1.18 WATERFALL MODEL/LINEAR SEQUENTIAL LIFE CYCLE MODEL

1. Waterfall model is the simplest and oldest process model.
2. It is represented by sequence of different stages.
3. The output of one stage flows into the second stage, the output of second stage flows into third stage and so on.
4. It is a definite structure of software development life cycle. Each phase is distinct and has a well-defined entry.
5. There is a provision for verification, validation and error correction at the end of every phase.
6. The part of waterfall model between the feasibility study and system testing is known as development part. After the development part, the software product is delivered to the customer. The maintenance phase comes after the delivery of the developed product to the customer.
7. This model is also known as the linear sequential model or the software life cycle model.
8. The waterfall model consists of following six stages:

(i) **Feasibility Study:** The feasibility study determines whether the developing model is financially and technically feasible or not. It is necessary to analyze the problem. More we understand the problem and the better we can identify alternative solutions. The feasibility study is usually done within limited time bounds. The outcome of feasibility study is a document that should contain at least the following factors:

- A definition of the problem.
- Determination of technical and economic viability.
- Alternative solutions and their expected benefits.
- Requirement of resources and detail about delivery

This report is called as feasibility study which is prepared by a group of software engineers. The client or the customer is also consulted through questionnaires. The report determines whether the project is feasible or not.

(ii) **Requirement Analysis and Specification:** In this phase, we exactly analyze the requirements and needs of the project. The purpose of a requirement analysis is to identify the qualities required for the application, in terms of functionality, performance, ease of use portability, and so on. The result of this phase is known as the software requirement specification (SRS) document.

An SRS document must contain the following items:

- Detailed statement of problem.
- Possible alternative solution to problem.
- Functional requirements of the software system.
- Constraints on the software system.

The SRS document must be precise, consistent, and complete and covers features like problem statement, introduction to the problem, functional requirements of the system, non-functional requirements of the system, behavioral descriptions and validation criteria.

(iii) **Designing:** The objective of the design phase is to convert the requirements specified in the SRS document into a structure that is suitable for execution in some programming language. Two different design approaches are available: the traditional design approach and the object-oriented design approach.

- In Traditional Design Approach, two activities are carried out, first a structured analysis of the requirements specification and second structured design. During structured design, the results of structured analysis are transformed into the software design.
- In Object-Oriented Design Approach, the various objects and their relationships within the system are identified.

(iv) **Coding and Unit Testing:** Coding and unit testing is the phase in which we actually write programs using a programming language. The output of this phase is an implemented and tested collection of units. The unit testing is the testing of code to check correctness and remove the bugs. The debugging is a related activity performed in this phase.

(v) **Integration and System Testing:** During the integration and system testing phase, the modules are integrated in a planned manner. The different modules are integrated to develop a software product. During each integration step, the partially integrated system is tested finally, when all the modules have been successfully integrated and tested, system testing is carried out. The objective of system testing is to determine whether the software system performs per the requirements mentioned in the SRS document.

The system testing is done in three phases: Alpha, Beta, and Acceptance Testing.

- **Alpha Testing** is conducted by the software-development team at the developer's site.
- **Beta Testing** is conducted by a group of friendly customers in the presence of the software-development team.
- **Acceptance Testing** is performed by the customers themselves. If the software is successful in acceptance testing, the software product is installed at the customer's site.

(vi) **Delivery:** The delivery of software is done in two phases. In the first phase, the application is deployed on few customers' sites. The purpose of this procedure is to check the product performance at client site, on the basis of feedback from users, whether any changes is necessary or not. In the second phase, finally the product is distributed to the customer's officially.

(vii) **Maintenance:** The maintenance as the set of activities that are performed after the system is delivered to the customer. Its purpose is to monitor the performance of the software, and removing error if nay faced by the customer in site.

$$\begin{tikzcd}
\text{Feasibility Study} \arrow[d] \\
\text{Requirement Analysis and Specification}
\end{tikzcd}$$

$$\begin{tikzcd}
{} \arrow[d] \\
\text{Designing} \arrow[d] \\
\text{Coding and Unit Testing} \arrow[d] \\
\text{Integration and System Testing} \arrow[d] \\
\text{Delivery} \arrow[d] \\
\text{Maintenance}
\end{tikzcd}$$

Fig. 1.20 Steps of Waterfall Model

Advantages of Waterfall Model

The various advantages of the waterfall model include:

1. It is a linear model.
2. It is a segmental model
3. It is systematic and sequential.
4. It is a simple one.
5. It has proper documentation.

Disadvantages of Waterfall Model

The various disadvantages of the waterfall model include:

1. It is difficult to define all requirements at the beginning of a project.
2. The model is not suitable for accommodating any change in later stage.
3. It does not good for large scale projects.
4. It involves heavy documentations.
5. We cannot go backward in the SDLC.
6. There is no prototyping model for clearly realizing the customer's needs.
7. There is no risk analysis.
8. If there is any mistake in any of seven phases then we cannot develop good software.
9. It is a document driven process that requires formal documents at the end of each phase.

1.19 **PROTOTYPING MODEL**

1. Prototyping model is an intuitive approach of the waterfall. It overcomes the limitations of waterfall model.
2. Multiple development cycles take place in prototyping, making it multi-waterfall cycle. Cycles are further divided into smaller and easily manageable iterations.
3. The interactions in the prototype enable the developer to better understand what needs to be done and to satisfy users.
4. The prototype model is helpful in understanding the currently known requirements. In the waterfall model, client has to specify his/her requirements before the designing and coding phase. Client's requirements are freezed in the requirement phase. But in the prototyping model, client can specify his/her requirements in more detailed manner.
5. In this model, the client can interact with the prototype and can get an actual feel of the system.
6. Prototyping model is suitable for those projects where it is very difficult to determine the specifications of complex and large projects like client's feedback regarding software product, what is correct and missing, what needs modification etc.

7. Developers make changes in the software product on client's suggestions. Clients use the new changed software product again and allowed to give his/her suggestions to the developers. This cycle repeats until client satisfies with the software product.
8. Prototyping ensures that end users constantly work with the system. Prototyping model is excellent for designing good human computer interface systems.
9. The steps of prototyping model are as follows:

(i) **Requirements gathering and analysis:** The model begins with requirement gathering and analysis in detail. In this phase, requirements are gathered and analyzed by the organization.

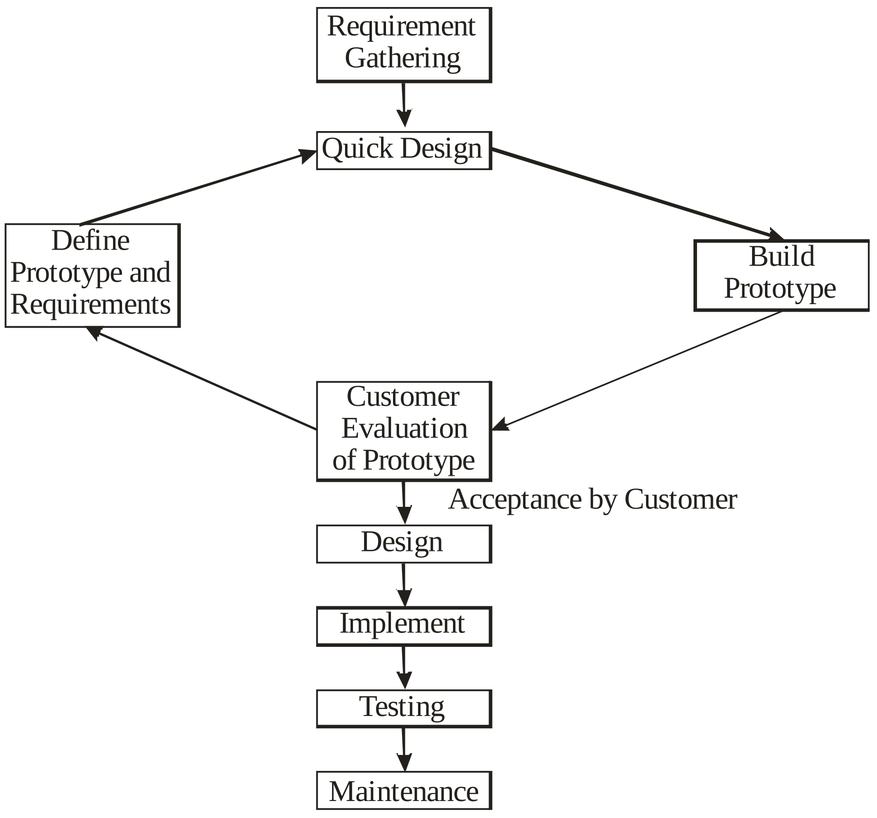
(ii) **Quick Design:** After analysis of requirements, quick design is created. It is a rough design not a detailed design. This quick design helps in developing a prototype and provides an idea of the software product to the user.

(iii) **Build Prototype:** With the help of quick design, a prototype is build which represents the working model of the required software product.

(iv) **User Evaluation:** A proposed prototype is evaluated by the user. User studies the strengths and weaknesses of this prototype. User provides his/her comments and suggestions like what is to be added or removed etc. to the developer.

(v) **Refine Prototype:** If after the evaluation of the prototype, user does not satisfy, then prototype is refined according to the requirements. A new prototyped is developed and being evaluated by the user. After the satisfaction of user, final system is developed.

(vi) **Engineer Product:** Once the user accepts the software product is maintained on regular basis.



Prototyping Model

Fig. 1.21 Prototyping Model

Advantages of Prototyping Model:

The various advantages of the prototyping model are as follows:

1. **Quick development:** Developer's approach is to generate software product quickly according to the client's requirements.
2. **Flexible:** Prototyping models are more flexible as compared to waterfall model. Clients can give suggestions to developers for the changes in the software product. In this way, model reduces the communication gap between client and developers.
3. **Easy to test and debug:** Developers can easily test and debug a smaller iteration rather than the whole software product.
4. **Manage risk:** During the small iteration, risky pieces are identified and handled easily.
5. **User Involvement:** Users are actively involved in the software development process.

Disadvantages of Prototyping Model

The various disadvantages of the prototyping model are as follows:

1. Each phase of iteration is rigid and do not overlap each other.
2. **High Cost:** Experienced developers and changing requirements are two main reasons of higher cost in prototyping model.
3. **Lower Quality:** During the development of software product according to prototyping model, the developer's focus is on quick development without thinking about the quality of software product. Development approach is to satisfy client's needs without bothering about the quality of software product.
4. **Compromises in Implementation:** Developer's aim is to get prototyping quickly and to meet client's desires. To achieve this aim, developers make many implementation compromises like inappropriate operating system, an inefficient algorithm etc.
5. **Time Consuming Process:** It is a slow process. The aim of prototyping model is to satisfy the customer. But requirements of customer keeps changing according to external environment. Hence,it is time consuming process.

1.20 ITERATIVE ENHANCEMENT MODEL

1. Iterative enhancement model provides the benefits of both waterfall model and prototyping model.
2. According to the iterative enhancement model, each increment adds some functional capability to the system until the full system is implemented.

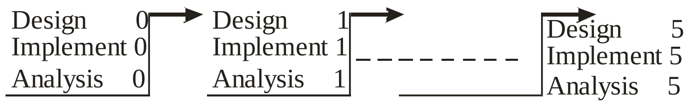


Fig. 1.22 Iterative Enhancement Model

3. The basic idea behind this model is to develop a system through repeated cycles (iterative) and in smaller portions at a time.
4. This model is used when requirements of system are clearly defined and understood.
5. Iterative enhancement model supports changing requirements.
6. Parallel development can be planned and progress can be determined during iterative enhancement model.

Advantages of Iterative Enhancement Model

1. Iterative enhancement model is better in testing. It is because testing increment is very easy as compared to testing the entire system. It overcomes the limitation of waterfall model.
2. Like prototyping, this model also provides feedback to the client. This feedback is very useful for determining the final requirements of the system.

3. This model also provide results periodically and early. It allows the users to get some early experience of the software.
4. Risks are identified and managed during iteration. During smaller iteration, testing and debugging is easy.
5. This model is suitable for large and mission critical projects.

Disadvantages of Iterative Enhancement Model:

1. This model is not suitable for small projects.
2. During iterative enhancement model, management complexity is more and highly skilled resources or required for risk analysis.
3. Sometimes, the iteration may never end and the user may never really get the final product.

1.21 **SPIRAL MODEL**

1. Spiral model was developed by Barry Bohem in 1988. In this model, software is developed in spiral with many loops.
2. The number of loops in a spiral model are not fixed and can vary from project to project. Each loop is the phase of the software project.
3. At each loop, risks are identified and resolved by prototyping. At every spiral, risk analysis is done to evaluate the development efforts/labour and risks.
4. This model is very flexible because number of phases are not fixed for the software development.
5. Inner cycles/loops represent the requirement analysis alongwith the prototype, whereas outer cycles represent the waterfall model.
6. Generally, spiral model is combination of strengths of various models. It uses stepwise approach of waterfall model and prototyping for resolving the risks occurred during software development.
7. Spiral model is also called as metal model because both waterfall model and prototype model are used in it. It reduce risk as well as follow systematic approach.
8. **Phases of Spiral Model:** There are four phases in the spiral model. In the starting of this model, requirements are gathered to analyze the risk. Spiral model makes communication effective between customer and developer. In this model, angular component represent the progress and reduces the represent cost.

The four phases of spiral model are discussed as follows:

(i) **Planning:** In the planning phase, requirements are gathered. All methods and strategies are decided in this phase. The objectives, constraints and alternatives are also discussed in this phase.

(ii) **Risk Analysis:** A process is undertaken to identify the risks. In this phase, alternate solutions are also identified to resolve the risk. Prototyping is used at the end of this phase to resolve the risks which are identified and analyzed. In this phase, risks associated with these new alternative solutions are also analyzed. In this phase, keeping new risks in mind, strong decisions are taken to resolve the risks with prototype.

(iii) **Engineering:** It involves the actual development of software project. After finding and resolving all the risks through prototyping, software is being developed in the engineering phase.

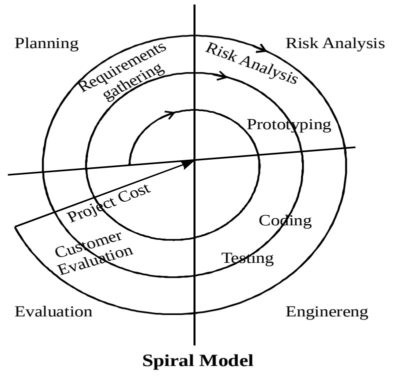


Fig. 1.23 Spiral Model

(iv) **Evaluation:** This phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.

Advantages of Spiral Model

1. Spiral model evaluates the risk analysis. It enhances the avoidance of risk.
2. It is good for large and mission critical objects.
3. Software is produced early in the software life cycle.
4. Strong approval and documentation control.

Disadvantages of Spiral Model

1. Sometimes, it is costly in use.
2. It is not suitable for small projects.
3. The success of spiral model depends upon the risk analysis phase.
4. Risk analysis requires highly specific expertise.
5. It is not suitable for low risk projects.
6. Spiral may continue indefinitely.

BACHELOR OF COMPUTER APPLICATIONS (BCA)
BCA-4-01T: SOFTWARE ENGINEERING

UNIT 2: SOFTWARE PROCESS

2.1 **INTRODUCTION**

Software is developed through a set of activities which are essential for software development of software; a collection of procedure is adopted to achieve certain goals or standards. Therefore the process that deals with technical and management issues of software development in mind is known as software process.

Computer software is loaded into the computer memory and the computer becomes capable of operating the software. The instructions are passed from application software to the hardware through the systems software. The following points are important to be noted about software process:

1. Processes use resources which under terms and conditions product intermediate and final product.
2. Processes are made up of sub processes and each sub process has its own process model.
3. A process has entry as well as exit criteria which controls and monitors the beginning and completion of an activity.
4. A process includes guidelines and it forces uniformity in all activities.
5. Many technical and managerial issues are needed to develop a software.

2.2 **CHARACTERISTICS OF SOFTWARE PROCESS**

A software process has so many characteristics which have been listed below:

1. **Comprehensiveness**: The process should be easy to understand the user.
2. **Visibility**: The progress of the process should produce clear results.
3. **Robustness**: The process should continue even if there is a unexpected problem.
4. **Supportability**: The process should be such as to support the process activities. It should also make the testing easy and it should be able to produce the software easy to maintain.
5. **Acceptability**: The process should be acceptable to the engineers and it should be usable.

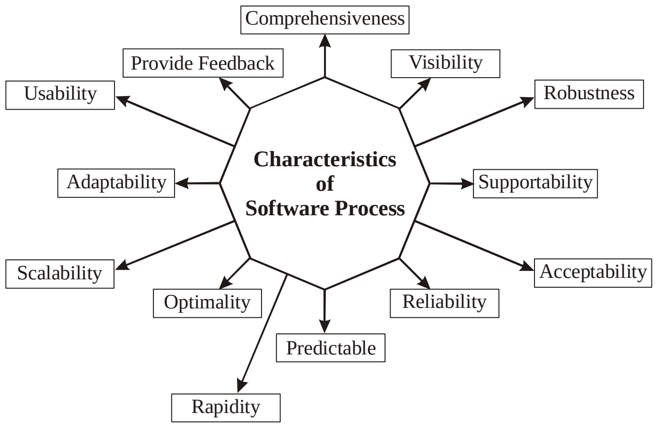


Fig. 2.1 Characteristics of Software Process

6. **Reliability**: The errors in the processes should be avoided or trapped before they result in errors to the product.
7. **Predictable**: The outcome of the project should be predicted before the project is completed.
8. **Rapidity**: The process of delivering the system with given specifications should be rapid.
9. **Optimality**: The process should be able to produce high quality software at a very cheap rate.
10. **Scalability**: It should be applicable for large scale software projects.
11. **Adaptability**: The process should adaptable to the changing requirements.
12. **Usability**: It should have proper user interface and adequate document.
13. **Provide feedback**: There must be feedback information for improvement.

2.3 **PROJECT MANAGEMENT PROCESS**

2.3.1 **Definition**

It can be defined as an activity to achieve an objective through the wisely use of knowledge, skills, tools and techniques by the Project Manager.

Project Management is an artistic use of various resources so that the project is completed on time and within budget. It should provide satisfaction to the End user. Project Management is combination of control, Leadership, Team work as a unit, resource management etc.

Process Groups

For the successful completion of a project; a proper system of different activities need to be followed. These processes can be placed into five inter-related process groups: 1. Planning, 2. Organisation, 3. Staffing, 4. Directing, 5. Controlling.

The main job of management is to enable a group of professional to work towards a common goal.

1. **Planning:** This is an important part of project management. The following points are kept in mind by the project management.

(a) What objectives are to be achieved?
(b) What resources are needed?
(c) How and when the resources are to be acquired?
(d) Ways and means to achieve the goal.
(e) To determine the flow of information, people and products within the system.

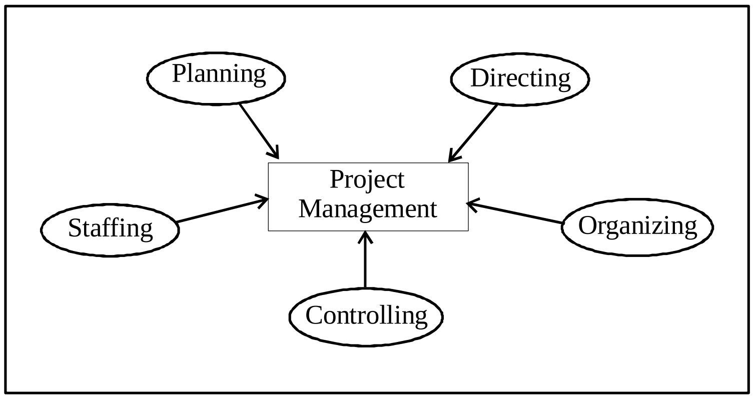
2. **Organizing:** It involves the division of duties and responsibilities among different groups in such a way that the goals of the enterprises are achieved. This is possible only when proper planning is done by the professional within the groups. Best organization means best results in the end.

3. **Staffing:** It deals with hiring of qualified and well trained personnel for the positions that are recognized by the organizational structure. It carries the process of recruiting, hiring, training, rewarding, retaining, compensating, developing and promoting employees.

4. **Directing:** It is the process guiding and leaving the subordinates so that they may understand and identify the organizational structure and the goals to be achieved by the enterprise.

5. **Controlling:** This step includes the process of measuring and correcting activities. The performance of each group is measured against plans and if there is any deviation, then corrective actions are taken to achieve the desired goal.

Software Project Management is based on sound technical skills and Management Skills. The Technical Process of software development depends on the application of sound management principles. It is not possible for one person to handle today's large and complex programs therefore team work is necessary under the leadership of the project manager.



Components of Project Management Process

Fig. 2.2 Components of Project Management Process

2.3.1.1 **Characteristics of Project**

A project is a temporary endeavor which is taken to provide a unique product or service. A project is different from production work in the sense that all projects have a beginning and an end. Projects are of different size. If can be quite small such as the repairing of a sofa set. Projects can be too big involving thousands of people and investing large sum of money. A project can take place at any level of organisation or even can include the whole organisation. A project can be of small duration (days or weeks) or long duration (to be completed in several month years).

Projects are of two types:

(i) **Temporary Projects:** Each and every project has a beginning and an end. The project can be closed due to the following reasons:

(a) Project's objective is achieved.
(b) When if become, clear that it is difficult to achieve the objectives in a practical way.
(c) The project is no longer required now.

(ii) **Unique Projects:** A project becomes unique when if tries to improve upon what was done in the past. Everything is planned and organized to bring a new product with new specifications.

A successful completion of a project demands better planning, well defined organization technical monitoring, project scheduling, cost, time and effort estimations. The proper measurements and metrics make a project worth its price and qualities. Composition of teams, assigning different tasks, providing resources and establishment of project standards are the major characteristics of a good project

2.3.1.2 **Project Manager Manages the Project**

Project Manager

The project manager is the most important individual who uses both the process and set of tools and techniques to define the aims and objectives of the project. The project manager plans the work to achieve the goals. He leads the project and support teams, monitors progress and so manages everything that the project is completed in a satisfactory way.

The software project manager keeps an eye on day by day operations. He can have the responsibility for multiple projects. The need for a good project manager has risen properly in the industry and commerce. Project manager is the central axis of the project-based organizations because it is establishing itself both as a professional career path and a way of controlling business.

The main responsibility of the project manager is to develop project plans including the work breakdown structure, time estimated, planning resources, training plan, risk management plans and identifies other plans. He uses PERT chart/Gantt chart etc. for scheduling all the tasks. **The main responsibilities of the project manager are:**

1. Define Goals: He defines goals, plans and monitor's tasks and resources. He recognizes and find solution to the issues and in addition to it controls costs and budgets.
2. Leads System Development: The project manager showed have a good understanding to the project management task and he leads system development on an expert system developer.
3. Manages and organizes the primary resources needed to complete the project. He interacts with people linked to the project and mobilizes resources and equipment very well.
4. Control and co-ordinates every aspect of the project and motivates the team members to achieve the project objectives.

To conclude we can say that the project manager, plays a vital role in selecting everything and organizes, plans and controls the development process. He communicated with different persons, participants in meetings and gives presentations.

2.3.2 **Four P's in Project Management**

A successful venture is the one where the project manager successfully builds the right product using the right process for the right project with the involvement of efficient people.

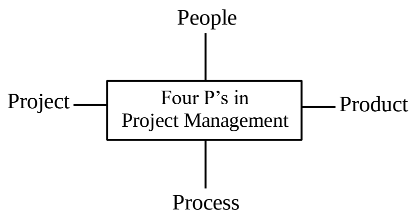


Fig. 2.3 Four P's in Project Management

1. **People:** The skillful and efficient people are the greatest asset of a software organization. The people represent the intellectual capital and efficient managers get best out of their investment in people. Success comes to those organizations which respect the people in the project team. The following people are involved in the software process:

(i) Senior Managers: Defines the business issues.
(ii) Project Managers: Who man, motivate, organize and counting the programmers?
(iii) Programmers: Who deliver the technical skills needed to develop a product?
(iv) Customers: Who specifies the requirements?
(v) End users: Who actually interact with the software once it is developed?

The people who work in a project have to work as a cohesive group and contribute their best to the project. They will have to be organized because the team structure has a direct influence on the product quality and project productivity. The project manager has to solve a lot of problems with the help of the people in his team in the most effective manner. He has to motivate plan and organize the work of the programmers for the proper functioning of the work. The managerial activities of the project manager and his leadership skills have a great influence in building a letter team. He must have the client management skills too. He should be able to communicate effectively with his clients.

2. **Product:** The software product is a tangible item that is produced by a project according to the required specifications. The project manager has to identify opportunities for the use of the product. He must explore the idea about the product in detail. For this scope of the project must be established.

Software Scope: The software product must be built in such a way to fit into a larger system. What constraints are needed must be known in advance. Software project scope must be unambiguous and understandable at the management and technical levels. Quantitative data are stated explicitly, Limitations are noted and mitigating factors are described.

Problem Decomposition: It is an activity that is important for software requirement analysis. Decomposition of the problem is applied in two major areas:

(i) The functionality which must be delivered
(ii) The process which will be used to deliver it. Software functions are evaluated and refined to furnish more detail prior to the beginning of estimation. As cost and schedule estimate are functionally oriented; some degree of decomposition is often useful.

3. **Process:** The project manager must decide to select the process model which is most appropriate to be produced by the project team the first requirement is to build the project environment in while the software team works. The attributes of the product must be kept in mind. After the selection of the process model, the preliminary plan is established and process decomposition begins the organization adopts a set of framework activities which include:

(i) Customer communication
(ii) Planning
(iii) Risk analysis
(iv) Engineering activities
(v) Documentation and Training
(vi) Customer evaluation etc.

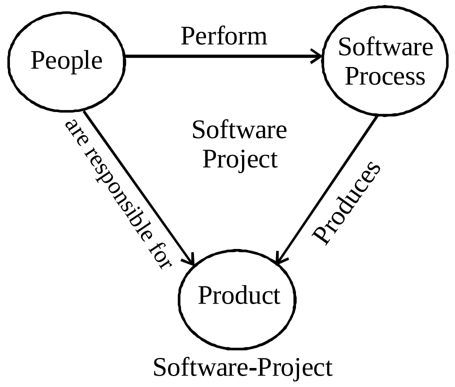
A software team must have the degree of flexibility in selecting the software engineering paradigm which suits well for the project. A relatively small project similar to the post efforts is chosen. If the deadline is very tight then an incremental strategy is adopted. Similarly projects with other characteristics are selected. In the end a more complex project with a broader scope and significant business impact is adopted.

4. **Project:** The software project management activity is just like an umbrella activity within the software engineering process. It begins before the start of any technical activity and continues throughout the definition, development and support of computer software. The project management activities include:

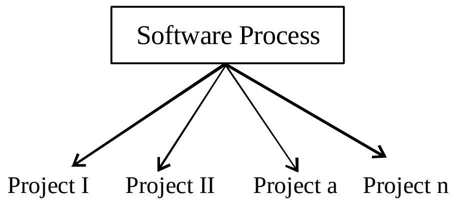
(i) Measurement and Metrics
(ii) Management activities
(iii) Project planning
(iv) Scheduling
(v) Tracking
(vi) Risk management activity

To avoid problems in the project activities, the project team follows the five past common-sense approach i.e. (i) Start on the right tool, (ii) Maintain Momentum, (iii) Track process, (iv) Mask smart data.

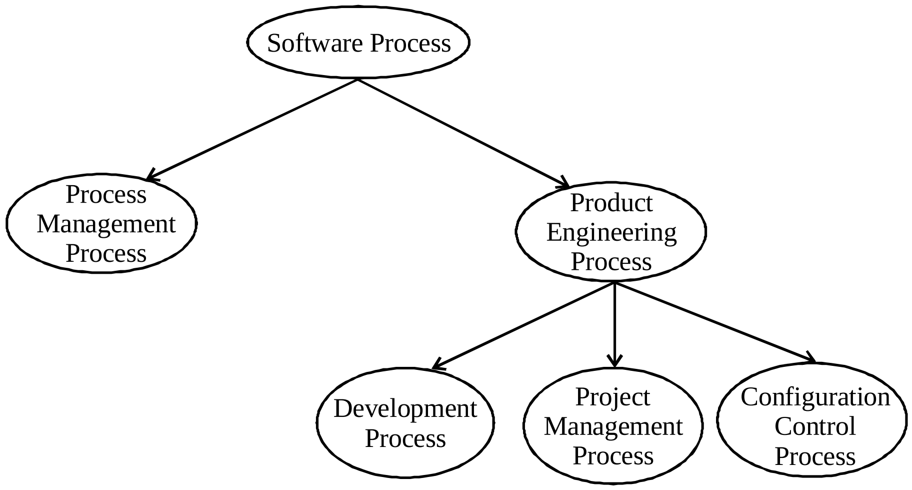
The inter relationship between 4 P's is best illustrated with the diagrams as under:



(A) Inter-Relationship between 4 P's



(B) Inter-Relationship between Process & Project


Software Process
Process Management Process
Product Engineering Process
Development Process
Project Management Process
Configuration Control Process

(C) Process Categories

2.3.3 Activities of Project Management

The following are the major activities which are to be performed by the Project Manager for successful management of the software project:

1. **Proposal Writing:** These are no set rules or guidelines for proposal writing. It only depends on the professional skills and experience proposals describe the aims and objectives of the project and the manner in which it will be carried out. If the proposal writing is effective; only then it will be accepted and contract will be awarded.

2. **Project Planning:** It is just like a Guide Map which guides the development team towards the achievement of goals. Project plan keeps the following points in mind.

- Identifying different activities
- Setting of Goals
- Identifying the final result in the form of deliverables to be produced by the project.

3. **Project Scheduling:** It involves the following things

- Breaking the main project into smaller tasks and identifying activities
- Estimating resources for these activities
- Judging the time required to complete these activities
- Allocation of people to different activities and coordinating their tasks.
- Full utilization of human resources

4. **Project Cost Estimation:** This activity is concerned with guessing and calculating the resources required completing the project plan.
5. **Project Monitoring:** In this activity the project manager check the progress of the project from time to time and compares the actual and planned progress on costs. If there is any deviation then project manager takes corrective steps.
6. **Project Review:** This activity is concerned with observing the overall progress and technical development and to study whether the project's status is in accordance with the aims and objectives of the organization commissioning the software.
7. **Personnel Selection and Revaluation:** The project manager selects a balanced team of experienced employees and youthful fresher's. It is necessary that the services of the staff are utilized effectively.

For selecting personals, a project manager has to keep in mind the following steps:

- Project Budget
- Impossible to appoint new staff
- Providing Training to employees

8. **Project Report Writing:** A Project Manager prepares Project Report. He is responsible to submit this report to both the client and contractor organization. He must have the written as well as oral skills to present his report in a very effective way. The language of the project report should be very simple and clear to understand.

2.3.4 **Issues in Project Management**

There are so many issues which are to be kept in mind in the Software Project management. There are specific goals that are to be achieved. Certain Laws and Principles have to be followed. The difficulties and challenges which stand in the way of management have to be overcome.

Goals

1. To develop the software within the specified budget and within the planned resources. It should be economically justifiable.
2. Time schedule is of utmost importance. The software should be produced on time and is ensured to deliver on time.
3. To develop the software according to the parameters and requirements of the user.

Principles

The following are the principles that are to be followed:

1. Before starting the project, aims and objectives should be clearly understood.
2. The project manager should understand various constraints which may be of hardware, software, cost or any other type of constraints.
3. Making plans in advance to achieve the goals in a better way. Cost estimation, Need for human resources and other factors should be planned well.
4. The project should be monitored so that no problem goes unnoticed.
5. Adjustments, modifications and appropriate changes as and when necessary should be made for appropriate result.
6. A healthy and good work environment should be provided so that each member gives his/her best performance.

Difficulties and Key Challenges

Software project are unmatchable and developed.

1. **Redundancy Problem:** Software projects are unmatchable and developed only once. They face redundancy problem in some cases.
2. **Multiple Solutions:** It becomes very difficult to decide which way to go when there are multiple solutions for a given problem.
3. **Difficult Cost Prediction:** It is difficult to predict personnel cost because they have different capabilities.
4. Technological advancement in hardware and software technologies creates a big obstacle in the development of the software.
5. Software is very difficult to monitor due to its intangibility.
6. Changes in software specifications and design can take place during any phase of software development.
7. **Not Well Designed:** The responsibilities of different team members are not sometime well defined. The user requirements are not precisely defined.

All these difficulties need to be overcome so that the software is developed within the specified time and estimated costs and fulfills the user requirements.

2.4 **SOFTWARE CONFIGURATION MANAGEMENT PROCESS**

Introduction

Software configuration management or in short SCM is the discipline of identifying, defining and controlling changes throughout the life cycle. It includes revision control and the establishment of baselines. Sometimes it so happens that changes arise due to near business or market conditions and the atmosphere so created requires a change in the final product. New user requirements also need modifications of data or functions produced by the software. Other reasons may be business growth, downsizing, cost and time constraints. SCM manages changes in the intermediate products in a controlled manner so as to maintain the integrity of the product. The traceability is also maintained and the development of the product is made more manageable.

Software design may change at different points of time like after testing phase or coding phase. The main points or goals of SCM are as follows:

1. **Configuration Identification:** The structure of the product is identified and then its components and their types are also identified. CCM then makes these products unique and accessible in some form.
2. **Control:** Implementing a controlled change process. For this a change control board is set up which approves or rejects all change requests. It answers consistent software by the creation of a baseline product.
3. **Configuration Status Accounting**: Recording and reporting the status of the components and change requests. It collects all information and vital statistics about the status of the development process.
4. **Auditing and Reviewing:** Validating that the configurations contain all the intended parts are sound. It keeps consistency with respect to specifying documents and ensuring that the product is a well-defined collection of requirements, architectural specifications and user manuals.
5. Build management, process management and environment management all ensure adherence to the organization's development process.
6. Facilitating team work and making sure that every defect has been tracked back to the source.

Software Configuration Items

It is one of the smallest unit of change. EEE defines it as "an aggregation of hardware, software or both that is designated for configuration management and treated as a single entity in the configuration management process".

Base lines

Whenever there is a need for a change, a basis for the change must be clearly known to making changes. A baseline helps to control change. IEEE defines a baseline as a specification or product that has been formally renewed and agreed upon that thereafter serves as the basis for further development and that can be changed only through formal change control procedure."

Activities of SCM

The various activities involved in configuration management are:

- **Configuration identification**: The structure of the product is identified and then its components and their types are also identified. CCM then makes these products unique and accessible in some form.
- **Change control**: Implementing a controlled change process. For this a change control board is set up which approves or rejects all change requests. It answers consistent software by the creation of a baseline product.
- **Configuration audit:** Validating that the configurations contain all the intended parts are sound. It keeps consistency with respect to specifying documents and ensuring that the product is a well -defined collection of requirements, architectural specifications and user manuals.
- **Configuration status reporting:** All these activities ensure that all the changes are properly implemented.

Changes in Software Development

— Diagrammatic Illustration.

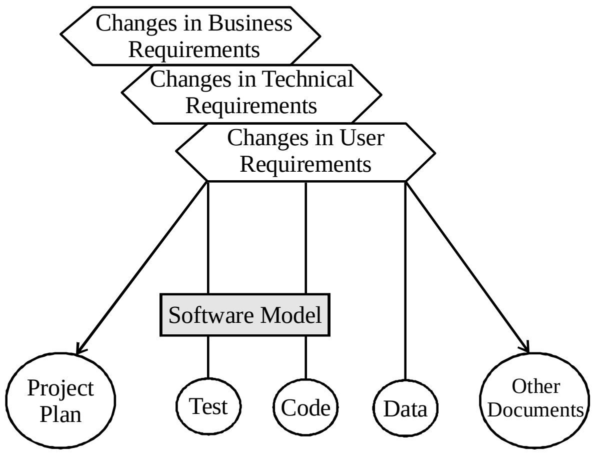

Changes in Business Requirements
Changes in Technical Requirements
Changes in User Requirements
Software Model
Project Plan
Test
Code
Data
Other Documents

Fig. 2.4: Diagrammatic Illustration of Changes in Software Development

BACHELOR OF COMPUTER APPLICATIONS (BCA)
BCA-4-01T: SOFTWARE ENGINEERING

UNIT 3: PROJECT PLANNING

3.1 **SOFTWARE PROJECT PLANNING**

Project Planning is the most important aspect of software project management. Project planning prepares a list of activities that need to be done to achieve the goal. The main purpose of project planning is to clarify the goals, needs and constraints. The success of a project depends upon its planning because poor planning is a sure ticket to failure and poor quality result with high maintenance cost. A good planning prepares a road map in advance what to do, how to do, when to do, who is to do and how much will it cost to complete the project.

3.1.1 **Main Objectives**

Project Planning is a well-organized and an integrated process which keeps in mind the ultimate goal to achieve:

- Project Planning aims at better utilization of resources.
- To finish the project in time.
- Defines roles of different team members.
- Assigns duties and responsibilities.
- It determines project constraints.

A large number of people are involved in the project planning. These include

(i) Senior Management which is responsible for employing personnel's and providing resources.

(ii) The project management team which includes project managers and developers. They are responsible for planning determining and tracking activities, selecting all ways and means to achieve the goal. The project manager is the key to successful project planning.

3.1.2 Principles of Project Planning

The following principles or fundamentals of software project planning are to be followed:

- **Planning is the basic need:** The software project management process begins with a group of activities called Project Planning. This is one of the most important management activity. It is necessary for modern business because helps in better co-ordination and reduces/removes uncertain. Planning is a never-ending process.
- **Risk Analysis:** It is the duty of the Senior Management and Project Management to study all the risks and hindrances that may team affect the final output.
- **Tracking of Project Plan:** The project plan should be continuously tracked and modified as per the requirements.
- **Quality Products:**Those process should be identified which can ensure quality in software.
- **Description of Flexibility:** Planning should be flexible so as to accommodate new changes the project is in progress.

3.1.3 Components of Project Planning

Different types of projects can have different components depending upon their working and implementation. Here are some common and powerful components of project plan.

1. **Project Scope:** Each project has its own wide project scope.
2. **Project Schedule:** Every project has its own time frame within which it must be completed.
3. **Project team organization:** Each project has a well-organized team under a project leader.
4. **Technical Description:** Each project has a well-organized team under a project leader.
5. **Project Standards, Procedures and proposed technical tools**: For success, completion of the project, it must have all the standards, process and technical tools.
6. **Quality Assurance Plan**: Each project has the assured plan for successful completion of the project.
7. Special Development Tools and Techniques are needed.
8. **Configuration Management Plan:** For the long range planned projects.
9. **Documentation Plan:** A project must have the documentation which is well-planned for the maintenance of the project at the later stage.
10. **Data Management Plan:** There should be Data Management Plan for every working project in the short range planning.
11. **Resource Management Plan:** There must be a Resource Management Plan to implement the project in a successful manners.
12. **Test Plan:** A detailed and ordered test plan is needed for the successful testing of two project.
13. **Training Plan:** It must be set for the users for the successful completion of the project and working of the project.

14. **Security Plan:** There should not be any un-authorized access. For this security plans like the secret purpose password should be established.
15. **Risk Management Plan:** A Project Manager must consider all the constraints and possibilities to avoid and type of risk management.
16. **Maintenance Plan:** Maintenance procedures and plans are needed to avoid any type of threat during working hours.

Planning is done on both formal and informal basis. The main aim is to provide the guidelines for decision making are based on future expectations.

3.1.4 Activities in Project Planning

Project Planning has various aspects like estimating, scheduling and assigning the project resources in such a way to deliver the find product of desired suitable quality. However activities have an important role which can determine the success of the project. The various planning activities team members will need to do are given as under:

- The recruitment of the team and building of the team as a unit.
- To organize the project
- To know and confirm the start and end dates by preparing a project schedule.
- To create the project budget.
- To know about the customer requirements for the final result.
- Limiting the project scope boundaries and defining them properly.
- To write a description of the find result.
- To assign duties
- To assign accountability.

A typical project plan can include the following activities which have been briefly discussed as below:

1. **Project Scope Definition and Scope Planning:** The assumptions, constraints and user expectations along with business needs, technical needs and project deliverables etc are documented. Project objectives and any other thing that defines the final product requirements is also documented.

2. **Quality Planning:** This is an important part of project planning. Quality of the final product can’t be over looked. For this various factors that influence the quality of the final product are taken its account.

3. **Project Activity:** Definition and activity sequencing: All activities are defined in advance which are required to deliver the product by producing the various product deliverables.

4. **Time, Effort and Estimation of Resources:** Effort, time and resources required are clearly estimated and documented in this step. Many techniques like function points, lives of code, complexity of code, Bench mark etc. can be used to calculate effort.

5. **Risk Factor Identification:** One should be ready to face the unexpected things in life. It is very important to pin point the risk factor based on the assumptions, constraints, user expectations and challenging conditions.

6. **Schedule Development:** Project Scheduling is one of the main task of project planning. Time frame work is to be prepared for each of the inter-dependent activities. It will have an influence on the cost estimates and the cost benefit analysis.

7. **Risk Management Planning:** In this process risk factors are identified and analyzed. A risk resolution plan is prepared to minimize the impact of the risk factor on the project.

8. **Cost Estimation and Budgeting:** The costs involved in the execution and implement of the project is estimated and only then budget allocation is done for different phases of the project.

9. **Organizational & Resource Planning:** The main objective of resource planning is the smooth and efficient running of the project. There are many types of resources like equipment, personnel, facilities, money etc. These resources will have to be fully utilized to achieve the goal.

10. **Project Plan Development & Execution:** The information gathered from all planning processes is used and each of the project tasks and activities are monitored and controlled from time to time delays are analyzed and the project plan is adjusted accordingly.

11. **Performance Reporting:** The progress of each of the tasks is compared with the schedule. Various Techniques such as EVM (Earned Value Management) can be used to measure the performance. Tools like PERT Charts, GANTT Charts, Logical bar charts, Histograms, Pie charts etc. can be used to report the performance.

12. **Planning Change Management:** If the need arises then certain aspects of the project can be changed, but the changes should not have any negative effect on the environmental or the performance. Changes should be studied properly and only then the project plan may be modified.

13. **Project Rollout Planning:** The success of the project depends on the project rollout and implementations. Where a project is rolled out; it has greater impact on the entire technical systems and the business systems. The users should be read to accept it and use it effectively. The users need to be trained to adapt to the new system.

3.2 **SOFTWARE PROJECT MANAGEMENT PLANNING**

This is an important part of project management. The following points are kept in mind by the project management.

(a) What objectives are to be achieved?

(b) What resources are needed?

(c) How and when the resources are to be acquired?

(d) Ways and means to achieve the goal.

(e) To determine the flow of information, people and products within the system.

Organizing

It involves the division of duties and responsibilities among different groups in such a way that the goals of the enterprises are achieved. This is possible only when proper planning is done by the professional within the groups. Best organization means best results in the end.

Staffing

It deals with hiring of qualified and well trained personnel for the positions that are recognized by the organizational structure. It carries the process of recruiting, hiring, training, rewarding, retaining, compensating, developing and promoting employees.

Directing

It is the process guiding and leaving the subordinates so that they may understand and identify the organizational structure and the goals to be achieved by the enterprise.

Controlling

This step includes the process of measuring and correcting activities. The performance of each group is measured against plans and if there is any deviation, then corrective actions are taken to achieve the desired goal.

Software Project Management is based on sound technical skills and Management Skills. The Technical Process of software development depends on the application of sound management principles. It is not possible for one person to handle todays large and complex programs therefore team work is necessary under the leadership of the project manager.

3.3 **SOFTWARE COST ESTIMATION**

Cost estimation is the most important task of a project. An accurate and scientific estimation of cost, effective usage of resources and time schedule is the basis for a successful completion of a project. Cost over-run may enrage. The customer and it can lead to the cancellation of the project. There are some factors such as complexity of the project, time availability, reliability and software size which mean a lot of cost estimation. It can be done during any phase of software development. It depends upon software information regarding user requirements, design and source code etc. It becomes easier to estimate cost in the later stages as more information becomes available. To avoid unforeseen delays and risks; cost estimation should be done throughout the life cycle of the project. The following diagram accurately depicts how the cost estimation becomes more and more accurate as additional software information is pouring in.

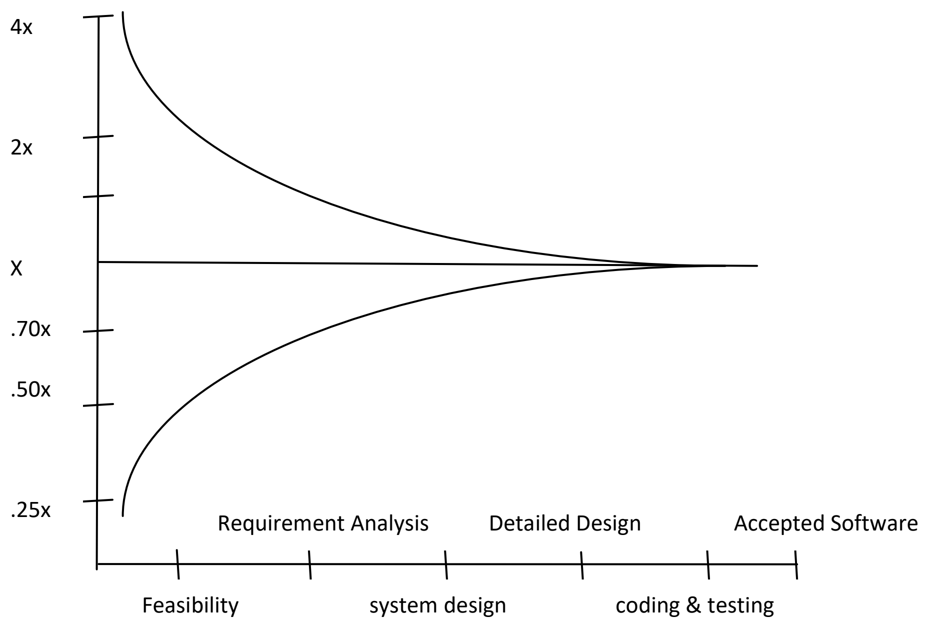


Fig. 3.1: Accuracy of Cost Estimation

The diagram clearly shows that as we proceed from requirement phase to coding and testing phase via system design phase; cost estimates becomes more accurate.

Cost Factors

The following factors influence the cost estimation:

1. **Experience:** If the software developer is well trained in programming language, operating system and hardware the cost will be less.
2. **Product Complexity:** The cost of software project rises up with the level of complexity. Three levels of product complexity are organic, semi-detached and embedded programs.
3. **Project Size:** A large sized project is more costly as compared to a small sized project. More effort is required for a large sized project.
4. **Available Time:** If development time is decreased, then the software project requires more resources and effort and thus the cost increases.
5. **Programmer Ability:** Efficient and capable software programmers bring down the expenditure whereas inefficient programmer increases the cost. Therefore, it is necessary that a programmer should be well-trained.
6. **Level of Technological Know:** How also has a great influence on cost estimation? Modern practices, System analysis, Design techniques etc. have a great effect on cost estimation.

7. **Reliability:** Reliability of the software project depends on the level of accuracy, robustness and consistency etc. cost estimates depend on the level of analysis, design, implementation and verification effort.

The cost estimation should be well-planned. It should be reviewed at regular intervals. The software process should be continually updated and improved upon.

3.4 SOFTWARE SIZING

Sometimes a problem is too big to the understood in a single attempt, similarly software cost estimation is not simple as it appears to be a complex problem like software cost estimation is broken up into fragments to achieve an accurate cost estimate.

Mainly two approaches are adopted:

(i) Problem based estimation
(ii) Process based estimation

A project planner must establish the estimate of software size which is considered to be the Quantitative result of the software project. The software is divided into smaller parts because it is convenient to calculate and know about the size of the smaller components. They are less complex. The addition of these smaller components gives an idea of the overall estimate of software size.

Sizing Approaches: There are two types of approaches for estimating size:

1. **The Direct Approach:** Measures sizes in terms of (LOC) lines of code.
2. **The Indirect Approach:** Measures in terms of functional point (FP).

Parameters: The following parameters influence the accuracy of size estimates:

1. The degree of properly estimating the size of the software.
2. The capability of converting Size Estimate to human effort, calendar time and money.
3. The ability of the software team.
4. The stability of product Requirements and the overall environment supporting the development process.

3.4.1 **Approaches/Types of Sizing**

The project planner finds that, the estimation of size is very important in determining the cost estimation. This problem is also known as Software Sizing. To solve this problem, the following methods can be used:

1. **Fuzzy Logic Sizing:** The planner identifies the application type and its magnitude on a quantitative scale. The magnitude is then refined within the original range.

2. **Function Point Sizing:** In this method the functionality produced by the software system is measured the function point estimates.

3. **Standard Component Sizing:** Standard components can be in the form of modules, screens, reports, lines of code and soon. The number of time a component occurs is estimated and the historical data helps to find out the delivered size of each standard component.

4. **Change Sizing:** Through this method, already existing project is modified and the number and type of the modifications are estimated.

3.4.1.1 **Lines of Code**

It can be defined as the number of delivered lines of code. Lines of code are a measure to calculate productivity metrics. LOC is used as follows:

(i) As an estimate variable to size each and every element of the software.
(ii) It is used as baseline metric obtained from the previous projects and used with estimation variables to develop projections for cost and effort.

Line of code is the most commonly used software size metrics. LOC depends on the programming language. In assembly language lines of code will be higher as compared to lines of code in high level language like C++/Java.

In the earlier stages of development, less information becomes available where as full information becomes available only at the end of the project. So exact number of lines of code can be determined after the completion of the project.

Guidelines

1. One line of code is for One Logical line of code.
2. Lines of code delivered as a part of software are included.
3. Lines of code for drivers, test stubs and other support software are excluded.
4. Software code by software developer is included whereas the code created by the application generators are excluded.
5. Declarations in the programme are counted as LOC whereas comments in the programme are not counted as LOC.

Keeping in view the historical Data, The project planner estimates. Three values optimistic S_{opt}, most likely (S_m) and pessimic (S_{pess}) for each size.

The equation ($S_{opt} + 4 \times S_m + S_{pess}$) is used to compute expected value. The following example of a software will clarify how the estimating size is determines. Here in the example software is a combination of six functions — (i) user face, (ii) word processing, (iii) file storage and retrievals, (iv) Data base management, (v) Word processor and (vi) Peripheral control.

Estimating Size

Function	S_{pess}	S_m	S_{opt}	**Expected Size** $S = \frac{1}{6}(S_{opt} + 4 \times S_m + S_{pess})$
User interface	1500	1800	2100	1800
Word Processing	1800	2400	3000	2400
File Storage & Retrievals	1700	2100	2400	2083
Data base management	2500	3000	4000	3083
Word Processor	1200	1600	2000	1600
Peripheral Control	1300	1700	2000	1683
Total Expected Lines of Code =				**12649**

Table showing method of calculating Size Estimation

The developer estimates the size of each and every function:
S for user interface =
S for word processing = and soon
We see that size of the software in terms of
LOC is = 12649

3.4.1.2 **Function Point**

Function Point lines of code are the most important methods for size estimation. The function point metric was devised and suggested by A.J. Albrecht. This is used to measure functionality delivered by the system; Function point estimates in determining estimate effort required to design, code and test software, predict the number of errors and it so forecasts the number of components being used in the system.

The measures of software information domain value and software complexity are the two things needed to find out function point through an empirical relationship.

Software complexity is of three categories; of simple, average and complex level.

Information domain value is a combination of the following points.

1. **Number of external inputs or (EI):** Users and other applications are a source of providing application oriented data.

2. **Number of external outputs or (EO):** Theserefer to Reports, Screens, Error, messages etc. The user gets information from each external output. Individual data views are not counted separately.

3. **Number of external inquires (EQ):** Online inputs are referred to as external inquiries. These in turn. Generate responses in the form of online outputs. In this case each and every district inquiry matters and is counted separately.

4. **Number of internal logical files/(ILF):** Logical grouping of data present in the application boundary e.g. master file as part of database is known as internal logical file.

5. **Number of external interface files (EIF):** Logical grouping of data which is external to the application e.g. Data files on tape or disk is known as external interface file.

The complexity value for each count is determined after gathering information about information domain value function point method helps in deter mining about a particular entry whether if is simple, average or complex.

Estimating Functional Points: Functional points in software are estimated by the equation as given below.

F.P = Count Total (o.65 + 0.01 × Σ (f1)]

Count Total = Sum of all F.P. entries

fi = (i = 1 to 14) are value adjustment factors.

The value adjustment factors are based on the answers to 14 Questions. These questions are answered using a scale of 0 to 5 where

0- means – No influence
1- means – incidental
2- means – Moderate
3- means – Average
4- Significant
5- Essential

Weighting Factor Table 1

Measurement Parameter	**Count**		**Simple**	**Average**	**Complex**		**F.D. Count**
Number of User inputs	21	×	3	5	7	21×5=	105
Number of user output	15	×	2	4	6	15×4=	60
Number of user inquiries	25	×	4	5	6	25×5	125
Number of files	7	×	8	10	14	7×10	70
Number of external interfaces	3	×	5	7	10	3×7	21
Count Total							**381**

Value Adjustment Factors Table II

Sr. No.	**Factors**	**Disadvantages**
1.	Backup and Recovery	2
2.	Data Communications	2
3.	Distributes Processing	1
4.	Performance Critical	3
5.	Existing operating environment	2
6.	Online data entry	3
7.	Input transactions over multiple screens	4
8.	Master files updated online	3
9.	Information domain value complex	4
10.	Internal processing complex	2
11.	Code design for reuse	3
12.	Conversions/installation in design	3
13.	Multiple installations	4
14.	Application design for change	4
	Value adjustment factor	40

Computations for Function point: To estimate size in terms of function point table I and Table II are used as follows

FP count = Count X weighting factor (AV)

From Table I FP for number of used inputs = 22×4=88

using Table I and II F.P. count Total × [0.65+0.01×?fi]

= 381× [0.65+0.01×40]

= 381× [0.65+0.40]

= 381× 1.05

= 400.05

= 400

3.4.1.3 **"Fuzzy Logic"**

A project estimate is only as good as the estimate of the size of the work to be accomplished. Different approaches to software sizing are given as under:

1. Lines of codes
2. Functions point
3. Fuzzy logic
4. Change sizing

Fuzzy Logic sizing is a type of approach in which the project planner must identify the application type and its magnitude on a quantitative scale. After this the magnitude is refined which in the original range.

"Fuzzy Logic" approach applies the approximate reasoning techniques that are the corner stone of fuzzy logic. The personal experience can find a place but the planner should also keep on eye on the historical database of the projects so that estimates can be compared to original/actual experience.

3.4.1.4 **Change Sizing**

Sizing signifies the project planner's first major challenge. Sizing steps are essential to make valid comparison cross systems or within systems. Productivity can be computed with the help of a software sizing measure.

The change sizing approach is followed only when there arises a need for changes in the already existing project specifications. This modification is used in the new project to bring a required change in its accomplishment. The planned estimates, the number and type (e.g. reuse, adding code, changing code, deleting code) of modifications that must be finalised. By the application of an "Effort Ratio" for each type of change, the size of the change may be guessed and this new estimate will help to produce the software according to the user requirement.

3.5 **PROBLEM BASED ESTIMATION**

LOC and F.P. data are used in two ways:

- As an estimation variable to 'size' each element
- As baseline metrics gathered from past projects to develop cost and effort projections.

Although LOC and F.P are clearly different techniques but they have many common characteristics. After a bounded statement about software scope, the project planner decomposes the software into problem functions so that they may be estimated individually. LOC or FP is then estimated for each function.

Using baseline productivity metrics, proper estimation variable and cost or effort for the function is calculated.

LOC and FP estimation are different so far as their technique of decomposition and the target of the partitioning is concerned when LOC is used the decomposition approach supposes that all functions can be broken up into sub functions. LOC estimates depend on the degree of partitioning. For FP estimates decomposition works in a different manner. Instead of focusing on function, it estimates on the basis of information domain characteristics i.e. inputs, outputs, data files, inquiries and external inferences. In addition to 14 complexity adjustment values are estimated and a FP value is derived that can be linked to past data and this is used to generate an estimate.

3.6 **PROCESS BASED ESTIMATION**

In order to achieve objectives in software project developed a specified and certain process is followed. The estimation of effort in a software is to base the estimate on the process to be followed. The whole process is divided into smaller tasks and after the identification of these tasks; the effort for each task is estimated. These smaller tasks are such as analysis, design and coding etc. These tasks are ordered according to the specified sequence. The project managers take the product size as a major productivity indicator and they become capable of tracking a project. A number of activities are performed in process based estimation. These activities can be listed as follows:

1. Customer communication
2. Planning
3. Risk Analysis
4. Engineering (analysis/Design)
5. Construction release and so on

It is the duty of the project planner to combine the functions and activities to calculate the effort required for each function. The labour rates which vary from task to task are applied. Top level management activities are more expensive on compared to activities performed by the junior staff in the beginning of the framework activities. The accuracy in process based estimation depends on many parameters.

- The degree of sizing of the software and the proper estimation.
- Changing size estimate into human effort, calendar time and money.
- Ability of a software team
- The stability of product requirements
- Environment supporting the development process.

The software considered for process based estimation is divided into functions and for each function a set of tasks and activities are taken last of all, the effort for each function and activity is calculated.

Activity	Customer Communication	Planning	Risks Analysis	Engineering		Construction Release		CE	Total
Task				Analysis	Design	Code	Test		
Function 1				0.50	2.00	0.40	5.00	n/a	7.90
Function 2				0.70	4.00	0.60	2.50	n/a	7.80
Function 3				0.50	3.00	1.00	3.50	n/a	8.00
Function 4				0.50	3.00	0.75	2.50	n/a	6.75
Function 5				0.50	2.00	1.00	2.50	n/a	6.00
Function 6				0.25	1.00	0.50	2.50	n/a	4.25
Function 7				0.50	1.00	0.50	1.00	n/a	3.00
Total	0.25	0.25	0.25	3.45	16.00	4.75	19.50	-	43.70
%Effort	1%	1%	1%	7.66% = 8%	35%	11%	43%	-	

Table showing percentage effort for each function & Activity

3.7 **COCOMO-MODEL (CONSTRUCTIVE COST ESTIMATION MODEL)**

Barry Boehm gave birth to the idea of COCOMO Model in early 19805 to estimate total effort required to develop a software project. This Model is widely used and if is based on the observation of previously developed software projects. According to Boehm any software development project can be categorized into three type's organic, semidetached and embedded depending on the development complexity.

Organic projects: These projects are in an area in which the organization has enough experience. These project are small in size (not more, the 50 KOLOC). Such system has small teams with prior experience. The term members work together with through understanding and co-operate to accomplish user requirements which are less stringent examples: Simple business system, simple inventory management system etc.

Semi-detached: In such a project, the development team consists of a mixture of experienced and inexperienced staff. The size of such projects is not more than 300 KDLOC

Examples: Operating system, compiler design and data base design

Embedded Projects: These projects are ambitions and novel, they have size more than 300 KDLOC. The organization has little experience developers have to achieve stringent user requirements and these systems have tight constraints from the environment (hardware, software and people)

Example: Software systems used in avionics and military hardware, real-time command system.

COCOMO model estimates the total effort in terms of person-months of the technical project staff. the efforts estimate includes development, management and support tasks but does not include the cost of the secretarial and other staff. Our person month means the effort an individual can typically put in a month. The basic steps in this model are:

1. Obtain an initial estimate of the development effort from the estimate of thousands of delivered lines of source code. (KOLOC)
2. Determine a set of 15 multiplying factors from the different characteristics of the project.
3. Multiply the initial estimate (E_i) with all the multiplying factors.

The initial estimate E_i in person\-months is determined by the equation E_i = a* (KDLOC) b. The value of the constants a and b depend on the type of the project. KDLOC is used as the measure of the size. COCOMO is based on the hierarchy of three models: basic model, intermediate model and advance model.

Basis Model: In this model only the size of the project is taken in given an approximate of the project parameters.

The Expression for calculating effort is as follows

Effort = a * $(KLOC)^b$ PM

Basic Model			
Project Type	Constant (a)	Constant ‘b’	E
Organic	2.4	1.05	2.4× (30) 1.05 for 30 KDLOC = 85 pm
Semi-detached	3.0	1.12	3.0× (30)1.12 for 30 KDLOC
Embedded	3.6	1.20	$3.6\times(30)^{1.20}$ for 30 KDLOC

Effort Calculation

By using logarithmic tables; effort can be found out. Estimation of development time (Tdev): The formulator estimating the development time based on the effort are as follows:

Organic : T_{dev} = 2.5 (effort) 0.38 months
Semidetached: T_{dev} = 2.5 $(\text{effort})^{0.35}$ months
Embedded : T_{dev} = 2.5 $(\text{effort})^{0.32}$ months

e.g. if an organic type software product is estimated to 32000 lines of source code and a software engineer gets Rs. 6000 as salary par month; then nominal effort, nominal development time and cost required to develop the project an be calculated as

Effort : $2.4\ (3.2)^{1.05} = 91$ PM

T_{dev} : $2.5\ (91)^{0.38} = 14$ months

Cost: 14× 16000 = 224000

Intermediate COCOMO Model

This model takes other relevant parameters into account besides development time and effort the intermediate COCOMO model refines the initial estimate (E_i) by using a set of 15 multipliers based on different characteristics of the software development to calculate total effort in this model, the procedure is as follows:

1. Calculate (E_i) considering the size in terms of KDLOC.
2. Identify 15 parameters and each parameter is given a numerical value called multiplying factor.
3. Effort adjustment factor (EAF) is derived by multiplying all multiplying factors with each other.
4. Multiply initial estimate in step 1 with EAF.

15 Different attributes called cost driver attributes determine the multiplying factors which depend on product, computer, personnel and project attributes are shown in the table below:

Cost Drivers		**Rating**				
		Very Low	**Low**	**Normal**	**High**	**Very High**
	Product Attributes:					
1	RELY = Required Reliability	0.75	0.88	1.00	1.11	1.40
2	DATA = Database size	0.94	1.00	1.08	1.16	-
3	CPLX = Product Complexity	0.70	0.85	1.00	1.15	1.30
	Computer Attributes:					
4	TIME = Execution Time Constraint	-	-	1.00	1.11	1.30
5	STOR = Main Storage Constraint	-	-	1.00	1.06	1.21
6	VIRT = Virtual Machine Volatility	-	0.87	1.00	1.15	1.30
7	TURN = Computer Turnaround Time	-	0.87	1.00	1.07	1.15
	Personnel Attributes:					
8	ACAP: Analyst Capability	1.46	1.19	1.00	0.86	0.71
9	AEXP: Applications Experience	1.29	1.13	1.00	0.91	0.82
10	PCAP: Programmer Capability	1.42	1.17	1.00	0.86	0.70
11	VEXP: Virtual Machine Experience	1.21	1.10	1.00	0.90	-
12	LEXP: Language Experience	1.14	1.07	1.00	0.95	-
	Project Attributes:					
13	MODP: Modern Programming Practices	1.24	1.10	1.00	0.91	0.82
14	TOOL: Software Tools	1.24	1.10	1.00	0.91	0.83
15	SCHED: Development Schedule	1.23	1.08	1.00	1.04	1.10

Determination of multiplying factor for different Attribute

Advance Model: In this model, effort is calculated as a function of programme size and a number of cost drivers for each phase. This model covers the limitations of both basic and the intermediate model. Both there models consider a software product as a single homogeneous entity. But most large systems are made up of several smaller sub systems. These sub systems have widely different attributes. Some may be organic type, some semidetached type of some embedded. Complexity of subsystems may be different.

There are four phases in the advance COCOMO model. They are (1) Requirements Planning and Product Design (RPD) (2) detailed design (DD), (3) Code and Units Test (CUT) (4) Integration and Test (IT)

Each cost driver is rated as very low, low, nominal, high and very high. Multiplying factors are assigned for Analyst capability cost driver (ACAP) for each phase of advanced model are tabulated below:

Rating	**RPD**	**DD**	**CUT**	**IT**
Very Low	1.80	1.35	1.35	1.50
Low	0.85	0.85	0.85	1.20
Nominal	1.00	1.00	1.00	1.00
High	0.75	0.90	0.90	0.85
Very High	0.55	0.75	0.75	0.70

Rating of Cost drivers

3.8 **THE SOFTWARE EQUATION**

The software equation is a dynamic multi variable model that assumes distribution of efforts over the useful life of the project. The software equation can be derived from data obtained by studying several existing projects and it been done by collecting over 400 contemporary software projects. For the calculation of effort, the following equation has been framed:

$$E = \left[\underset{\underset{\text{LOC}}{\Downarrow}}{\text{Size}} \times B^{\mathbf{0.333}} / P \right]^{\mathbf{3}} \times \left(1/t^4\right) \text{.......equation}$$

Where

P = Productivity parameter which indicates the maturing of overall process and management practices. This parameter also indicates:

(a) The level of programming languages used.
(b) The state of software environment.
(c) The skill and experience of the software team.
(d) Complexity of software application.

T	=	it indicates project duration in months or years
E	=	Efforts in person-months or person-years.
B	=	Special skills factor — the value of B increases over a period of time depending upon the importance and need for integration, testing, quality assurance, documentation and management.

Software equation has to important independent indicators they are:

I. An estimate of size (in LOC)
II. Time duration in months or years.

Potnam and myers suggested another set of equation derived from the software equation for simplification of estimation process.

t_{min} = 8.14 [LOC/P]$^{0.43}$ in months for t_{min}> 6 months equation-II (a)

E = 180 Bt^3 in person – months for E $\geq$ 20 person months eqn-II (b)

____For small programs with sizes between 5KD LOC and 15KD LOC, the value of B is 0.16 and for programme with sizes greater than 70KD LOC; the value of B is 0.39

3.9 **PROJECT SCHEDULEING**

Definition

Project Scheduling can be defined as the process of effectively managing the tasks of the project by identifying and organizing the tasks into a series of events. Project scheduling prepares details such as (i) start dated of the project, (ii) end date of the project, (iii) milestones, (iv) specifying the resources required and (v) tasks for the project. In ensures a harmonious completion of the project on time. It also helps to avoid the extra cost incurred when the project gets delayed. Project scheduling tries to predict the future. Although it is not possible to say with certainly about the length of time yet there are methods which can increase likelihood of being close.

Principles of Project Scheduling

The most commonly adopted principles for project scheduling are as follows:

(i) **Compartmentalization:** The project is divided into several tasks to manage the project effectively.
(ii) **Interdependency:** All the activities of the project are inter-related and not independent of each other. Various activities are performed in a sequential manner. Some activities depend so much on other activities that they cannot begin until the activity on which they depend is completed.
(iii) **Time allocation:** Another principle of scheduling is time allocation. It determines the time for performing specified activities. It is also very important to estimate the effort required. The project management should assign final time to each team member.

(iv) **Effort Validation:** The project manager assigns tasks to the team members according to their capabilities and efforts required for the task.

(v) **Defined Duties and responsibilities:** The roles and responsibilities of each project team member should be decided in advance depending on their skills and capabilities.

(vi) **Defined Outcomes:** According to this principle, the outcomes of every task should be specified. A product is outcome of a task and these products are well combined in deliverables.

(vii) **Defined Milestones:** Mile stones be specified when task products are complete and it should be renewed for quality.

Delaying Factors

There are various factors which are responsible for delay in the project schedule. The most common factors are:

- **Unrealistic Deadlines:** Sometimes the allocation of time for completing a project is not practical because inexperienced individuals establish a wrong deadline. The project gets delayed if deadline is not achieved.
- **Change in User Requirements:** There is delay if user requirements are changed after the project has already started. This disturbs the project schedule and thus more time is taken.
- **Under Estimation of Resources:** If there is under-estimation of resources, it leads to delay in the completion of tasks of the projects.
- **Lack of Analyzing Risk Factors:** If risk factors are ignored during project planning and scheduling; these risk factors put on a bad effect on the software development.
- **No Communication among Team Members:** Lack of communication among team members makes it difficult for the project management to complete the project on time. Personal problems of team members also stand in the way of the completion of the project.
- **Lack of Action on the part of:** If the management fails to take timely action and does not understand that the project takes more time to complete; then it also becomes the reason of delay.

Activities in Project Scheduling

Various activities are involved in project scheduling. There are to be observed sincerely. The experience in the project area and in the scheduling. Those persons are interviewed who had experience with similar projects. Transition between activities is taken into account uncertain resources of talent, equipment or data are also studied so involved in project scheduling is as follow:

1. Gant Technique
2. Mile stone Scheduling
3. Pert and CPM

The Gant technique is used where production operations are highly repetitive and work performance of different departments can be combined.

In the mile stone scheduling system the milestones are developed and established in the planning phase.

PERT/Networking and CPM method are designed to be used in the development phases for identifying the critical path, float and slack.

PERT Chart (Program Evaluation and Review Technique Chart)

A PERT chart is a graphical chart meant to determine the activities that from the “Critical Path” which if delayed will harm the overall project and the project will be delayed. The use of PERT chart is justifies in large projects in which activities can be divided into two categories (i) Critical activities, (ii) Non-critical activities. The critical activities should not be delayed.

A PERT chart is a method of project management which is used to schedule, organize, and coordinate different tasks within a project. If analyses the involved tasks and determines the time needed to complete each task and thereby identifies the minimum time needed to complete the total project in time.

The PERT chart methodology was first used by U.S. Navy in 1950s to manage the Polaris Missile Project.

Advantages of PERT Chart

1. It illustrates the project in a graphic form.
2. It guesses expected time required to complete a task.
3. It provides information regarding the time expected to complete the project.
4. It describes if there is the probability of completion of the project before the date specified for it.
5. It specifies the activities forming the critical path.
6. It provides information about start and end dates of different activities.
7. It describes the inter-dependence of one or more tasks on each other.

Steps for Creating a PERT Chart

1. Identification of different activities and milestones.
2. Recognizing sequence of Activities: Different activities are inter dependent and inter-related. It is decided whether the activities are serial wise or concurrent and then relationship among activities is depicts.
3. Preparation of PERT chart.
4. Estimation of Time: An activity can be shorter or longer depending upon the time required completing it. The time estimates can be months, weeks or days hence three categories arise.

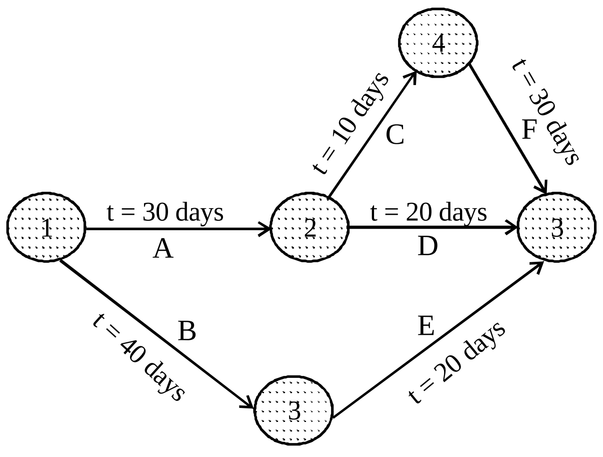
Optimistic Time: It is the shortest time needed by an activity to be completed.

Most Likely Time: It has the highest probability for the completion time required.

Pessimistic time: It is the longest time taken by an activity.

5. Critical Path Determination: Critical path for each activity is specified which determines the time required as per the project schedule.
6. Updating of PERT Chart: If any need arises; PERT chart is modified to avoid delay or when additional resources are to be provided.

Illusion of PERT Chart



PERT Chart

Fig. 3.2: PERT Chart

Milestones

In figure 1, 2, 3, 4, 5 represent the mile stones (represented by circles). The can also be represented by rectangles. When a mile stone is completed, It gets one number higher than the previous milestone. Each milestone is linked to other milestone by one or more arrows.

Activities

English letters A, B, C, D, E, F represent different activities. The direction of arrows shows the sequence of activities. There are two types of activities i.e. Serial activities and concurrent or parallel activities.

(a) (i) Activities A, C,F are serial activities taken in the same sequence. Similarly Activities B and E are also in the same sequence.

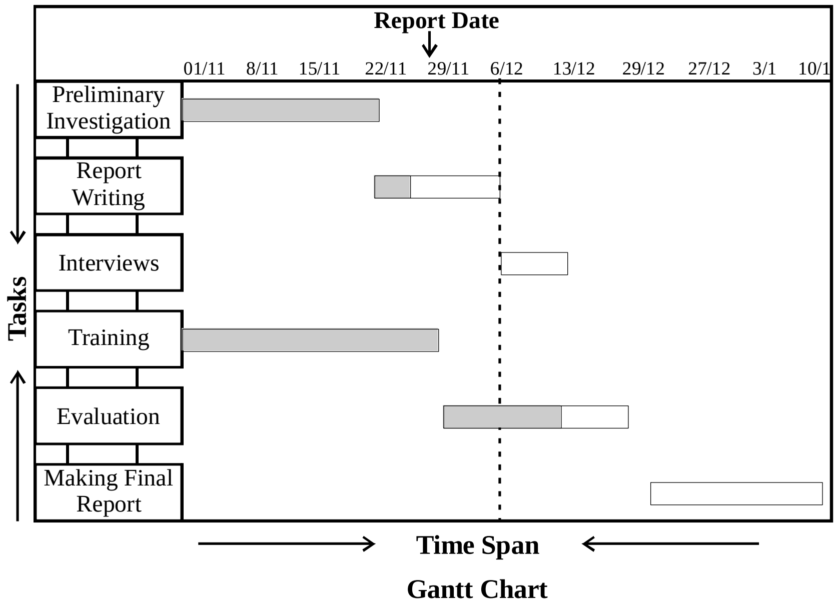
(ii) Activities A and B are concurrent activities. Similarly C and D activities are concurrent activities as these are performed simultaneously.

(b) Time for each activity is depicted by 'L' in days. It can be depicted in weeks or months also.

Gantt Chart

There are various techniques of project scheduling, task method and tracking the schedule which keep an eye on the activities that are completed as per the project schedule. These techniques provide information about activities in a graphical form – through bar charts or histograms etc. It becomes easy for the management activity. The commonly followed techniques include Gantt Chart, PERT Chart etc.

Gantt chart is in the form of a graphic diagram. This chart prepares a picture of the activities of the project. The horizontal bars in the diagram show the total time span required. This time span is divided into months, weeks and days. The time taken to complete an activity is shown in parts. The vertical column shows the activities involved. The graphical chart also shows the start and end dates of each activity. This is the reason that Gantt chart is also known as timeline chart.



Gantt Chart

Fig. 3.3: Gantt Chart

A Gantt chart is commonly prepared on a graph paper. If the Gantt chart is big and complex then other applications like Microsoft fixed etc. A project manager comes to know about the status of the activities. Gantt chart has many advantages as given below:

1. Representation of the project in a graphical form

2. It shows the progress of each activity.
3. It provides a record of the different tasks facilities.
4. It depicts milestones after each activity is completed.
5. It clarifies the tasks which are assigned to project management team members.

There can be a change in the schedule of two projects. Hence Gantt chart also varies accordingly. The diagrammatic representation given above shows the schedule if a project:

1. The horizontal bars show the total time span in the form of months, weeks and days.
2. The time for each activity is shown in increments.
3. The vertical axis shows different tasks like preliminary members, Training and so on.
4. Shaded parts show the part of the activities that has been completed.
5. Unshaded parts show uncompleted part of the activity.
6. Horizontal bars have different lengths showing the time required to complete a specific activity.
7. Time span of one or more activities can overlap each other.
8. With the progress in the project the unshaded bars are shaded showing the completion of activities.
9. The vertical line represents the report date on which an arrow is drawn.

3.10 **BASICS OF SOFTWARE MEASUREMENTS**

Measurements in everyday life has a significant role to play. Without accurate measurements; we shall not be able to take right and timely decisions in area of production and manufacturing. Software engineering uses the concept of measurement in a big way to estimate costs, to monitor inventories and asses quality. This is desirable to understand and improve the software process by predicting, planning and controlling the software projects measurements are in no way an conditional or non-value added task but it has evolved into a very significant discipline in software engineering. This is the reason that most of the organizations are using measurements to ensure better quality products. Software engineers use measurements to gain insight into the design and development of the work products.

Measure, Measurements, Metrics and Indicators are the terms which are used in measuring various aspects of software. Although they have resemblances but they are different.

Measure: It means to mention some aspect of the object in a quantitative way. It can be defined as the quantitative indication of amount, dimension, capacity or size of the products and process and their attributes. It is established. When a number of errors are detected in a single review.

Measurement: The method of determining a measure a known as measurement. It is the process of assigning number or symbols are assigned to different attributes. It is established when different components are reviewed and each unit is tested to collect the measure of a number of errors in all these components. Clearly defined rules are followed.

Metrics: A metric is used to compare two or more measures. It is a quantitative measure of the particular characteristic of a program’s efficiency. It also assesses the level of complexity, strength of the module and estimates time and cost of the project.

Indicator: This term is used to denote a representation of metric or a number of series of metrics that provides information about the software development project in process. The indicators enable the project manager to adjust the process, the project or the product for better results. It draws a person’s attention to a particular problem, tracks potential risks and evaluates the project team’s ability to control the quality of the products.

3.11 **TYPES OF SOFTWARE MEASUREMENT**

Some sort of measures are required to assess the quality of the final product in a software design and to better understand the models produced. The main aim is to improve the software process on a regular basis. This helps in estimation, quality control project control throughout the life cycle of a software

Project Decision making and further planning becomes easy and fruitful. There are two types of measurements:

1. Direct Measurement
2. Indirect Measurement

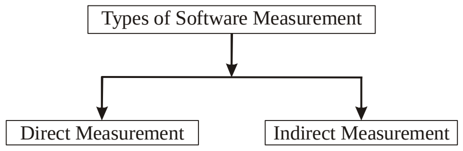


Fig. 3.4: Types of Software Measurement

3. **Direct Measurement:**When the attribute of an entity is measured directly without taking consideration of other attributes, we name it as direct measurement. Direct measures include cost and effort applied and products like lines of code, execution speed, memory size and other defects that have been found.
4. **Indirect Measurement:**There are some attributes like functionality, quality, complexity, reliability, maintainability efficiency can't be measured directly. They are calculated on the basis of the study of other attributes.

The measurements once taken help the project manager and the entire software team to take right decisions that will lead to successful completion of the project.

3.12 **CRITERIA OF MEASUREMENT**

The measurement program is considered successful if it follows the following key characteristics.
diagram

1. **Objective and Repeatable:** The measurement program should be very easy to calculate and understand the same measurement tool should be possible to repeat again and again on similar set of data, without any problem.
2. **Timely:** The measurement program must be should be made available at the correct time to put on influence on the quality development and maintenance of the software project.
3. **Iterative:**The measurement process should be iterative which means that the project manager and team members should continually focus measurement efforts on the most critical areas. It should continually work unstopped during the project life cycle to provide necessary information and knowledge to the project managers.
4. **Related to information needs:** The whole process which includes collection, analysis and reporting of data must be related directly to the information needs of the decision taking persons. The information needs should be filled on a priority basis and addressed accordingly.
5. **Interpretation:** Measurement should enable the project manager and team members to gain insight into the design and development of the work products.
6. **FeedBack:** There must be some follow up and measurement communicator's recommendations derived from product metrics to the software team. This develops appropriate metrics for software under consideration.

3.13 **PROCESS OF MEASUREMENT**

A software measurement process is a management tool which if conducted effectively helps the project manager and the entire software team to take steps that lead to successful accomplishment of the project. This process is all about defining what information is needed by the decision makers. Five sets of activities are done:

1. **Formulation:** To measure and develop appropriate metrics.
2. **Collection:** To collect data to derive the formulated metrics.
3. **Analysis:** This calculates and analysis metrics and the mathematical tools are used.
4. **Interpretation:** To attain insight into the quality of the product.

5. **Feedback:** The suggestions / recommendations are forwarded to the software team.

The important stages in this process are:

(i) **Plan Measurement:** In this stage, information needs are identified and the most appropriate measure is chosen to address those needs. Only important measurements are collected which are relevant to the information needs; unnecessary ones are ignored. This stage also determines the resources and the technologies required to implement a software measurement program. An organization is interested in assigning defects, errors, cost and schedules linked to the development process.

(ii) **Perform Measurement:** In this step procedure for defined measure is documented to collect each measure. in other words we can say that this stage revolves around the collection and processing of measurement data. The procedures should be clear and repeatable. A mechanism is needed to verify the correctness and effectiveness of measurement.

(iii) **Evaluate Measurement:** This stage is concerned with the study analysis and evaluation of the main events of the process to ensure that the measurement approach and alerts about extra cost and warns about schedule over runs.

(iv) **Provide feed Back:** Preparation of Reports, Review exercises and project meetings etc. Provide valuable feedback to implement efficient measurement program. This activity ensures that measurement is amply supported at the project level and organization level.

The decision makes and concerned team members operate within these stages and in the end use the measurement information to develop the software as per requirement specifications.

3.14 BASIC CONCEPTS OF SOFTWARE METRICS

Metrics is defined as the quantitative measures that allow software engineers to identify the effectiveness and improve the quality of software process, project and product IEEE defines as metric as "a quantitative measure of the degree to which a system, component or process possesses a given attribute" soft metrics serves the main purpose of identifying and controlling essential parameters that has an effect on the software development. It is also helpful in fulfilling the following objectives:

(i) Quantitative measurement of the size of the software.
(ii) Assessment of the level of complexity.
(iii) Assessment of the strength of the module.
(iv) Assessment of the testing techniques.
(v) It specifies when to stop testing and determines the date of release of the software.
(vi) It estimates cost of resources and project schedule.

Software metrics are needed to quantity the development process and maintenance of software. We get information about the status of an attribute and helps evaluate if an objective way. It becomes easy to make modification plans. To achieve the desired goal, software metrics are used in different projects for a long period of time to obtain indicators software metrics help in project planning and project. Management activity and thus project younger and project team members become capable of taking right path to complete the project.

3.15 IMPORTANCE OF SOFTWARE METRICS

Software Metrics has achieved a very crucial role in finding answers to questions within the discipline of software engineering software metrics is necessary to decide the following things :

(i) Total time required to complete the project.
(ii) Total cost required to complete the project.
(iii) Number of persons and other resources required.
(iv) Approximate maintenance cost required.
(v) What is needed to test better quality and find the ways for better quality.
(vi) Number of errors to be discovered before delivering the product.
(vii) Estimation of effort to make modifications.

Software metrics help in minimising the software problems and thereby making if possible to efficiently measure and evaluate the attributes of the software. Thus we see that software metrics are helpful in giving the right information about the software development process and in this way help in determining the areas of improvement and build a successful system.

3.16 **CATEGORIES/TYPES OF SOFTWARE METRICS**

An effective software metrics helps a software engineer to develop software as per user requirements within estimated schedule and cost estimate etc. For this different type of metrics for measurement are there. They have been categorized on follows:

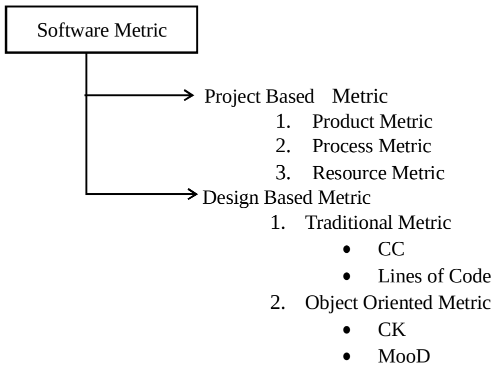


Fig. 3.5: Software Metric

3.16.1 **Projects Based Metrics**

Project metrics illustrate the project characteristics and their execution. Project metrics help the project manager in assessing the current projects. He can identify potential risks and problem areas. After that he adjusts work flow and evaluates the project team's ability to control the quality of work products Projects metrics are used for tactical purposes and not for strategic purposes.

Project metrics help to decrease the potential risks and problems. By making necessary adjustments; project metrics help to minimize the development schedule. This avoids delay, the other purpose of project metrics is to assess the product quality and to modify the technical issues as and when required. By reducing the number of errors and defects, the quality of the project improves and this leads to decrease in the overall cost of a software project.

In project metrics; often the first application takes place in estimation process. The study and the metrics of the previous projects help in estimating time and effort for the current project. As the project moves ahead, original estimates of time and effort are compared with the new measures of effort and time. This helps the project manager to supervise and control the project.

Project metrics are used to track the errors detected during each development phase. For example of software progresses from design to coding, project metrics are used to assess quality of the design and obtain indicators which will affect the approach chosen for coding and testing. Project metrics are used to measure rate of production in terms of models developed, function points and delivered source lines of code.

3.16.1.1 **Product Metrics**

At the end of each phase, a working product is developed in the software development process. Each product can be measured at any stage to check whether it is according to the required specifications or not. Metrics are developed for these products. If a product does not meet user requirements; then necessary steps are taken. Product metrics help the software engineer to detect and correct potential errors before they may result into bigger problems. Product metrics assess the internal attributes of the products so that the efficacy of the following may be judged:

(i) Analysis, design and code model.

(ii) Potency of test cases.

(iii) Overall quality of the software under development.

The following different types of metrics have been devised for products in the development process:

(i) **Metrics for Analysis Model:** These metrics study and find answers to various aspects of the analysis model like system functionality, system size etc.

(ii) **Metrics for Design Model:** Software engineers take the help of the metrics to assess quality of design that includes architectural metrics, component level design metrics and so on.

(iii) **Metrics for source code:** These metrics are used to assess source code complexity, maintainability and other characteristics.

(iv) **Metrics for testing:** These help in designing efficient and effective test cases and if also evaluates the effectiveness of testing.

(v) **Metrics for maintenance:** To assess the stability of the software product.

3.16.1.2 **Process Metrics**

Process metrics assess the effectiveness and quality of the software process. It also determines the maturity of the process and the amount of effort required to develop the project. Software engineers measure specified attributes to improve the quality of the project. In order to improve any process, a set of meaningful metrics are developed to measure its specified characteristics. These metrics are used to obtain indica for to devise a plan for process improvement. With the help of process metrics if becomes easier for a software engineer to assess the efficacy of the software process. Three face namely product, people and technology have a great influence on software quality and organization performance as is from the given diagram:

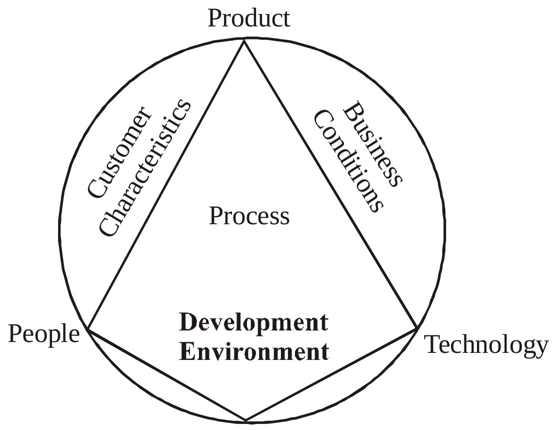


Fig. 3.6: Effect of Factors On Software quality & Organization

The skill and motivation of people, the complexity of the product and the technology level have a great influence on the quality and team performance. For measuring the desired qualities in a software process, a set of metrics is formulated based on different outcomes derived from the process. These outcomes include the following:

(i) Number of errors detected before the release of the software.

(ii) Defects found by the user and reported after the delivery.

(iii) Tune spent in the fixation of errors.

(iv) Work products delivered.

(v) Human efforts used

(vi) Time expended

(vii) Conformity to schedule

(viii) Wait time

(ix) Number of modifications

(x) Estimated cost compared

Types of process metrics: The process metrics are of two types —

(i) Private Metrics,

(ii) Public Metrics.

Private Metrics : This type of metrics are private to the individual and serve as an indicator only for the specified individual(s) e.g. defect rates by a software module and defect errors by an private individual.

Public Metrics: If includes that information which is private to both individuals and teams. E.g. project level defect rates, effort and related data collected analyzed and assessed to get indicators that help in organizational process performance.

Process Metrics Etiquette: Process metrics can be beneficial because the organization works to improve its process maturity. These metrics if mis-used can create problems for the organization. The following are the guidelines to be used by managers and software engineers to avoid misuse of the metrics.

Guidelines

(i) Rational thinking and sensitivity toward organization should be the first priority.
(ii) Feed back to the individuals or teams is very necessary to be provided.
(iii) Metrics should neither appraise nor threaten individuals.
(iv) Use of single metrics should be avoided.

As the process goes on, the derivation of simple indicators lead to a stringent approach called statistical software process improvement (SSPI). SSPI uses software failure analysis to know about all errors and defects faced during the development.

3.16.1.3 **Resource Metric**

Resource metrics is a source code metric and quality analysis tool unlike any other on the market. It provides a standard method for analyzing C, ANSI C++, C# and Java source code across operating systems.

The unique ability of Resource Metric to support virtually any operating system provides enterprise with the ability to standardize. The measurement of source code quality and metrics throughout organization. Resource standard metrics provides the fastest, most flexible and easy to use tool to exist in the measurement of code quality and metrics.

Example Resource Metrics

* Effort expended
— On tasks with a project, classified by life cycle phase software function.
— On extra project activities.
— Training
* Elapsed Time
* Computer Resources

3.16.2 **Design Based Metrics**

The goal of having metrics for the software design is to assess the quality of the design which finally assesses the quality of the product. The quality of the intermediate product is of interest in the development process because it is believed that this leads to a high quality product. The software design complexities are too many and knowledge collected from various fields can be useful in designing systems. It is important to develop software metrics from which meaningful indicators can be derived. With the help of these indicators necessary steps are taken to design the software keeping in view user requirements. Various design metrics such as architectural design metrics, metrics for object oriented design (MOOD), Component Level Design Metrics and user interface design metrics are used to indicate the complexity, quality and so on.

Design metrics can be extracted much earlier in a project and if helps to predict greater number of activities. Large number of Recent Software Design Metrics can be divided into three categories—(i) Network Metrics, (ii) Stability Metrics, (iii) Information flow Metrics. Software metrics measures the product under development and after a system are fielded. Design Metrics includes three activities:

(i) **Total Number of Modules:** This metric specifies the total number of modules. The fundamental benefit of this metric is that by using an average size of a module, a size estimate of the final product can be ascertained. If there is difference in the initial size estimate, it will require a change in plan especially allocation of personal, the schedule and the cost of the product etc. and it will have to be negotiated with the client.

(ii) **Number of modules called:** This metric is computed easily after the completion of the software design. For each module, we can calculate how many other modules if calls and this can also be used to determine how many modules call a particular module. A module with high fan in may point that module has functional cohesion of module with high fan out means that the module depends on to many modules where as a high fan in means that a number of other modules depend on this. During software design, the designer must evaluate those modules that are called by a large number of module, or that call a large number of modules.

(iii) **Number of Parameters:** The software a design specifies all those modules that will be present in the system. It also specifies then inter-connections with other modules. When the software design reaches its completion; a number of parameters of a module can be obtained. Some questions arise about the constitution of a parameter. Whether a complex object in a parameter list be counted as a single parameter? or it should be considered to be made up of many parameters ?

The simplest way is that to consider each logical entity in the parameter list as a parameter. The second way is to think about each basic data which is used in a module as a parameter.

This metric in a way tries to find "Coupling between Modules". When the module performs so many functions and the parameter list is very large; then a high number of parameters can have low cohesion. Modules with higher number of parameters will be more tightly coupled with their caller modules than those with a small number of parameters.

This means that more effort and time will be needed to understand such modules. Any modification in them will affect other modules as well so the job of the maintainer becomes a bit difficult. It is clear that module with large number of parameters needs careful scrutiny.

3.16.2.1 **Traditional Metrics**

Traditional metrics are applied to the methods that carry out the operations of a class in an object oriented system. The method is a component of an object that operates on data after receiving the message and is defined as part of the declaration of a class. This method demonstrates how a problem is broken into segments and the capabilities other classes expect of a given class. The following three metrics given below are traditional metrics — (i) Cyclomatic Complexity, (ii) Size Metric, (iii) Comment Percentage

The traditional metrics have been widely used. Researchers and practitioners well understand them. The traditional metrics are related to software quality attributes which have been validated.

Metric–1: Cyclomatic Complexity: It is used to evaluate the complexity of an algorithm in a method. A method will low Cyclomatic complexity is considered better. Due to in heritance, cc cannot be used to measure the complexity of a class. Cyclomatic complexity of individual methods can be combined with other measures for the evaluation of complexity of the class. Although this metric is used to evaluate quality attribute complexity; it is also related to all of the other attributes. The formula for testing is equal to the number of edges minus the number of nodes plus 2. Number of independent test paths = Edges – nodes + 2. Number of independent test paths = Edges – nodes + 2

Metric–2: Size Metric: Size of a class is used to evaluate the case of understanding of code by developers and maintainers. Methods to measure size can be counting all physical lines of code, the number of statements and the number of blank lines. Coding language and the complexity of the method form the basis for evaluating the size measures. Size affects the ease of understanding, routines of large size will always pose a higher risk in the attributes of understanding, Reusability and Maintainability. The size metric include — Total lines of code, Total function calls, Number of windows.

Metric—3: Comment Percentage: The line counts which compute the size metric can be expanded further to take into account a count of the number of comments both on line i.e. with code and stand alone. The comment percentage is equal to the total number of comments divided by total lines of code less the number of blank lines. This metrics is used by Developers and Maintainers to evaluate the attributes of Understandability, Reusability and Maintainability.

3.16.2.1.1 **McCabeCyclomatic Complexity (CC)**

- It is a software metric developed by Thomas J. McCabe in 1976. It is a very useful, logical metric. It tests the complexity of the program. It also estimates the amount of effort required to understand the code. It is created by calculating the number of different code paths in the flow of the program. A program that has complex control flow will require more tests to achieve good code coverage and will be less maintainable.
- It measures the number of linearly independent paths through a source code. Cyclomatic complexity is calculated using the control flow graph of the program. The nodes of the graph correspond to indivisible groups of commands where as a directed edge links two nodes, if the second command immediately follows the first command.
- Cyclomatic complexity may also be applicable to individual functions. CC is an indication of the number of 'Linear' segments in a method. A method with no branches has a cyclomatic complexity of since there is 1 arc. This number is incremented whenever a branch is encountered. CC is procedural rather than an OO metric.
- Cyclomatic complexity helps in estimation of complexity of code, testing effort and program reliability.
- **The statements which represent branching as:** 'For', 'while', 'do', 'if', 'case', 'catch' optional and ternary operator (optional). If the source code contained no decision points like 'IF' statements or 'FOR' loops, the complexity will come out to 1 because there is only one path. But if the code had a single IF statement, there would be two paths through the code:
  (i) First path where the IF statement is evaluated as TRUE.
  (ii) Second path where the IF statement is evaluated as FALSE.
- **The formula of the cyclomatic complexity is defines as**:

CC=E-N+P
CC= Cyclomatic Complexity
E= Number of Edges
N= Number of Nodes
P= Number of Connected Components

Example 1:

If number of edges= 7, number of nodes= 6 and number of connected components= 2.
Then cyclomatic complexity of ***figure*** *5.4 is:*
CC= 7-6+2 => CC=3

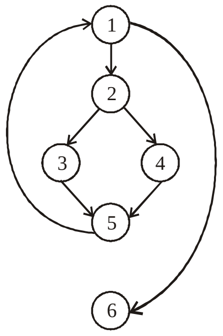


Fig. 3.7: Example of Cyclomatic Complexity

Example 2:

If number of edges= 10, number of nodes= 8 and number of connected components= 1
Then cyclomatic complexity of ***figure*** *5.5 is:*
CC= 10-8+1 => CC=3

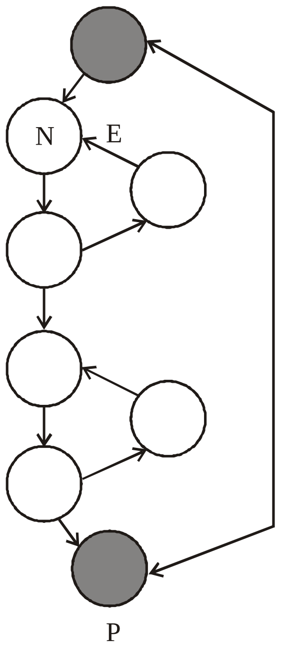


Fig. 3.8: Diagrammatic representation Structured Program

- It is clear CC is calculated by creating a connected graph of the source code. Each line of code is considered as a Node and the arrows between the nodes show the Execution Path. After cyclomatic complexity of a code has been calculated, then we can compare its complexity of other programs as per the standard range tabulated as under:

Cyclomatic Complexity	**Code Complexity**
(a) 1-10	It is a simple program with less risk
(b) 11-20	More complex program with moderate risk
(c) 21-50	Complex with high risk
(d) CC > 50	Untestable having very high risk

- Second method to determine the metric is to count the number of decision points (conditionals) and adding

CC = D + 1 Where D is the number of decision points in the source code.

Advantage of McCabe Cyclomatic complexity

(i) It can be used as a case of maintenance metric.
(ii) When it is used as a Quality Metric, gives relative complexity of various designs.
(iii) It can be computed early in life cycle of halstead's metrics.
(iv) If measures he minimum effort and best areas of concentration for testing.
(v) If guides the testing process by limiting the program logic during development.
(vi) It is easy to apply.

Disadvantages

(i) It does not measure the Data Complexity. It measures only the program's control complexity.
(ii) The same weight is placed on nested and non-nested loops. However, deeply nested conditional structures are harder to understand than non-nested structures.
(iii) CC may give a misleading figure in the case of simple comparisons and decision structures.

3.16.2.1.2 **Source Lines of Code**

SLOC is the most common metric or size today and is most widely used for size estimation. Line of code is easy to compute and enjoys some reputation. If highly depends on programming language because code writing varies from one programming language to another. Simple line metrics like errors per LOC, defects per LOC, cost per LOC etc. can be derived from LOC. Lines of code has also been used to pre-guess program complexity, development effort, and programmer performance.

LOC is the earliest and simplest metric for estimating the effort and size of a computer program even then there is no standard definition of LOC. It is due to this reason, different workers may obtain different counts rural emphasis is given to each line of code. LOC is often used during the testing as well as maintenance phases.

Advantages

It is very simple to measure.

Disadvantages

(i) It is dependent on programming language.
(ii) It does not accommodate non-procedural language.
(iii) Poor software design may lead to excessive and unnecessary line of code.

Guidelines for determining LOC

(i) One LOC is for one logical line of code.
(ii) Only those line of code are included which are delivered as part of software.
(iii) Test drivers, test stubs and other support software are excluded.
(iv) The software code written by the software developer is included whereas code created by the application generator is excluded.
(v) All declarations in the programs are counted as lines of code.
(vi) No comment is counted as lines of code.

3.16.2.2 **Object Oriented Metrics**

Source line of code and Functional Point Metrics can be helpful in estimating object oriented software projects. But these metrics do not succeed in providing details for effort and schedule estimation in incremental software development. Different types of metrics have been suggested for object oriented projects which have been given below:

(i) **Number of Scenario Scripts:** These are a series of steps which depict the interaction between user and application. There is direct relationship between number of scenarios and the application size and number of test cases. Which are developed to test the software. The scenario scripts are analogous to use cases.

(ii) **Number of key classes:** Key classes are independent components. The key classes give indications about effort required to develop software and the amount of 'reuse' feature to be used during the development process.

(iii) **Number of support classes:** Support classes can be defined as those classes which are needed to implement the system but are related indirectly to the problem domain. E.g. user interface classes, computation class. For each key class, a support class can be developed. Support classes and key classes function in a similar way.

(iv) **Average number of support classes per key class:** Key classes are defined very early in the software project whereas support classes are defined throughout the project. If average number of support classes per key class is pre-determined; then estimation process gets simplified.

(v) **Number of sub systems:** Sub system can be defined as a collection of classes that is supporting a function visible to the user. After the sub systems are identified if becomes easy to prepare on appropriate schedule, according to which work on sub systems is divided among project members.

Above mentioned projects along with other metrics like effort used, errors and defects detected are collectively used.

3.16.2.2.1 **CK (Chidamber&Kemerer's Metrics Suite)**

CK metrics for 00 Design is the deepest research / study in object oriented investigation. The following six metrics have been defined for it.

Metric 1 : Weighted Methods per class (WMC) : Let us suppose a class C1
Where M1...............Mn are the methods.
and C1............. Cn be the complexity of the methods
Then the formula for WMC = $\sum_{i=1}^{n} Ci$
The following points are to be noted:

* The number and complexity of the method, give an indication of Time and Effort required developing and maintaining the class.
* The number of methods in a class is proportional to the potential impact on children. Larger the number of methods, greater is the potential impact.
* Classes having larger number of methods are likely to be more application specific thereby limiting the possibility of reuse.

Metric 2 : Depth of Inheritance (DIT) : In those cases where multiple inheritance is involved; DIT is the maximum length from the node to the root of the tree :

* If a class is deeper in the hierarchy, if inherits greater number of methods and makes it more complex to predict its behavior.
* Deeper trees constitute greater design complexity.
* The deeper a particular class in its hierarchy; the more is potential reuse of inherited methods.

Metric 3 : Number of Children (NOC) : It is defined as the number of immediate sub classes which are under a class in the class hierarchy and are going to inherit the methods of the parent class :

* If NOC is greater; the Reuse will also be greater because inheritance is a form of Reuse.
* If NOC is greater; there are greater chances of improper abstraction of the parent class.
* NOC is linked to potential influence of a class on the design. Greater NOC in a class may require.

Metric 4 : Coupling between object classes (CBO) : Classes having same properties are said to be :

* Excessive coupling between object classes has a bad effect on the Modular Design and prevents Reuse.

* Inter object class couples should be minimum to improve Modularity and Promote encapsulation. As the number of couples becomes large; the sensitivity to any change in other parts of the design also becomes large so the maintenance becomes more difficult.
* The higher the inter object class coupling, the more rigorous the testing needs to be.

Metric 5 : Response for a class (RFC) :

* It indicates number of methods that can be invoked in response to a message.
* RFC measures potential communication between classes.
* When RFC Grows; the complexity of a class becomes more and more whereas understandability decreases.
* In case of a large number of methods invoked in response to a message; the testing and debugging of the class becomes more complicated and if becomes more difficult for a tester to understand.

Metric 6 : Lack of cohesion in methods (LCOM) : LCOM can be defined as the count of the number of methods pains whose similarity is 0 minus the count of methods pairs whose similarity is not zero. As the number of similar methods in a class becomes more and more; the cohesiveness increases:

* Cohesiveness promotes encapsulation.
* Lack of cohesion years that classes should be split into two or more than two sub classes.
* Low cohesion increases complexity and it increases the chances of errors during the development process.
* If there is any measure of disparateness of methods; it helps to identify flaws in the design of classes.

3.16.2.2.2 **Mood (Metrics for Object Oriented Design)**

The metrics for Object Orient Design refers to the basic structural mechanism of the 00 paradigm as encapsulation (MHF, DHF) inheritance (MIF and AIF), poly morphism (PF), message passing (CF) and are expressed as Quotients. It include the following metrics:

1. **Method Hiding Factor (MHF) :**It is defined as the ratio of sum of the invisibilities of all methods defined in all classes to the total number of methods defined in the system under consideration. The invisibility of a method is the %age of the total classes from which this method is not visible.
2. **Attribute Hiding Factor (AHF) :**It is the ratio of sum of the invisibilities of all attributes defined in all classes to the total number of attributes defined in the system under consideration.
3. **Method inheritance factor (MIF) :**MIF is defined as the ratio of the sum of the inherited methods in all classes to the total number of available methods (locally defined plus inherited) for all classes of the system under consideration.
4. **Attribute inheritance factor (AIF) :**It is defined as the Ratio of the sum of inherited attributes to the total number of available attributes (locally defined + inherited) for all classes of the system under consideration.
5. **Polymorphism Factor (PF) :** It is the ratio of the actual number of possible different polymorphic situation for all classes ci & the maximum number of possible distinct polymorphic situations for class ci.
6. **Coupling Factor (CF) :**It is defined as the ratio of the maximum possible number of couplings in the system to the actual number of couplings not computable to inheritance.

3.17 **SOFTWARE METRIC APPROACH**

Software Metrics are numerical data linked to the development of the software. They have the following four function to play:

(i) **Planning:** Software metrics lays the foundation for cost estimation, resource planning, scheduling, and budgeting.
(ii) **Organizing:** A projects organization is influenced by size and schedule metrics.

(iii) **Controlling:** They provide an important tool to supervise, monitor and control the progress of various development activities.

(iv) **Improving:** Software metrics are used to ensure a quality product in the end. This is achieved by continuous process improvement.

The following steps are required to select, design and implement software metrics so that a light software metric program is developed:

(i) **Identify Metrics Customer:** This is the first step. Those people are identified who will be using metrics for decision making and implement actions. These customers can be programmers, Testers, Experts and users.

(ii) **Select Measurable Goals:** The second step is to select goals at the organization level and the project level. At the organization level these may below cost provider, higher level of customer satisfaction and profit margins etc. At the project level the goals can be to deliver the software product in time, completion of project within budget and to achieve the desired level of quality.

(iii) **Ask Questions:** Some questions are raised and answers to satisfy these questions are found which will ensure the achievement of the goals. The questions can be:

(a) Whether the software product adequately tested or not?

(b) He all the errors properly corrected?

(iv) **Select Metrics:** The further step is the selection of the metrics suitable to provide information to the people that helps them to take right decisions to ensure the achievement of goals after solving problems.

(v) **Select a Measurement Function:** A measurement function is very necessary. This defines the way how to calculate the metric. Some metrics are measured directly by variables.

(vi) **Collect Data :**After taking the decisions about what to measure and how to measure, the data is collected accordingly to compute the metrics.

(vii) **Provide feedBack :**After the computation of the metrics, the need arises to provide feedback by evaluating the metrics. The patterns indicators and goals identified by the metrics are used to determine the need for action a further investigation or to describe the level of confidence in a given result. Software metric approach is better depicted in the following diagram:

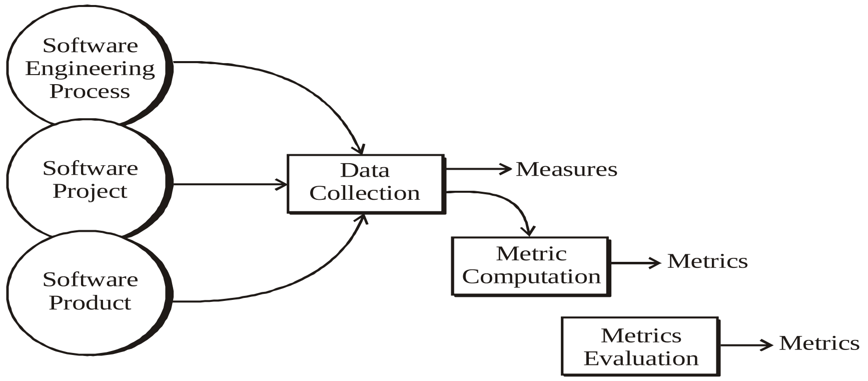


Fig. 3.9: Steps for developing Software Metric program

3.18 **GUIDELINES FOR METRICS**

An ideal software metrics is easy to understand, effective and efficient. An ideal software metrics is always validated and characterized effectively. The following guidelines should be followed to develop metrics:

(i) **Simple and computable:**The derivation of software metrics should be very easy to understand and if should take average among of time and effort.

(ii) **Consistent and objective:** Very clean results should be produced by software metric.

(iii) **Consistency of use of units and dimensions:** Mathematical calculations in metrics should carry the use of dimensions and units in a consistent manner.

(iv) Programming language should be independent and the metrics should be developed on the basis of Analysis Model, design model or program's structure.

(v) **High Quality Product :** If should lead to the production of high quality software product.

(vi) **Easy to Calibrate :** Metrics should be easy to adjust according to the requirements of the product.

(vii) **Easy to obtain:** Metric should be easy to be developed at a reasonable cost.

(viii) **Robust :** Metrics should be relatively insensitive of small change in process project or product.

(ix) **Value :** The value of metrics should be proportional to the value of software characteristic they represent. If one increases or decreases, the other should also increase or decrease relatively.

(x) **Validation :** Metrics should be validated before it is being used for making decisions.

(xi) **Focus :** The main focus of the metrics should be on the product and process.

(xii) **Never to use a single metric:** Software is complex and multifaceted. One should not obsess to much on a single metric. Other metrics should not be avoided.

(xiii) Metrics data should not be ignored or considered negative.

(xiv) Metrics should not be used to threaten the individual or teams.

These people might manipulate the data, the next time. Metrics should never be used as a 'stick'. These guidelines when followed in true sense result into a very high quality metrics.

3.19 QUALITIES OF GOOD SOFTWARE METRICS

A Good software metrics should possess the following characteristics:

(i) **Simplicity:** A good software metrics is very to understand. It should not be complicated.

(ii) **Computability:** It should be easy to compute and derive the metrics. It should not require a lot of time and effort for computation.

(iii) **Objectivity:** It should be of objective and should not be affected by subjective judgments. If should provide clear cut results.

(iv) **Consistency:** The software metrics should show consistency so that even an independent third party should be able to derive the same metric value and using the same type of information.

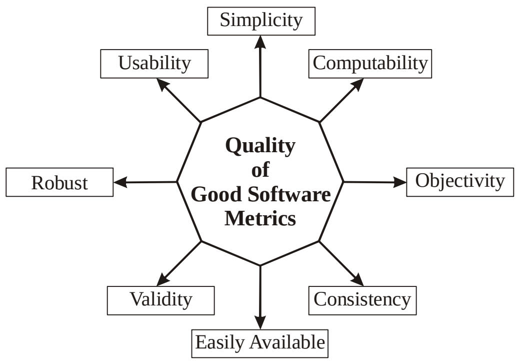


Fig. 3.10 Quality of Good Software Metrics

(v) **Easily Available:** It should not be too much expensive. It should be available at a reasonable cost.

(vi) **Validity :** The metric should measure what is required if the metric should provide valid result.

(vii) **Robust :**If there are minor changes in the product; the metrics should be able to withstand. The changes which can be accidental or intentional security breaches.

(viii) **Usability :** Software metric tries to measure or predict some attribute of some product or process. The software should be user friendly in order to avoid user's reluctance and failures.

3.20 **SOFTWARE QUALITY**

1. Each type of customer will have their own definition of quality. It means quality definition may differ from person to person. For example: an organization defines quality in terms of profits, whereas user defines quality as bug free, within budget and user friendly.
2. Quality can be defined as:

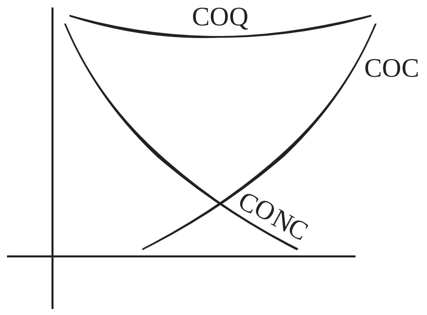
- ***According to Oxford dictionary: Degree of excellence***
- ***According to Edward Deming: Fitness for purpose***
- ***According to ISO: The totality of characteristics of an entity that bear on its ability to satisfy stated or implied needs.***

3. Quality of software depends upon many factors like bug-free, portability, reusability, delivered on time, within budget, user friendly, meet user's requirements and maintainability.
4. It is a relentless process means ongoing process.
5. The quality system activities encompass the following:

- Audits of projects
- Review of System
- Development of standards and guideline.

6. Cost of quality depends upon the level of testing and maintenance.

   Cost of quality = cost of conformance + cost of non-conformance
   COQ = COC + CONC



7. Need of Software Quality

(a) If a software has a high quality, it means software has fewer defects. This saves time during testing and maintenance phases.

(b) Higher the quality of software, lower the maintenance cost. It increases the reliability and customer satisfaction.

3.21 **ATTRIBUTES OF QUALITY SOFTWARE**

The attributes are those factors which increases the quality of a software which are as follows:

1. Bug free
2. Portability
3. Reusability
4. Delivered on time
5. Within budget
6. Customer Satisfaction
7. Maintainability
8. User friendly

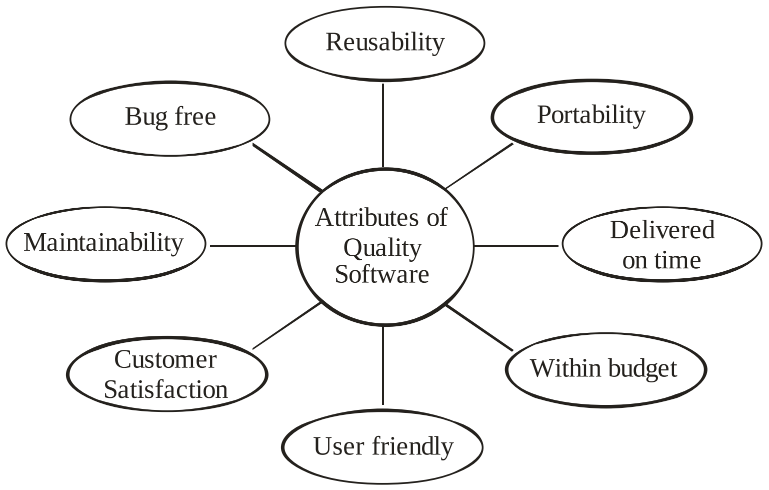


Fig. 3.11 Attributes of Quality Software

1. **Bug-free:**A quality software is always bug free. A software is a quality software if there are lesser number of bugs in the software product.
2. **Portability:** Portability is an attribute in which a software product can be adopted to run on computers (machines) other than the one for which it was designed. According to this feature, a quality software can run on different machines, different operating system environments and with other software products.
3. **Reusability:** According to this feature, we can make some changes in the software product to build a new version of the same product. In this process, minor changes are made in the old software product to make a new software product. In this process, different modules of the product are, reused to develop new product.
4. **Delivered on Time:**A quality software is always delivered on time. It means software is delivered in a fixed time period and within budget.
5. **Within Budget:**A software is called a quality software, if it is under budget of a client. If a client/customer cannot afford a software because of its high cost, then that software is not a quality software for that particular client/customer. A quality software should be within budget of a customer.
6. **User Friendly:** A software product is user friendly if its users find it easy to use. A quality software is always user friendly. A software with many features is never called a quality software if a user faces problems to use it. In a quality software product, users can easily invoke the functions and features of software product.
7. **Customer Satisfaction:** It is another important attribute of software quality product. A software is known as quality software for a particular user if it meets the user's requirements. Software should satisfy the needs/requirements of the customer/user.
8. **Maintainability:**A quality software is maintainable if errors can be easily corrected when they show up, functions of the software product can be easily added to the same software product. This attribute provides the suitability for debugging, extension of functionality and ease of modification.

3.22 **SOFTWARE QUALITY MANAGEMENT SYSTEM**

1. Software quality management system is also known as quality system. Software quality management system ensures that organizations use a adequate methodology to develop a desired quality software.
2. Software quality management system is the responsibility of whole organization. Software quality is not a responsibility of individual person. The top management of the organization should activity participate in this system. If top management of the organization will actively participate in the system, then every member of the organization will take the quality task seriously.

3. Software quality management system involves many activities like auditing and reviewing of the quality system, development of standards, procedures and guidelines etc to enhance the quality of software product.

4. In the software quailty management system, finished products are inspected to eliminate the defective products.

5. This process has mainly four stages which is known as quality assurance method.

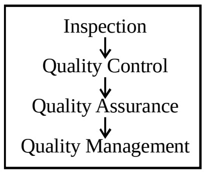


Fig. 3.12 Quality Assurance Method

In the first stage, the product is examined from each angle. In the second stage, quality control finds the defects and eliminate the defects. Quality control also determine the reasons behinds the defects and corrects the reasons of defects. In the third stage, quality assurance takes place. The planned systematic activities necessary to ensure that a component, module or system confirms to established technical requirements. The modern quality paradigm includes certain guidance for recognizing, defining, analyzing and improving the product process. In the last fourth stage, quality management aims at continuous process improvement.

3.22. 1 **Software Quality Management Principles**

The principles of software quality management system are as follows:

Principle 1:Organizations should clearly understand the needs of current and future customers. The aims of the organizations should meet the customer requirements.

Principle 2:Top management of organization should create such an environment in which all people can become fully involved in achieving the organizations common goal.

Principle 3:All people of organization should activily involve in achieving the organization's goals and benefits.

Principle 4:All resources and activities of an organization are managed efficiently to achieve common goal.

Principle 5:Organizations should take effective decisions and actions to achieve desired goals.

Principle 6:Achieving quality is a relentless process means ongoing process.

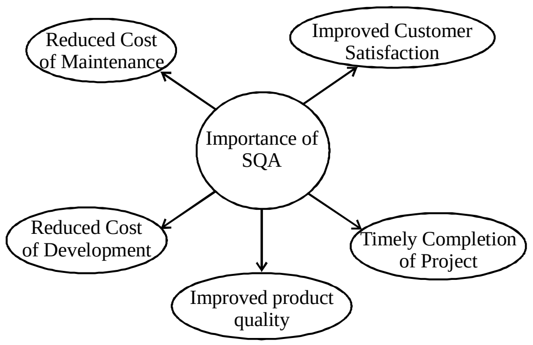
3.23 **SOFTWARE QUALITY ASSURANCE (SQA)**

1. Software quality assurance (SQA) is a planned and systematic approach to the evaluation of the quality.
2. SQA ensures that the standards, processes and procedures are established and followed throughout the software development life cycle.
3. In the software quality assurance, processes and methods are monitored throughout the software development life cycle.
4. The goal of quality assurance is to prevent introducing defects in the software application which helps to improve the development process. Its aim is prevention of defects to improve the quality.
5. Software quality assurance is pro-active process. It identifies the weaknesses in the process but it does not involve in the execution of program.
6. All people who are involved in the software development process are responsible for quality assurance.
7. Verification is an example of quality assurance.

3.23.1 **Importance of Software Quality Assurance (SQA)**

Software quality assurance (SQA) is an important process in the software development approach to enhance the quality of software product. It is a proactive process used to prevent the defects and to improve the quality of the software.

The following are some points which explains the importance of SQA:



Reduced Cost of Maintenance
Improved Customer Satisfaction
Importance of SQA
Reduced Cost of Development
Timely Completion of Project
Improved product quality

Fig. 3.13 Importance of SQA

1. **Improved Customer Satisfaction:** SQA ensures that the developed software meets the customer's requirements and fulfill their desired goals. SQA ensures that software product does what it is supposed to do. Customer satisfaction is the most important factor, for which quality assurance is done.
2. **Timely Completion of Project:** In SQA process, all testing and development processes go smoothly and quickly. SQA assures that the software product is free from errors/bugs/defects. It means that software development project consistently reach towards completion within budget and time.
3. **Improved Product Quality:** The quality of software product gets improved by following standards, procedures and actions. The aim of quality assurance is to prevent introducing defects in the software product which improve the product quality.
4. **Reduced Cost of Development:** In this process, developers are working with SQA team. Both developers and SQA team monitors the software development process from planning to implementation stage to maintain quality. In this way, there are lesser chance of an error to come up in the software development process. The developers don't need to go back to the application and fix something all the time and cost of development is automatically reduced.
5. **Reduced Cost of Maintenance:** If a software product is properly quality assured it means software is bug free. It will give less chances to users to complain and time spent on maintenance will be much less.

3.23.2 **Activities of Software Quality Assurance(SQA)**

The Software Quality Assurance is the process of evaluating the quality of a product. It is an umbrella activity that ensures conformance to standards and procedures throughout the SDLC of a software product. For the enhancement of quality in the software product, the following activities are performed:

1. Formulating a quality management plan
2. Applying software engineering techniques
3. Conducting formal technical reviews
4. Applying a multi-tiered testing strategy
5. Enforcing process adherence
6. Controlling change
7. Performing SQA audits
8. Keeping records and reporting

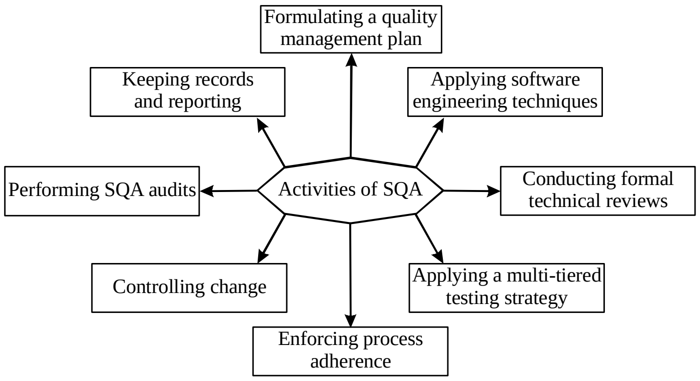

Formulating a quality management plan
Keeping records and reporting
Applying software engineering techniques
Performing SQA audits
Activities of SQA
Conducting formal technical reviews
Controlling change
Applying a multi-tiered testing strategy
Enforcing process adherence

Fig. 3.14 Activities of SQA

1. **Formulating a Quality Management Plan:** The first task of SQA is the formulation of a quality management plan.

- Thequality management plan identifies the quality aspects of the software product tobe developed.
- Plan check points for work products and thedevelopment process.
- Tracks changes made to the development processbased on the results of the checks.

2. **Applying Software Engineering:** The software engineering theories and practical techniques help the software designer to achieve high quality specification.

- Information gathering techniques help designer to collected information.
- The project estimation techniques calculate the estimation of the project.

3. **Conducting Formal Technical Reviews (FTR):** The formal technical review (FTR) in conducted to perform following activities:

- To assess the quality and design of the prototype.
- It is a meeting with the technical staff to discuss the quality requirements of a software product and its design quality.
- It helps in detecting errors at an early phase of development.
- It prevents errors for moving down to the latter phases and resulting in rework.

4. **Applying a Multi-tiered Testing Strategy:** The software testing is a primary task of SQA activity which is described as

- Error detection and correction in project.
- Different types of testing is performed in sequence, fest of all unit testing is performed and in the subsequence levels, integration testing and system level testing are applied.

5. **Enforcing Process Adherence:** It emphasizes the need for process adherence during product development. It is a combination of two tasks, product evaluation and process monitoring.

6. **Product Evaluation:** The product evaluation is conducted to ensure the following:

- To ensure that the standards lay down for a project are followed or not.
- To ensure that the compliance of the software product to the existing standards is verified.
- To ensure that the software product reflects the requirements identified in the project management plan

7. **Process Monitoring:** The process monitoring ensures and monitors the following activities:

- Does the appropriate steps to follow the product development procedures are carried out.
- Monitors processes by comparing the actual process with the written documented procedures.
- It ensures that the development and control processes described in the project management plan are correctly carried out.
- It ensure that products and processes confirm to standards and procedures.
- Audits ensure that product evaluation and process monitoring are performed.

8. **Controlling Change:** It evolves both human and automated procedures to provide a mechanism for change control. The change control mechanism is implemented during the development and maintenance stages.

- It ensures software quality by formalizing requests for change.
- It evaluates the nature of change.
- It controls the impact of change.

9. **Performing SQA Audits:** TheSQA audits scrutinize the software development process by comparing it to established processes.

- It ensures that proper control is maintained over the documents required during SDLC.
- It ensure that the status of an activity performed by the developer is reflected in the status report of the developer.

10. **Keeping Records and Reporting:** The keeping records and reporting involves the process of collection and circulation of information relevant to SQA. The results of reviews, audits, change control, testing, and other SQA activities are reported and preserved for future reference.

3.23.3 **Software Quality Metrics**

Software quality metrics are used to express the quality of software product. No single metric is there which appropriately explains the quality of the software product. Software quality metrics are some measurements which are related to system, process and documentation of software. Software quality metrics are divided into three types which are as follows:

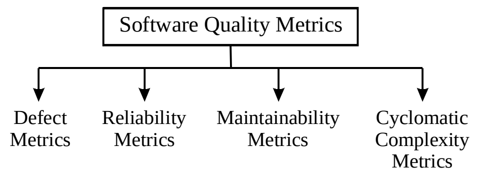


Fig. 3.15 Metrics of Software

1. **Defect Metrics:** Defect metrics provide the number of defects observed in a particular software product. It is used to reflect the man-hours spend during testing to resolve the defects. Defect metrics also explain the types of defects like computation defects, usability defects, requirement objects, design defects etc. to prevent the future problems.

2. **Reliability Metrics:** Reliability metrics are used to know the probability of software failure or the rate at which software will occur. Reliability metrics are discussed in chapter 11. 3.

Maintainability Metrics: Maintainability metrics are used to measure the maintainability of the software product. The maintainability of a program is related to its complexity. The complexity is measured by assuming no. & frequency of operators, no. & frequency of operands.

4. **Cyclomatic Complexity Metric :** It is a very useful and logical metric. It tests the complexity of the program. It also estimates the amount of effort required to understand the code. It is created by calculating the number of different code paths in the flow of the program. A program that has complex control flow will require more tests to achieve good code coverage and will be less maintainable. It measures the number of linearly independent paths through a source code. Cyclomatic complexity helps in estimation of complexity of code, testing effort and program reliability.

3.23.4 **Principles of SQA**

The SQA team has to follow some principles which ensure that the application will live up to the expectations of the users. Some commonly known important principles which are used for proper execution of software quality assurance (SQA) are as follows:

1. **Feedback:** The time is always a best friend and biggest enemy of any developer. If SQA team give feedback early to the developer, then developer can easily take action. In short, faster the feedback, faster the development will move forward.

2. **Motivation:** Quality assurance is a very tedious task. SQA team members should be motivated, passionate and have right mindset so that they can work efficiently, effectively and with creativity.

3. **Evolution:** This principle is for future use. SQA team should be able to mark every time something new is done. Every time something new happens, it should be always noted.

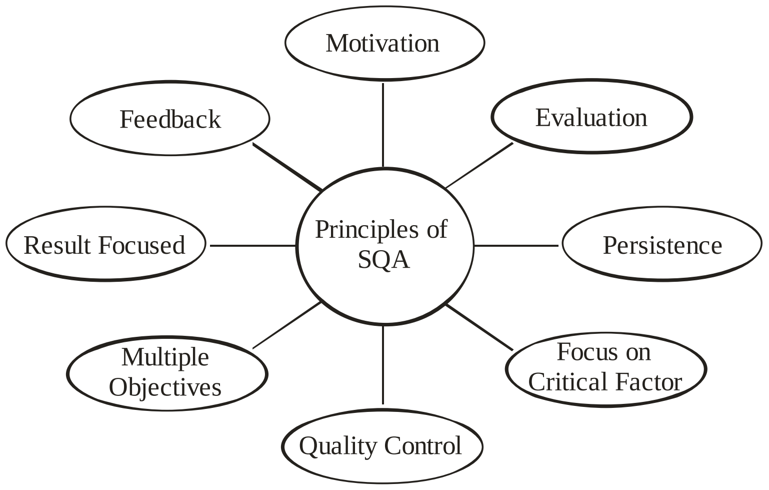


Fig. 3.16 Principles of SQA

4. **Persistence:** The SQA team members should be very patient in every aspect of the software development process. There is no perfect application and every port should be scrutinized without hesitation.

5. **Focus on Critical Factors:** There are many factors developed in the software are not as critical compared to others. SQA team members should be focus on more critical areas first.

6. **Quality Control:** Quality control is the pillar of quality assurance. Everything needs to have quality control from start to the finish. The biggest and tightest quality control should be executed as early as possible which emphasis on where to start.

7. **Multiple Objectives:** This principle is a challenge and risk for SQA team members. At the start of SQA planning, the SQA team members should have more than one objective and each objective should be focused on.

8. **Result Focused:** The SQA process should always look for results whenever a phase is set. SQA should be result focused because it effects to the clients and users.

3.23.5 Software Quality Assurance Standards

The following are some standards which assure the quality in a software product:

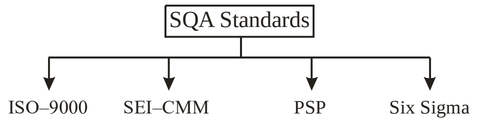


Fig. 3.17 Standards of SQA

I. ISO-9000

1. ISO is also known as International Standard Organization. It is a non-governmental organization. It is a network of 63 countries which is established to formulate standardization. In 1987, ISO published its 9000 series of standards. ISO 9000 interpret the ISO standards for software industry in 1991 because manufacturing industry and software industry have different standards. In India, ISO certification is offered by BIS (Bureau of Indian Standards) and IRQS (Indian Register Quality System).

2. **ISO 9000 Certification:** ISO 9000 certification is a contract between independent parties. It is basically a set of guidelines. It is a series of three standards which are as follows:

- ISO 9001: It is used in software development organizations. It is applicable to organizations which are engaged in design, development and servicing of goods.
- ISO 9002: It do not design products. It is involved in production only.
- ISO 9003: This standard is applicable to those organizations which are involved in installation and testing only.

3. **Reasons to get ISO 9000 Certifications:**

- If an organization have ISO 9000 certification, confidence of customers in that organization increases.
- This certification provides basic framework for the development of the software product.
- ISO 9000 certification points out the weak points of an organization and recommends some actions to the organization so that they can improve those weak points.
- This certification helps the organization to make development process more focused, effective, efficient and cost-effective. It increases the quality of a particular software product.

4. **Procedure to get ISO 9000 Certification**

- An organization who wants to get ISO 9000 certification, has to register itself to a registrar. It is known as application stage.

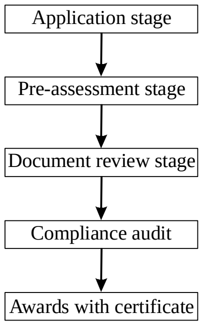


Fig. 3.18 Procedure to get ISO 9000 Certification

- After the registration, the registrar makes rough assessment of an organization. It is known as pre-assessment stage.
- In the third stage, the registrar reviews all the documents which are submitted by an organization during application stage. It is known as document review stage. The registrar suggests some possible improvements to the organization.
- During the fourth stage, which is known as compliance audit, registrar checks whether the organizations made those improvement or not which were suggested by him during document review stage.
- At the last stage, the registrar awards an organization with ISO 9000 certification after the successful implementation of all improvements suggested by the registrar.

After providing the ISO 9000 certification to the organization, the registrar monitors the organizations periodically to check whether the organizations maintain their quality or not. This process is known as continued surveillance.

5. **Issues related to ISO 9000 Certification**

- ISO 9000 certification does not provide any guidelines for an appropriate process and does not guarantees the process to be of high quality.
- There is no international accreditation agency exists which takes guarantee of ISO 9000 certification. Certification is awarded by non-governmental organization.

- After getting the certification, many organizations fail to maintain the quality.

II. **SEI-CMM**

1. It is also known as software engineering institute capability maturity model. It was proposed by SEI, USA in 1986.
2. It is a referenced model and used in two ways: capability evaluation and software process assessment.
3. It is intended to help software organizations to improve the maturity of their software processes. The focus is on identifying the key process areas.
4. SEI-CMM five maturity levels which are as follows:

- **Levels 1—Initial Level:** It is characterized by adhoc activities. It is also called chaotic level because different engineer follow their own process. No standard process is defined and followed during the software development. When the engineers leave the project, then it becomes very difficult for new engineer to understand the process.
- **Level 2—Repeatable:** At this level, size and cost estimation techniques like COCOMO, function point etc. are used to establish the tracking cost and schedule.
- **Level 3—Defined:** ISO 9000 aims to achieve this level. In this level, review techniques are emphasized to achieve phase containment of errors. Development and management activities are defined and documented.
- **Level 4—Managed:** At this level, process metrics and product metrics are collected and measured. Basically focus is on software metrics. Pareto charts, diagrams etc. are used to measure process and metric product metric.
- **Level 5—Optimizing:** For continuous process improvement, process and product measurement data are analyzed in detail. In this manner, organization finds out the best software engineering practices and innovations.

III. **PSP—Personal Software Process**

1. PSP is suitable for individual use. It is a scaled down version of industrial software process.
2. It is a framework that helps engineers to measure and improve the way they work.
3. It does not tell engineers how to analyze, design, code and test the software product. It just helps them to improve their work.
4. The methods of PSP are as follows:

(a) **Time measurement:** Use stopwatch to get an objective picture of the time spent.

(b) **PSP planning:** PSP planning helps individuals to plan their project.

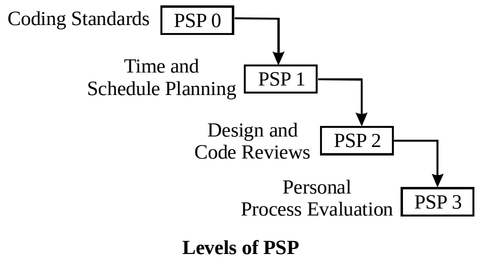


Fig. 3.19 Levels of PSP

IV. **Six Sigma**

1. The purpose of six sigma is to improve process to do things better, faster and at lower cost. It was pioneered by Motorola in 1995.
2. Six sigma is a rigorous and disciplined methodology that uses data and statistical analysis.

3. It improves the company's operational performance by identifying and eliminating defects from products.
3. Six sigma can be defined and understand at three distinct levels—metric, methodology and philosophy. It is applicable to every industry.

3.24 SOFTWARE QUALITY CONTROL (SQC)

1. Quality control is the pillar of quality assurance. Everything needs to have quality control from start to the finish.
2. Quality control is used to verify the quality of the output.
3. The aim of the quality control is to detect the defects and corrects the defects.

Quality Assurance
Quality Control

4. Quality control is a Re-active technique. It is product oriented. The testing team members are responsible for quality control.
5. Activities of quality control are : Reviews, Testing etc.
6. In short, quality control is a subset of quality assurance.

3.25 DIFFERENCE BETWEEN QUALITY ASSURANCE AND QUALITY CONTROL

Sr. No.	Quality Assurance	Quality Control
1	Quality assurance is the process of managing quality.	Quality control is used to verify the quality.
2	Quality assurance is a pro-active technique. It identifies weakness in the process	Quality control is a Re-active technique. It identifies the defects and corrects the defects.
3	The aims of the quality assurance is to prevent introducing defects in the software application.	The aim of the quality control is to detect the defects in the software application.
4	It is process oriented.	It is product oriented.
5	It does not involve executing the program.	It involves executing the program
6	Example—verification	Example—validation or software testing.
7	It is done before quality control.	It is done after quality assurance.
8	It is a staff function.	It is a line function
9	All people who are involved in the software development process are responsible for quality assurance.	Testing team members are responsible for quality control.
10	It is a set of activities for ensuring quality in software engineering processes.	It is a set of activities for ensuring quality in software products.
11	Activities are—Audits, Training, Process definition and Implementation.	Activities are—Reviews, Testing
12	It is a subset of software test life cycle (STLC).	It is a subset of quality assurance (QA).

BECHELOR OF COMPUTER APPLICATIONS (BCA)
BCA-4-01T: SOFTWARE ENGINEERING

UNIT 4: SOFTWARE REQUIREMENT ANALYSIS

4.1 INTRODUCTION

The software development must be based on the requirements raised by the customer. The requirements must reflect the user's view in both for designing and implementation. The documentation of user requirements is done in a systematic in the form of a document SRS. It is written in a natural language, and contains a description of what the system will do without describing how it will do it. It is designed to describe user requirements, system requirements, limitation, functions, economic and social issues. The following section describes requirement analysis, need of SRS, format of SRS and detail about different tools.

4.2 SYSTEM ANALYSIS

1. System analysis is a planned and systematic investigation to determine the functions of a particular system.
2. Its aim is to study the application area and its problems. For this purpose, study involves many data gathering methods like interviews, consultations and observations etc.
3. A person called system analyst conducts this study and identifies various activities, objectives and procedures to achieve his goal.

4.2.1 Role of System Analyst in Software Development Process

System analyst is an engineer who gathers and analyzes the customer requirements. System analyst gathers data and analyzes the gathered data to understand the customer's requirements. The role of system analyst is as follows:

1. System analyst defines problem in a precise and cleared manner.
2. System analyst uses many data gathering methods like interviews, consultations and observations.
3. After the data collection, system analyst analyzes the collected data to get the correct solution of a problem.
4. System analyst also studies the pros and cons of various proposed solutions and choose the one best solution for a problem.
5. System analyst develops plans, design those plans with standards to achieve goals.
6. System analyst visit to customer and make SRS.

4.3 **REQUIREMENT GATHERING AND ANALYSIS**

1. Many projects are failed to satisfy customers. This is because developers started developing the software without clear understanding of what the customers exactly wanted.

2. Whether a developer uses his/her complete knowledge to develop software, but it is impossible to develop a satisfactory solution without a clear understanding of a problem. If a developer fails to understand the customer's requirement, he/she is unable to develop satisfactory software.

3. Today, in many organizations, developers start spending considerable time to understand the exact requirements of the customer. This is known as requirement analysis and specification phase.

4. The aim of the requirement analysis and specification phase is to fully understand the customer requirements and to systematically document these specifications. SRS (software requirement specification) document is the final outcome of the requirement analysis and specification phase.

5. The main activities performed during the requirement analysis and specification phase are:

(I) Requirement Gathering
(II) Requirement Analysis
(III) Requirement Specification
(IV) Requirement Validation

(I) **Requirement Gathering:** It is also known as requirement elicitation.

1. In the requirement gathering activity, system analyst collects the data. It sounds very simple but it is very difficult process. It is very difficult to collect the relevant and useful information from large number of people.
2. For this purpose, working model is used. This working model helps the system analyst to collect the customer's requirements. But in the absence of working model, system analyst uses his/her experience, imagination and creativity for collecting customer's requirements.
3. System analyst uses various methods/ways to collect the customer's requirements which are as follows:

(i) ***Interview:*** Interview is an effective data collection method. It is basically consist of asking questions, listening to individuals and recording their responses. In this method, one is interviewer and one is interviewee. Interviewer is one who takes interview and interviewee is one who gives interview. It can be conducted individually or as a group.

The following are the types of interview:

— ***Informational Interview:*** The objective of this interview is to ask for advice and gain more knowledge about a particular field. The knowledge that we gain here will be sharper and more informed.

— ***Telephonic Interview:*** A phone interview is a very cost effective way to collection information. It can last anywhere from 10 to 30 minutes. It is very challenging because interviewer can't see body language of interviewee.

— ***Face to Face Interview:*** It is sometimes known as individual interview. It is based on evaluation method, individual youth, volunteers or parents. The number of interviews and selection of interviewees will depend upon the purpose, time and resources.

Advantages of Interview:

— It provides deep and free response.
— It is flexible and adaptable method.
— It glimpse into respondent's tone, gestures.
— It has ability to probe, follow-up, clarify misunderstanding about questions

Disadvantages of Interview:

— It is costly in time and personnel.
— Sometimes, it become impractical with large numbers of respondents.
— It requires skill among interviewer and interviewee.
— It may be difficult to summarize responses.
— There may be chances of biasness among interviewer, respondent, situation

(ii) ***Questionnaire:*** In this type of data collection method, questionnaire is made which consists of multiple questions. These questionnaires are sent to people via mail, courier or by hand. This is a useful technique because no one knows your intention and body language. But it is the time consuming process.

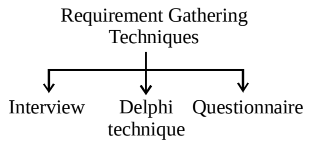


Fig. 4.1 Requirement Gathering Techniques

(iii) ***Delphi technique:*** The Delphi technique is a reliable and creative method. It begins with the development of a set of open-ended questions on a specific issue. These questions are then distributed to various experts. The responses to these questions are summarized and a second set of questions that seek to clarify areas of agreement and disagreement is formulated and distributed to the same group of experts.

Advantages of Delphi Technique:

1. It is conducted in writing and does not require face-to-face meetings.
2. It helps to keep attention directly on the issue.
3. It is inexpensive.

Disadvantages of Delphi Technique:

1. Its information comes from a selected group of people and may not be representative.

2. Its tendency is to eliminate extreme positions.
3. It is a time-consuming technique.
4. It requires skill in written communication.
5. It requires adequate time and participant commitment.

(II) **Requirement Analysis:**

1. The aim of requirement analysis activity is to analyze the gathered data and to obtain the clear understanding of the software product which is going to be developed.
2. The purpose of this activity is to fully understand the exact requirements of the customer and remove all ambiguities, incompleteness and inconsistencies in the requirements.
3. With the help of requirement analysis activity, the system analyst is able to clearly understand the following basic questions:

- What is the problem?
- What are the exactly data input to the system?
- What are the exactly data output to the system?
- What are the possible procedures?
- What are the possible solutions?
- Why is it important to solve the problem?

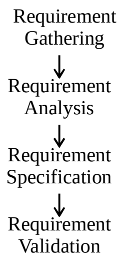


Fig. 4.2 Steps of Requirement Gathering and Analysis Process

(III) **Requirement Specification**

1. During the software requirement specification activity, analyzed information is translated into document form which defines the set of requirements. This document is known as software requirement specification.
2. The document is basically an agreement or contract between the customer and suppliers on what the software product is going to do.
3. The document is prepared by the senior analyst (SA).
4. The document specifies two types of requirements i.e. user requirements and system requirements.

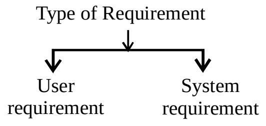


Fig. 4.3 Type of Requirement

User requirements are detailed description of customer requirements. Whereas system requirements are detailed description of the functionality to be provided by the system.

(IV) **Requirement Validation:**

1. In the requirement validation activity, it is checked that whether the requirements mentioned in the requirement specification are complete and consistent or not.
2. In this activity, errors are discovered in the requirement documentation. Requirement documentation is modified to remove the discovered errors.

4.4 **SOFTWARE REQUIREMENT SPECIFICATION (SRS)**

1. SRS is prepared by the senior analyst (SA).
2. SRS is basically an aggrement between the customer and suppliers on what the software product is going to do.
3. It is the final outcome of requirement gathering and analysis phase.
4. It is a complete description of behavior. It describes the scope of the product.
5. SRS document specifies the certain functions of a particular software product in a specific environment.
6. SRS document is very difficult to write. It covers all the requirements of the customer, functional and non-functional capabilities of the software product.
7. All functions/tasks of software development process are based on the software requirement specification (SRS).
8. Users of SRS document: Many people use SRS document according to their needs. The users of SRS document are: customers, software developers, test engineers, maintenance engineers etc. Software developers use SRS document to ensure that they are developing the software product according to the customer's requirements. Maintenance engineers use SRS document to understand the functionality of the software product.

4.4.1 **Need of SRS document**

1. SRS removes the communication gap between customer and supplier.
2. SRS document helps the client's/customer's to fully understand his/her own needs along with the developer's capabilities. It forces client to think and visualize his/her needs and discuss with developer in detail.
3. SRS document almost guarantees that after the development of software product, the client will be happy with this project.
4. With the help of SRS document, the client can determine whether the software product meets his/her requirements or not.
5. If a good quality SRS document is used in the software development process, then the developed software product is of high quality and low cost. The software can be developed and delivered in a fixed duration.

4.4.2 **Attributes/characteristics of SRS document**

The following are the attributes or characteristics of SRS document which enhances the quality of a software product.

1. SRS should be complete. The complete SRS specifies everything about the software product. The complete SRS includes a table of contents, page number, figure numbers etc.
2. SRS should be modifiable in easy manner when requirements are changed.

3. SRS should be unambiguous. Single requirement should have only one interpretation. Single requirement with many interpretation leads to confusion state and developed unsatisfactory software product.
4. SRS should be precise. It is used as a blue print for the coding in a program. It should also follow writing standards. SRS should precisely defines the system's capabilities along with interfaces.
5. SRS should be accurate. It should specifies exact requirements of the customer and system. The correct SRS clearly explains the functional and non-functional capabilities of the system.
6. SRS should be testable. The good SRS document easily identifies and removes its errors.
7. SRS should be traceable. Each requirement in SRS document should be traceable from source to implementation. It should also be able to work in backward direction means from implementation to source in necessary conditions.
8. SRS document should be consistent. Its capabilities, functions and performance levels should compatible and consistent. An SRS should not suffer with conflicting terms, conflicting characteristics and contradictory specifications.
9. SRS should be verifiable. Every single requirement in the document should be verifiable against the developed software product. It removes the communication gap between client and supplier.
10. SRS should be valid. It should be written in natural language so that all participants can easily understand, analyze and accept it.
11. SRS should be well structured. A well-structured SRS document is easy to understand and modify.

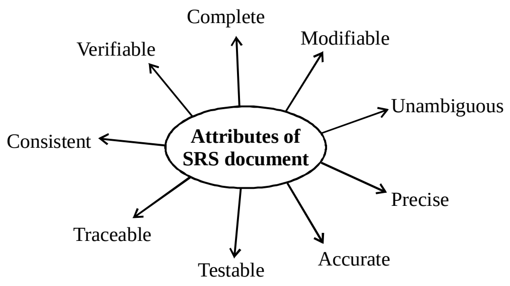


Fig. 4.4 Attributes of SRS Document

4.4.3 Structure/Organization of SRS

The SRS document should be organized in proper sections. The following is the structure of SRS document.

1. Introduction
   a. Purpose
   b. Overview
   c. Environmental Characteristics
      - Hardware
      - Peripheral
      - People
   d. Definitions, Acronyms and Abbreviations
2. Goal of Implementation
3. Functional Requirements
   a. User Class I
      - Functional Requirement
      - Functional Requirements
   b. User Class II
      - Functional Requirements
      - Functional Requirements and so on
4. Non-Functional Requirements
   a. External Interface
      (i) User Interface
      (ii) Software Interface
      (iii) Communication Interface
   b. Constraints
   c. Performance Requirements.
5. Behavioural Description
   a. System States
   b. Events and Actions

Fig. 4.5 Structure of SRS

The following are the explanation of structure of SRS document. These sections are also known as components of SRS document.

1. ***Introduction:*** It explains the context in which the system in being developed. It provides the overall description and environment characteristics of the system. The environmental characteristics specifies the hardware (central processing unit usage, memory usage, network communication etc.) on which the system will run. It also explains the people who are involved in the development and usage of the system. This section also explains the definitions, acronyms and abbreviations such as QoS: Quality of Service. It meets what users actually want and maximize their utility, GUI: Graphical user interface, CPU: Central processing unit of a computer etc.

2. ***Goal of Implementation***: It offers some general suggestions regarding software development. It provides guidelines for the successful development of software product. It is not tested by the user at the time of acceptance testing.

3. ***Functional Requirements:*** It is the critical part of the SRS. It explains all the functionalities required by the user. It specifies which output should be produced from the given input.All the operations to be performed on input data is specified in this document. It describes the relationship between input and output of the system. For each functional requirement, a detailed description of all the inputs, their resources, unit of measure and the range of valid inputs must be specified.

4. ***Non-Functional Requirements:*** This section explains the characteristics of the system such as maintainability, portability, throughput, usability etc. It explains the external interface

(user interface, software interface and communication interface) of the system.It also specify the constraints which restrict the choice of design like security, reliability and fault tolerance. All the requirements related to performance are clearly specifies. The performance requirements are of two types:

- ***Static requirements:*** These requirements do not impose constraints on the execution behaviour of the system.
- ***Dynamic requirements:*** These requirements specify constraints on the execution behaviour of the system.

5. ***Behavioural Description***: It is used to specify the possible states of the system and the transitions among these states due to occurrence of the system. It is not necessary for all the system.

4.4.4 **Reasons of Bad SRS Document**

The following are the reasons of bad SRS document:

1. ***Ambiguous***:The SRS document specify an ambiguity in the requirement. It arises many interpretations of that requirement.
2. ***Incomplete:*** Some requirements have been overlooked and the document left many important aspects of the requirement.
3. ***Inconsistent:*** The requirements in SRS document contradicts each other.
4. ***Unstructured:*** The document is not in structured manner and makes confusion among developers and users.
5. ***Over specification:*** It restricts the freedom of the designer in arriving at good design.
6. ***Wishful thinking:*** When the document specify those requirements which are very difficult to achieve.

4.5 **STRUCTURED ANALYSIS**

1. Structured analysis is the activity of deriving a structured model of the system requirements.
2. Structured analysis techniques help an analyst to decide what type of information is obtained at different points in analysis.
3. It helps to organize information in such a manner that analyst is not over-burdened with complex problems.
4. It is basically a technique/methodology which is used to build a system model which explains the flow and content of information.
5. Special notations and symbols are used to describe the system functionality.

4.5.1 **Objectives of Structured Analysis**

1. The aim of the structured analysis is to define a set of requirements that can be validated once the software is built.
2. It describes the requirements of the customer.
3. It establishes a base for the software design creation.
4. It is basically a documentation for the system which describes the clear and complete specification.

4.5.2 **Principles of Structured Analysis**

The structured analysis approach follows the following principles:

1. The behavior of the system must be represented.
2. The functions that the software is to perform must be defined.
3. The analysis process should move from essential information towards implementation detail.
4. The model should be partitioned in such a way that details should show in a hierarchical manner.
5. The problem of the software must be understood and represented.

4.6 **TOOLS FOR STRUCTURED ANALYSIS**

The following are the tools used for structured analysis:

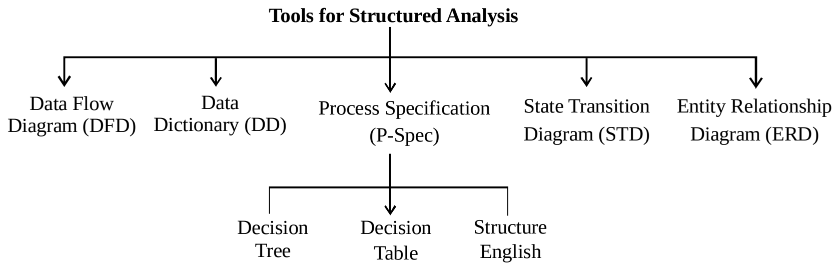
(I) Data Flow Diagram (DFD)

(II) Data Dictionary (DD)

(III) Process Specification (P-Spec)

(IV) State Transition Diagram (STD)

(V) Entity Relationship Diagram (ERD)



Tools for Structured Analysis
Data Flow Diagram (DFD)
Data Dictionary (DD)
Process Specification (P-Spec)
State Transition Diagram (STD)
Entity Relationship Diagram (ERD)
Decision Tree
Decision Table
Structure English

Fig. 4.6 Structured Analysis Tools

I. **Data Flow Diagram (DFD):**

1. It was developed by Larry Constantine. It is also known as bubble chart.
2. It is a graphic representation of the flow of data or information through a system
3. It represents the logical data flow rather than how they are processed. It only show the flow of data through the system.
4. It consists of a series of bubbles joined by lines.The bubbles represent data transformations and lines represent data flow.
5. DFD can be partitioned into levels. Each level has more information flow and data functional details than the previous level.

6. ***Symbols of the DFD:*** There are different types of symbols used to construct DFDs. The meaning of eachsymbol is explained below:

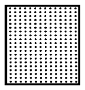
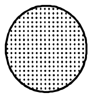
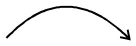
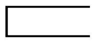
Sr. No.	Symbol	Meaning
1.		Source or destination of data
2.		Process which tranforms data flow
3.		Data flow
4.		Data Store

Table 4.1

7. ***Rules for the construction of DFD:***

(i) The direction of flow of data is from top to bottom and from left to right.

(ii) The names of sources, data stores and destinations are written in capital letters.

(iii) The processes are named and numbered for easy reference. When a process is exploded into lower level details, it is also numbered.

8. ***Level of DFD:***

(i) ***Level 0:*** It is also known as context level DFDs or highest abstraction level DFDs. It depicts the entire information system as one diagram covering all the essential details. It represents scope of the system and identifies external entities and related inputs and outputs

(ii) ***Level 1:*** It depicts basic modules and flow of data among various modules The Level 0 DFD is broken down into more specific, Level 1 DFD. It provides overview of full system. It identifies major processes and data stores.

(iii) ***Level 2:*** At this level, DFD shows how data flows inside the modules mentioned in Level 1. It describes the deeper level of understanding unless the desired level of specification is achieved. Level 1 process is expanded into more detail.

Advantages of DFD:

1. Data flow diagrams are easy to understand check and change data.
2. It give a very clear and simple look at the organization of the interfaces between an application and the people or other applications that use it.
3. It is an effective and efficient method in the absence of required design.
4. It is not limited to software.

Disadvantages of DFD:

1. Modification to a data layout in data flow diagram may cause the entire layout to be changed.
2. Maintenance of data flow diagram is harder, more costly and error prone because in the large application, number of units in a DFD are high and changes are impractical to be made on DFDs.

II. **Data Dictionary (DD):**

1. It is also known as Meta data. A data dictionary is a structured repository of data about data.
2. It is a centralized collection of information.
3. It stores meaning and origin of data, its relationship with other data, data format for usage etc.
4. It removes any chance of ambiguity and helps to synchronize the working of programmers and designers.
5. It is an important and essential step in building a database.
6. Data dictionary contains following information:

(i) ***Data Element:*** It is a smallest unit of data which has no further decomposition. It provides the description of data like: name, source, date of origin, users etc.

(ii) ***Data Structure:*** A data structure is a systematic way of organizing and accessing data. It explains how data are stored in a computer.

(iii) ***Data Flow:*** Data flows are data structures in motion.

(iv) ***Data Stores***: Data flows are data structures at rest. It includes files and tables which stores the information from where the data enters into the system and exists out of the system.

III. Process Specification (P-Spec):

1. It is used to define all the processes which appear in the DFD.
2. All bottom level processes are specified.
3. The tools of process specification are:

Decision Tree	• It is a way of breaking down the complicated situation into easier to understand scenario. • Graphical technique representing decisions using a series of nodes and branches • Each node is a decision point - a choice has to be made • Each branch has a corresponding value to the decision choice • Subsequent action is the result. • It is easy to understand, no need for special training.
Decision Table	• Representation of logic that is part of the processing • Based on a set of conditions, different actions will be performed • Can be simplified by removing impossible actions • Used when the process result is based around several different variables and the logic gets too complex for other methods
Structure English	• In Structured English, decisions are made through IF, THEN and ELSE statements. • It can use CASE construction as well as LOOP construction. • It is basically a subset of English with restrictions on usage. • It can use some acceptable verbs such as FIND, ADD, SUBTRACT, MULTIPLY, DELETE etc.

Table : 4.2

IV. State Transition Diagram (STD):

- It represent a process specification for a control bubble in DFD.

- It is a tool for representing the design.
- It has one initial state and may have multiple final states.
- **State transition diagrams have 4 components :**

States	It is some behavior of a system that is observable and that lasts for some period of time like doing, waiting etc.
Transitions	It is an instantaneous change in state/behavior.
Conditions	A condition is typically some kind of event. It is an event in the external environment which triggers a transition to a new state.
Actions	An action is the appropriate output or response to the event. It is a response sent back to the external environment whose result is stored by the system.

Table : 4.3

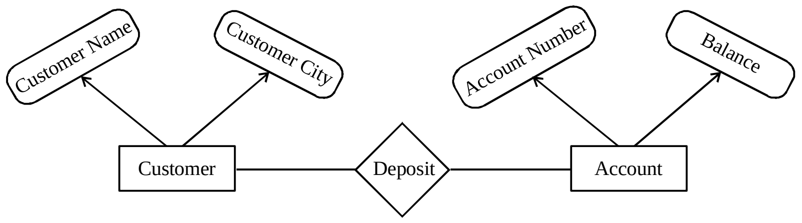
V. **Entity Relationship Diagram (ERD)**

1. Entity relationship diagram is based on real world. It is a collection of entities (objects) and relationship among entities.
2. Entity relationship model has mainly three components:

(i) Entity

(ii) Attributes

(iii) Relationship



E-R Model

Fig.4.7 E-R Model

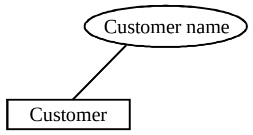
(i) **Entity:**
- Entity is a person, place or thing which can be identifies.
- It is represented by rectangle ▭
- It is of two types:

Weak entity: It depends upon some other entity. It is represented by ⧈

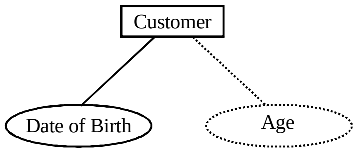
Strong entity: It is not dependent on some other entity. It is represente ▭

(ii) **Attributes:**
- Attributes are the properties of entity.
- It is represented by ellipse ⬭
- It is of four types:

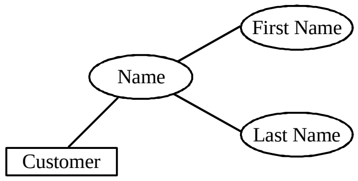
***Single attribute*:** These attributes cannot divided into subparts.



Derived attribute: The value of these attributes are derived from another attribute.



Multivalued attribute: These attributes have more than one value.

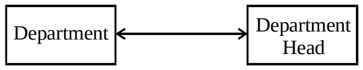


Composite attribute: The value of these attributes can be divided into another attributes.

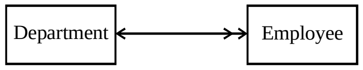
(iii) **Relationship:**

- It is used to connect entities. The entity involved in relationship is known as participants.
- The number of participants in a given relationship is known as degree of relationship.
- It is of four types:

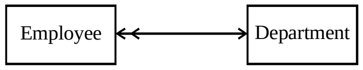
One to one relationship:In one to one relationship, for one record in entity A, there is exactly one record in entity B.



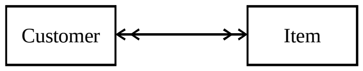
One to many relationship: In one to many relationship, for one record in entity A, there is more than one record in entity B.



Many to one relationship: In many to one relationship, for many records in entity A, there is only one record in entity B.



Many to many relationship: In many to many relationship, for many records in entity A, there are many records in entity B.



BACHELOR OF COMPUTER APPLICATIONS (BCA)
BCA-4-01T: SOFTWARE ENGINEERING

SECTION- B

UNIT 5: SOFTWARE DESIGN

5.1 INTRODUCTION

Software design is a process of transforming the customer requirements into a form that is implementable by using a programming language. It is a very creative process. It is the basis of effective engineering. It should be practiced and learnt by experience and study of the already existing systems. The design phase begins when the requirements document is available. This is the first phase of transforming the problem into a solution. The design process is made up of a set of principles, concepts and practices which help a software engineer to model the system or product that is to be developed.

According to the Stevens in 1991, "Software Design is the process of inventing and selecting programs that meet the objectives for a software system."

In the words of Coad and Yourdon, "Software Design is the practice of taking a specification of externally observable behavior and adding details needed for actual computer system implementation, including human interaction, task management and data management detail."

5.1.1 **Outcomes of a Design Process**

The Design Process is an activity in which software requirements are studied and analyzed to produce a description of the internal structure and organization of the system. During the software design phase, many serious, strategic, technical decisions are taken to achieve the required functional and quality requirements of a system. These decisions help in developing the software in a successful manner and carry out its maintenance in a well-defined manner to improve the quality of the Final Product. Software design transforms the problem into a solution. In this phase the customer requirements, the business requirements and the technical needs, all come together to formulate a product or a system.

5.1.2 **Classification of Design Activities**

A good software design can never be achieved by using a single step procedure but requires several steps. Software Design is an activity in which software requirements are studied analyze and then a systematic procedure is adopted to produce a description of the internal structure and organization of the system that becomes the basis of its construction. Broadly speaking thus are two activities.

1. Software Architectural Design
2. Software Implementation Design

The meaning and scope of these two activities tend to vary considerably. In software architectural design, the top level structure and organization of the system is described and different components are identified. The system is decomposed and organized into components and interfaces between these components are described. The outcome of this architectural design is known as Program Structure or Software Architecture.

In the second activity i.e. the software implementation design, the data structure and the algorithms of different modules are designed and its outcome is usually known as Module Specification Document. In this activity each component is sufficiently described to allow for its coding. After studying different design approaches; seven steps are necessary to obtain a good design these steps are as follows:

(i) Function decomposition, (ii) Interface definition, (iii) Operational time line development, (iv) Data Definition, (v) Concurrency and real time consideration, (vi) Consolidation, (vii) Test procedure development.

5.1.3 **Classification of Design Methodologies**

Software Design is the central activity of software engineering. It is also the integrative activity at the core of software engineering. The designer should be methodical, disciplined, communicative and self-analytical. Various design methods are used today. The design principles and concepts lay a foundation for the creation of a Design Model that encircles around representation of Data, architecture, interface and components. The major software Design Methods include – (i) Function Oriented Design Methods, (ii) Data structure leased Design methods, (iii) Object oriented Design Methods, (iv) Reuse Based Design Methods.

Design Methodology is a very broader area and it is a problem solving activity. Design Methodology focuses on:

1. **Divergence:** It applies critical thinking through qualitative and quantitative approach to create better designs solutions.

2. **Transformation:** It includes better redefining of specifications of design solutions that leads to better guidelines for traditional and contemporary design activities.
3. **Convergence:** Prototypes probable scenarios for better design solutions.
4. **Sustainability:** It controls systematically the process of exploring, redefining and prototyping of designs solutions continually overtime.
5. **Articulation:** It articulates the visual relationship between the parts and the whole.

The true Goal of Design Methodology is to gain key insights to obtain more holistic solutions to achieve better experiences for users with products, services, environments and systems.

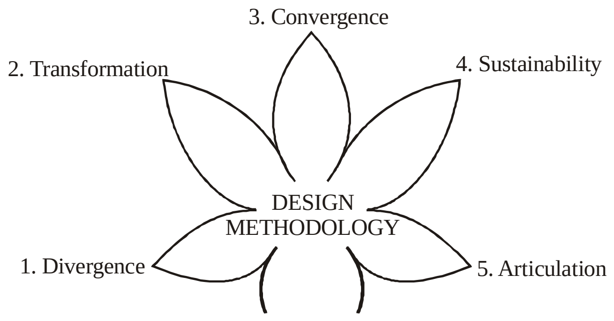


Fig. 5.1: Software Design Methodology

5.1.3.1 **Procedural Approach**

Procedural design techniques are very common and popular and at present they are being used in many software organizations. It is the result of focusing full orientation to the function of the program. This is an approach to software design where the design is decomposed into a set of interacting units where each unit of module performs a specific function Niklans Wirth who is the creator of Pascal and a number of other languages are one of the best known advocates of this method. His special variety to known as step-wise refinement. This stepwise refinement is a top-down strategy where a program is refined as a hierarchy of increasing levels of detail (Mills 1988). Refinement is actually a process of elaboration.

Procedural design is also named as component level design. It occurs after data, architectural and inter face designs have been developed. The aim is to translate the design model into an operational software.

5.1.3.2 **Object Oriented Approach**

Object-oriented design is a design strategy of planning a system of interacting objects for the purpose of solving a reaching a predefined software generated result. These Objects manage their own private state and offer services to other objects and hide representation of the state to limit access to it.

Object-oriented concepts

The five basic concepts of object-oriented design are:-

1. Object/Class
2. Information hiding
3. Inheritance
4. Interface (object-oriented programming)
5. Polymorphism

5.2 **OBJECTIVE OF SOFTWARE DESIGN**

Software Design is expected to deliver the requirement as specified in the Feasibility Report. The quality of the design is assessed with a series of technical reviews or design walk throughs. The main software objectives which a software design should provide are as follows:

Basic Objectives

1. To produce such models that can be analysed and evaluated to find whether they allow the various requirements to be fulfilled.
2. To study and evaluate different alternative solutions and trade-offs.
3. To plan the following development activities.

Major Objectives

1. To understand the requirements and to generate the solution to accomplish that requirements.
2. To transform the problem space of the implementation.
3. To solve complex problems by dividing them into a set of sub-problems and then finding partial solutions easily in a better way.
4. To identify the proper level of detail when design should stop and implementation should start.
5. To define the software architecture and describe if through step by step details.
6. To make the module functionally independent.
7. To implement all the explicit requirements of the Analysis Model.
8. To define the relationship between major structural element of the software.
9. To produce a design that is readable and understandable.
10. To provide a complete picture of the software that can be assessed for quality from an implementation prospective.
11. To translate the customs requirements into a well-furnished software product or system.
12. To follow a simple design approach so that designs are easily understood, easily built and easily tested.
13. To accommodate the changes that may be required during its life time.

5.3 **CRITERIA OF EFFECTIVE SOFTWARE DESIGN**

Most researchers and software engineers agree on certain characteristics that every Good Software Design must possess. The Goodness of a design depends on the targeted application. The definition of a Good Software Design can vary depending on the application on which it is based. The characteristics of a Good and effective Software Design are as under:

1. Correctness—A Good Design must implement correctly all the functionalities of the system.
2. A Good design should be easy to understand.
3. It should be efficient and should accommodate all the implicit requirements desired by the user.
4. It should not be difficult to change.
5. A good design should exhibit on architectural structure with recognizable design patterns.
6. An effective software design should be modular. It should be logically divided into elements that perform specific functions and sub functions.
7. A good design always possesses will written documents.
8. A design is said to be good and effective if it separates data, procedures and tuning considerations as far as possible.
9. In a good design, every requirement is testable. If a design cannot be tested easily against its requirements, then it is unacceptable.
10. Each and every point about the design should be easily communicated to all concerned through proper abstractions and representations.
11. A good design is liable to be repeated or re-used.

Each of the above characteristics is actually a goal of the design process. The design should be readable, understandable and must provide a complete picture of the software and if should fulfill the requirements desired by the customer.

5.4 **DESIGN PRINCIPLES**

Fundamental Principles for any discipline remain the same throughout. They provide the underline basis for development and evaluation of techniques. Fundamental concepts of software design include abstraction, structure, information hiding, modularity, concurrency, verification, testability, discreteness, and reusability and design aesthetics.

Some factors that help in making good software are creative skill, past experience and an overall commitment to quality. Davis has suggested a set of principles for software design which have been adapted and extended in the following list :

1. The Designer Process should not suffer from tunnel "Vision". A Good designer applies alternative approaches judging each according to the requirements of the problem and also takes into consideration the resources available to do the job.
2. The design should be traceable to the analysis model — A Good design must have means to track how the requirements have been satisfied.
3. The design should not re-invent the wheel — A set of design patterns should be chosen as an alternative to re invention. Design time should be used to represent truly new ideas and integrate all preexisting patterns.
4. The design should "minimize the intellectual distance" between the software and the problem as if exists in the real world.
5. The design should exhibit uniformity and integration.
6. The design should be structured to accommodate change.
7. The design should be structured to degrade gently, even when data, events or operating conditions are encountered. If should be designed to adjust according to the unusual circumstances.
8. Design is not coding and coding is not design — even when detailed procedural designs are produced, the level of abstraction of the design model is higher than source code design decisions at the coding level address the small implementation details which enables the procedural design to be coded.
9. The design should be assessed for quality as if being created not after the fact.
10. The design should be reviewed to minimize conceptual errors.

By following the design principles, Software Engineer produces a design of high quality which shows both external and internal quality factors. The user readily observes the external qualities (Speed, correctness, usability and reliability etc.). The internal quality factors are of immense significance to the Software Engineer. The designer must understand basic principles to create a high quality design from the technical perspective.

5.5 DESIGN STEPS

By critically examining and evaluating different design approaches, seven steps have been devised which should be followed to obtain a good design. These steps are as follows:

1. **Function Decomposition:** In the step software product is partitioned into smaller components. Rules are observed to assure that recombining and reassembling the components to intelligent testing scenarios which helps the management in maintaining control over the process. Block diagrams, structure charts and tools are the software development tools. Some characteristics and constraints for partitioning are as follows:

(a) Unit Testing Improvement
(b) System testing enhancement
(c) Cohesion and coupling
(d) Managerial Visibility
(e) Managerial control
(f) Information hiding
(g) Contractual an organizational compatibility
(h) Data, time dependency and function separation within limits.

The components are the building blocks. The partitioning step is the realization of a structure or architecture for the final design of the software product.

2. **Interface Definition:** Component to component and component to external interface are identified and defined. Many tools like object diagrams, data flow diagrams, timing chart and data dictionaries support this step. Another requirement of this step is to provide interface information that a designer, coder and tester needs for using other design components in developing a software product.
3. **Operational time line Development:** In this step realistic operational timeline or scenario to exercise are developed for each design step. These timelines should have operator interactions with the system.
4. **Data Definition:** This step is linked to develop data structures and this step serves the purpose of defining the files, records, fields, access methods, global and local data structure and the procedure to maintain files. Transactions are chalked out and the file size estimates are made. Tools for this step are data dictionary, information modeling and entity relation diagrams.
5. **Concurrency and Real Time Considerations:** The fifth step is linked to concurrence and real-time requirement. It is strongly desired to decouple timing requirements and their design and correctness tests from function, performance and data correctness. This step is used to establish the timing for the design component. The tools used for this step are timeline extensions to data flow diagrams and McCabe's analysis approach.
6. **Consolidation:** Information gathered from in steps one to five and a realization of the consolidated design information is used as a direction to coders in the form of diagrams, pseudo code structured English or any other form of program design languages.
7. **Test procedure Development:** This step studies those ways and means by which the design can be tested as a design component and finds how the design component contributes to confirm a system level acceptance criteria. This slip helps in verification that all requirements in the software requirements specifications have been met and traces if to implementation in the system design specifications.

Control hierarchy and data structure are too important aspects of the overall software products Architecture.

Control Hierarchy: The control relationship among various components of the product is sometime, referred to as visible. Each component has access to many other components and connectivity is done only to those components directly involved by the given component.

Data Structure: It is a representation of the relationship among data items in a software product design organization, access methods and controls, definitions, degree of association etc.

Data structure is consistent with overall architecture and if stays connected with control and procedure design throughout the design process. Formal design reviews can be inserted at proper interval to ensure that design efforts are fully coordinated.

5.6 DESIGN CONCEPTS

By following a rigorous approach in any engineering activity, satisfactory products can be produced. We can control. The costs of the products and increase their reliability. This does not mean to constrain creativity rather if improves the engineer's confidence in precision and accuracy. He brings creative results after critically analyzing in the light of a rigorous assessment.

Some basic steps have been devised over the past four decades which provides a basis to the software designer to apply more sophisticated design methods. He becomes capable of knowing different criteria to partition software into individual components; he comes to know how data structure detail can be separated from a conceptual representation of the software.

In the words of M.A. Jackson, "The beginning of Wisdom for a software engineer is to recognize the difference between getting a program to work and getting if right". The basic concepts for the Software Design are given below:

1. Abstraction

2. Refinement
3. Modularity
4. Software Architecture
5. Control Hierarchy
6. Structural Partitioning
7. Data Structure
8. Software Procedure
9. Information Hiding

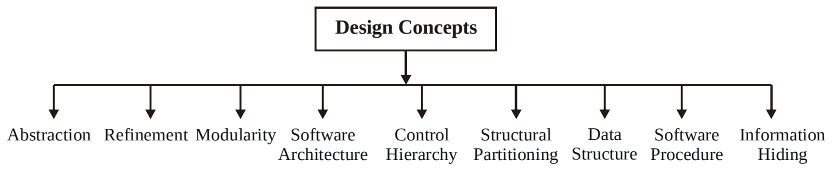


Fig. 5.2: Design Concepts

1. **Abstraction:** It is a very powerful concept and an intellectual tool used in all engineering disciplines which permits a designer to consider a component at an abstract level without worrying about the details of the implementation of the component. It is a special case of separation of concerns where in concern of the important aspect is separated from the concern of the unimportant details. Abstraction is an indispensable part of the design process and is essential for problem partitioning. Abstracts is used to existing components as well as components that are being designed. Abstraction of existing components plays a vital role in the maintenance of the system and we become able to modify the system in a step wise manner. Abstraction reduces the information content of a concept or an observable phenomenon, typically to retain relevant information for a particular purpose. Abstracting happiness to an emotional state of mind reduces the amount of information conveyed about the emotional state. There are different levels of abstractions. These levels are as follows:

(i) **Data Abstraction:** In Data Abstraction; the component is specified by the data types or data object and operations that performed on data object. Like any data object, the data abstraction for door would comprise of a set of characteristics (e.g. door type, swing direction, opening mechanism, weight, length, breadth, height etc.) Many modern programming languages provide mechanisms for creating abstract data types.

(ii) **Procedural Abstraction:** It is a named sequence of instructions that has a specific and limited function. In procedural abstraction component is specified by the function if performs e.g. Auto sum function in Excel. OR the word open for a door that implies a long series of procedural steps (i.e. walk to the door, reach out and grasp knob, turn knob end pull door, step away from moving door etc.)

(iii) **Control Abstraction:** It means a program control mechanism without specifying internal details. If describes the external behavior of that component. In example of a control abstraction is the Synchronization Semaphore used to co-ordinate activities in an operating system.

(iv) **Cluster Abstraction:** In cluster abstraction the component in a group of related classes that work together. These are also called framework e.g. for net frame work.

2. **Refinement:** Stepwise refinement is a top down strategy where in a program is refined in the form of hierarchy. In this technique decomposing of a system is done from high level specifications into more elementary levels. In each step, one part of high level description is taken and refined. It is actually a process of elaboration. Refinement is also known as “Stepwise Program Development and Successive Refinement” Niklans Wirth suggested following activities:

(i) Decomposing design decisions to elementary levels.
(ii) Isolating design aspects those are not truly independent.
(iii) Postponing decisions concerning representation details as long as possible.

(iv) Carefully demonstrating that each successive step in the refinement process is a faithful expansion of previous steps.

The following steps should be revised at each level in the design:

(a) Study and understand the problem.
(b) Identify gross features of at least one possible solution.
(c) Describe each abstraction used in the solution in some design description language.

Refinement begins with the specifications which are derived during requirements analysis and external design. It is a very effective method for describing small sized programs but fails in the large sized programs refinement helps the designer to know low-level details as design progresses.

3. **Modularity:** The real power of partitioning comes if a system is partitioned into modules, so that these modules are solvable and modifiable separately. It is a program unit. A module can be a macro, a function, a procedure a process or a package. Some criteria is followed to select modules so that modules support well defined abstractions e.g. coupling and cohesion are two modularization criteria used in Functional Abstraction.

Modularity promotes design clarity which cases implementation, debugging, testing, documenting and maintenance of the software product. A modular system has the following characteristics:

(i) Each Processing Abstraction is a well-defined sub system which can be effectively used in other applications.
(ii) Each function has a single well defined motto.
(iii) Each function manipulates only one major data structure.
(iv) Functions share global data selectively. It is convenient to identify all solutions that share a major data structure.
(v) Functions that control instances of Abstract Data types are encapsulated with the data structure being manipulates.

4. **Software Architecture:** Shaw and Garlan (1996) suggested that Software Architecture is the initial step in producing a software design. They gave three design levels namely Architecture Design, Code Design and Executable design.

Software Architecture is the hierarchal structure of Program Components (Modules) and expresses the way in which these components interact and structure of data used by the components.

Architecture Design: It links the system capabilities observed in the requirements specification with the system components that will implement them. The architecture describes the inter connections among different modules. He also defines operators who create systems from sub systems.

Code Design: If involves algorithms and data structures the components such as programming languages primitives such as numbers, characters, pointers and control threads. There as primitive operators which involves the language's arithmetic and data manipulation primitives and composition mechanisms (arrays, files, procedures)

Executable Design: The code design is studied of a lower level of detail. It discusses memory allocation, data formats, bit patterns and so on.

Keeping all the above mentioned specifications the architectural design can be represented using any number of the following models in the figure of system I and system II, it is to be noted how one component is connected to the other only when the first component can invoke the other. For a given component, the set of other components to which arrows are drawn is called scope of control of the components. The components invoked by the given component are collectively referred to a scope of effect e.g. in System I-II, are two possible designs for the same system. It should be noted that no component should be in the scope of effect; if it is not in the scope of control. If the scope of the effect is wider than the scope of its control, if is almost impossible to give assurance that a change to the component will not damage the whole design.

Fan in: It is the number of components controlling a particular component.

Fan out: It is the number of components controlled by a component. Component "A" has a fan-out of 3 in system I but a fan-out 5 in system II.

Fan in for component 'C' is 1 in both systems.

Generally a system with high fan-out tries to minimize the number of components. The controlling component performs more than one function. Comparing in this way system I with low fan out may be better than System II.

Studying from another angle, when number of levels are increased a particular component is used more than once. If we design one general purpose component to do that task, the results design is more efficient and easier to test. In such cases design with a large number of levels is considered better one of the goals in designing systems is creating such components which have high fan in and low fan-out.

(a) **Structural Models:** It puts forth Architecture as a systematized collection of program components.

(b) **Frame Work Models:** These models enhance the level of design abstraction. If tries to identify architectural design frameworks that appear in similar types of applications.

(c) **Dynamic Models:** They address the behavioral aspects by indicating how the system configuration may be changed as a function of external events.

(d) **Process Models:** They lay emphasis on the design of the business or the technical process.

(e) **Functional Models:** These models can be used to represent the functional hierarchy of a system.

5. **Control Hierarchy:** It represents the organization of program components and if shows a hierarchy of control. It does not represent procedural aspects of software. It does not apply to all architectural styles. Number of components under the control of a particular component can be represented diagrammatically as under.

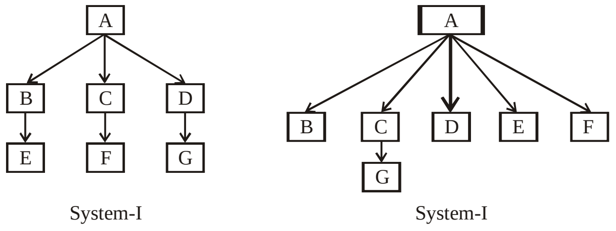


Fig. 5.3: One component controlling other components

6. **Structural Partitioning:** When the architectural style of a design follows a hierarchical nature, the program structure can be partitioned either horizontally or vertically.

Horizontal Partitioning: In this method the control modules (shown as shaded boxes) are used to co-ordinate communications between functions and execution of the functions. It has the following benefits:

- It results in software that is easier to best and maintain.
- It results propagation of fewer side effects.
- It results in software that is either to extend.

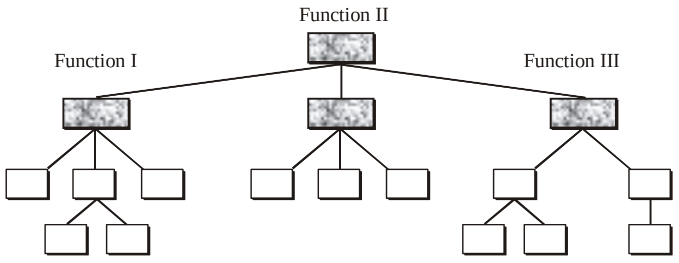


Fig. 5.4: Horizontal Partitioning

Disadvantage of Horizontal Partitioning: In this more data has to be passed across the module interface. This complicates the overall control flow of the problem, especially during speedy movement from one function to another.

Vertical Partitioning: This is also called factoring. In this the control (decision making) modules are located at the top and work is distributed in a top down manner in the program structure. Top level models perform control functions and do little processing. Modules that reside low in the structure perform all input, computation and output tasks.

The nature of change in the program structures requires vertical partitioning because a change in a control module (high in the structure) will have a higher probability of propagating side effects to modules that are subordinate to it. A change to a worker module (low level in the structure) is less likely to propagate side effects. Therefore vertically partitioned structures will be more maintainable a key quality factor.

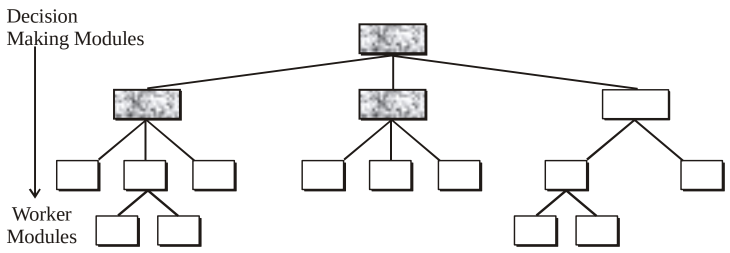


Fig. 5.5: Vertical Partitioning

7. **Data Structure:** Data structure is a representation of the logical relationship among individual elements of the data. As the structure of information has a great effect on the final procedural design. Data structure has an effect on the organization methods of access degree of associativity and processing alternatives for information. Although organization and complexity of a data structure depend on the thinking and skill of the designer, yet a limited number of classic data structures form the building blocks for more highly skilled and well-designed structures.

A scalar item represents a single element of information and it is the simplest form of data structure. Its access may be achieved by assigning a single address in the memory. The size and format of a scalar item are controlled by a programming language and they may very within limits. A scalar item may be a logical entity of one bit length, an integer or floating point number being 8 to 64 bits long or a character string that is hundreds or thousands of bytes long.

When scalar items are well organized in the form of a list or a group, sequential vector is formed. Vectors open the door to variable indexing of information.

When the sequential vector is extended to two, three and finally an arbitrary number of dimensions, an n-dimensional space is created. In many programming languages, an n-dimensional space is called an array. Two dimensional matrix is the commonest n-dimensional space.

Items, vectors and spaces may be organized in various formats & linked list is a data structure that organizes noncontiguous scalar items, vectors or spaces in a manner (called nodes) so that they may be processed as a list. Each node contains the appropriate data organization and one or more pointers that indicate the address in storage of the next node in the list. Nodes may be added at any point in the list. It is done by redefining pointers to accommodate the new list entry. Other data structure is constructed using the fundamental data structures. A hierarchical structure is commonly encountered in applications that require information categorization and associativity. Data structure can be represented at different levels of abstraction. For example, a stock is a conceptual model of a data structure which can be implemented as a vector or a linked list. The internal working of a stock depends on the level of design detail.

8. **Software Procedure:** Program structure defines control hierarchy without taking into consideration sequence of processing and decisions whereas software procedure focuses on the processing details of each module individually. Procedure must provide a precise specification of processing including sequence of events, exact decision points, repeatable operations and even data organization and

structure. There is a relationship between structure and procedure. A procedural representation of software is layered. The processing indicated for each module must include a reference to all modules subordinate to the module being described as is clear from the diagram given below:

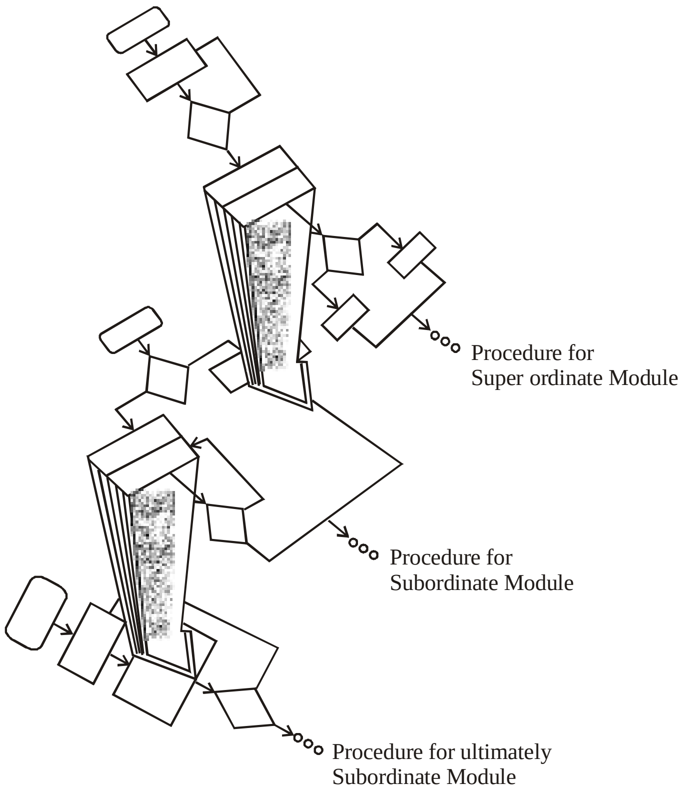


Fig. 5.6: Layered Software Procedure

9. **Information Hiding:** Modules should be designed and specified in such a way that each module in the system hides the internal details of its processing activities and modules may communicate through well-defined interfaces. The way of hiding in necessary details is known as Information Hiding. IEEE defines it as "the technique of encapsulating software design decisions in modules in such a way that the modules interfaces reveal as little as possible about the module's inner workings; this each module is a 'black box' to the other modules in the system."

According to Parnas, design should begin with a list of difficult design decisions and design decisions that are likely to change. Each module is designed to hide difficult and changeable design decisions because these design decisions transcend. Execution time, design modules may not correspond to processing steps in the implementation of the system.

Information Hiding is of much significance when modifications are required during the testing and maintenance phase. In object oriented design it gives rise to the concepts of encapsulation, modularity and is also linked to the concept of abstraction. Information hiding also hides the following information's:

(i) A data structure, its internal linkage and the implementation details of the procedure.
(ii) The format of control blocks such as those for queues in an operating system.
(iii) Character codes, ordering of character sets, shifting masking and other machine dependent details.

Advantages of Information Hiding

(i) It reduces the likelihood of adverse effects.
(ii) Leads to low coupling.
(iii) Attaches importance to communication through controlled interfaces.

(iv) Limits the global impact of local design decisions.
(v) Helps in producing higher quality software.

5.7 DESIGN QUALITY METRICS

Design quality metrics is a methodology of determining the quality of a design. A design is supposed to be good if it implements a specification correctly and allows efficient code to be produced. A good design is one which is maintainable and understandable. A maintainable design can be adapted to modify existing functions and add new functionality. The design components should be cohesive as they should have close logical relationship. They should be loosely coupled and should not be tightly integrated. Design quality metrics are used to assess the quality of design. Coupling & cohesion measures have been developed in conjunction with functional.

5.7.1 Cohesion

The internal cohesion of a module is measured in terms of the close of the relationship between its components or in other words as the strength of binding elements within the module. Cohesion occurs on the scale of weakest elements (least desirable) to strongest elements (most desirable). It shows how tightly the elements of the module are attached to one another. A component should implement a single logical function or a single logical entity. Contribution to implementation should be from parts of components. There is no need to modify many components if a change is desired. Cohesion gives the designer an idea about whether or not the different elements of a module belong together in the same module.

Constantive and Yourdon identify seven different levels of cohesion in order of increasing strength:

1. Coincidental
2. Logical
3. Temporal
4. Procedural
5. Communicational
6. Sequential
7. Functional

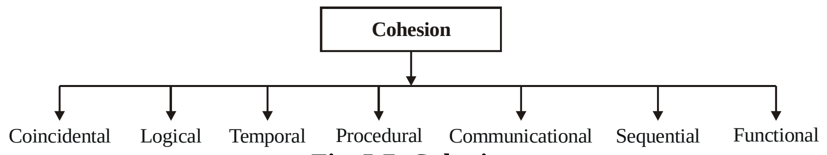


Fig. 5.7: Cohesion

1. **Coincidental:** If occurs when there is no meaningful relationship among the elements of a Module. Unrelated functions, processes or Data are present in the same component for the reasons of easiness serendipity. Coincidental cohesion can happen if an existing program is 'modularized' by cutting if into pieces and creating different pie modules. If a module is created to save duplicate code by combination some part of the code that occurs at many different places that module may have coincidental cohesion. In General, it is not wise to create a most merely to avoid duplicate code or chop a module to reduce the module size.

2. **Logical Cohesion:** A module is said to have logical cohesion if there is logical relationship between the elements of a module. Several logically related functions or data elements are kept in the same component. A typical example of this type of cohesion is a module that performs all the inputs or all the outputs. One component may read all types of input (from tape, disc or telecommunications port). It does not matter where the input is coming from or how it will be used. Although it is more reasonable than coincidental cohesion, the elements of a logically cohesive component are not related functionally. In general, logically cohesive modules should be avoided, if possible.

3. **Temporal Cohesion:** Temporal cohesion and logical cohesion are similar except that in temporal cohesion the elements a related in time and are executed together. Modules that perform activities like "Initialization" "clean up" and "Termination" a generally temporally bound. This avoids the problem of passing the flag and the code is generally simpler.

4. **Procedural Cohesion:** A procedurally cohesive module contains elements that belong to a common procedural unit. Sometimes it is needed that functions must be performed in a certain order e.g.

Data must be entered before they can be checked and then manipulated; all these three functions in a specific order. When functions are grouped together in a component to maintain a proper procedure, the component is procedurally cohesive. Such modules occur when modular structure is determined from same form of flow chart. Procedural cohesion, often, cuts across functional times.

5. **Communicational Cohesion:** In this type of cohesion, elements are related to each other by a reference to same input or output data because they operate on or produce the same data set. Sometime, unrelated data are fetched together because if can be done with only one disc or tape access. Communicably cohesive modules can do more than one function. e.g. "Print and Punch record". It is to be noted that sometimes communicational cohesion may destroy the modularity and functional independence of the design.

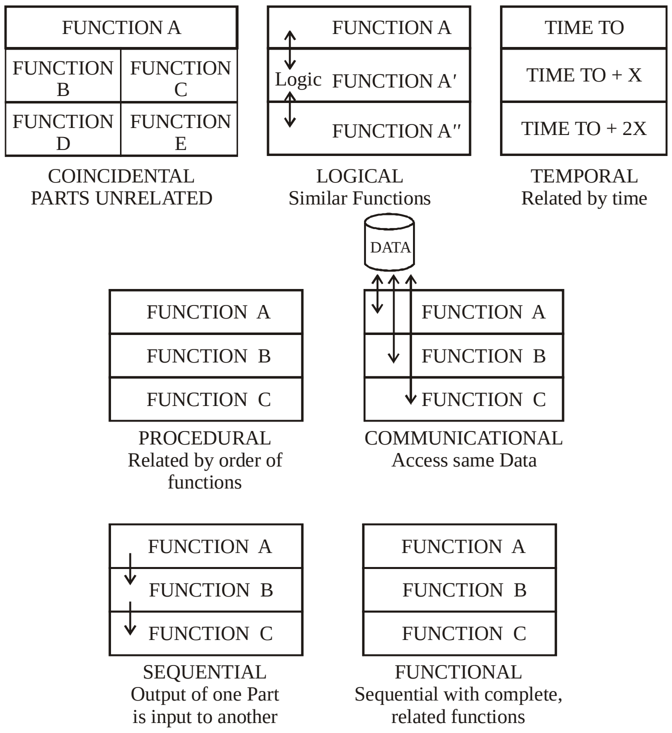


Fig. 5.8: Examples of Cohesion

6. **Sequential Cohesion:** When the output from one part of a component forms the input to the next part, we get sequential cohesion. Sequential cohesion does not provide any directions on how to combine them into modules. Different possibilities exist:

(a) Combine all in one module.
(b) Put the first half in one and the second half in another.
(c) The first third in one and the rest in the other and so on sequentially cohesive modules bear a close resemblance to the problem structure however they are still considered to be for from the ideal; which is function cohesion.

7. **Functional Cohesion:** Functional cohesion is the strongest cohesion. In this type of cohesion, every processing element is necessary for the performance of a single function and all the desired elements are contained in one component. A functionally cohesive component performs only and only that function for which it is designed and nothing else.

A cohesive object is one in which a single entity is represented and all the operations on that entity are included with the object. Such a cohesion called object cohesion is a new class of cohesion in which each operation provides functionality which permits the characteristics of the object to be modified, inspected or used as a basis for service provision.

5.7.2 **Coupling**

Coupling is the degree to which each program module relies on each one of the other modules. Coupling is a property of a collection of modules. It is an indication of the strength of the inter connections between the components in a design. Highly coupled systems have strong inter-connections with program units dependent on each other. Loosely coupled systems on the other hand are made up of components which are independent or almost independent. Loose coupling is desirable for good software engineering but tight coupling may be necessary for maximum performance. Uncoupled components have no inter connections at all coupling depends on several things:

(i) The references made from one component to another.
(ii) The amount of data passed from one component to another.
(iii) The amount of control one component has over the other.
(iv) The degree of complexity in the interface between components.

The following diagram showing component coupling explains everything. Dependency between modules has been shown by taking four modules A, B, C and D.

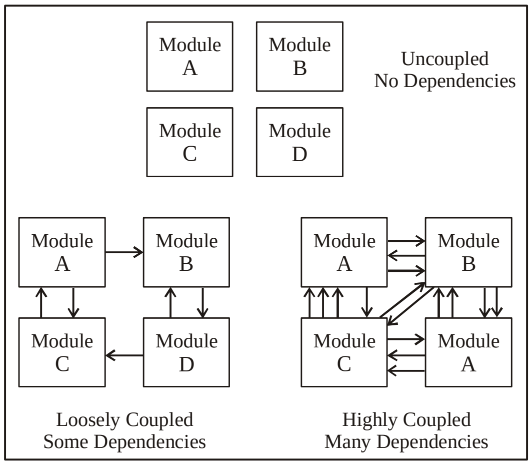


Fig. 5.9: Coupling

Coupling can be measured from complete dependence to complete independence. Actually a system can not be built of completely uncoupled components. When coupling is high, large parts of the system may be perturbed by the change. Low coupling helps in minimizing the number of components requiring revision.

5.7.2.1 **Types of Coupling**

Some types of coupling, in order of highest to lowest coupling are as follows: 1. Content coupling, 2. Common coupling, 3. External coupling, 4. control coupling, 5. Stamp coupling, 6. Data coupling message coupling

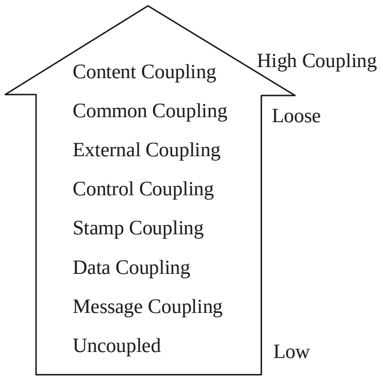


Fig. 5.10: The Range of Coupling Measures

1. **Data Coupling:** Two modules are data coupled, if they communicate by using an elementary data item that is passed as a parameter between the two, e.g. an integer, a float, a character etc. Data coupling is simpler and leaves less room for error. It is most convenient to trace data and to make changes. If coupling must exist between components data coupling is the most desirable.
2. **Message Coupling:** This is the loosest type of coupling. Modules do not depend on each other; instead they use a public interface to exchange parameter less messages.
3. **Stamp Coupling:** Stamp coupling is when modules share a composite data structure and use only a part of it, possibly a different part (e.g. whole record is passed to a function which needs one field of it).
4. **Control Coupling:** Two modules are control coupled if they pass a piece of information intended to control the internal logic using at least one 'control flag'. The control flag is a variable that controls decisions in subordinate or superior modules. The advantage of control coupling is that each component controls only one function or executes one process. This restriction is necessary to minimize the amount of controlling information which must be passed from one component to another and localizes control to a fixed and identifiable set of parameters forming a well-defined interface.
5. **External Coupling:** External coupling occurs when two modules share an externally forced data format, communication protocol or device interface.
6. **Common Coupling:** Two modules are common coupled if they both are the same global data area. Changing the shared resource implies changing all the modules using it. The amount of coupling can be reduced by organizing the design so that data are accessible from a common data store but dependence is skill there because making a change to the common data means tracing back to all components that access the data to calculate the effect of that change. This kind of coupling is known as common coupling. With common coupling it becomes difficult to find which component is responsible for having set a variable to a particular value.

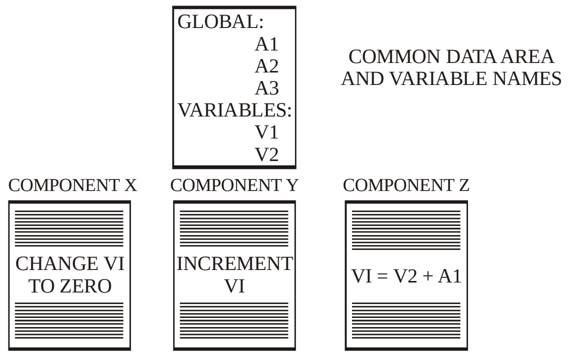


Fig. 5.11: Example of Common Coupling

7. **Content Coupling:** Content coupling exists between two modules if these module changes a statement in another module, one module references or changes data contained inside another module, or one module branches into another module. One module actually modifies the procedural contents of another module. In content coupling, modules are inter dependent on each other.

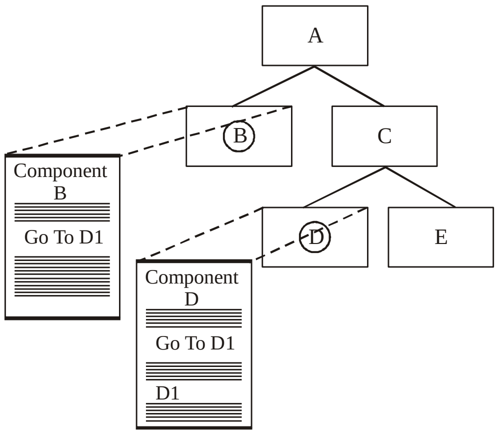


Fig. 5.12: Example of Content Coupling

5.8 DATA DESIGN

Data design is the first of all the activities in a software design process. It leads to a better program structure, effective modularity and decreased complexity. Data Design is developed by the transformation of the Data dictionary and entity relationship diagram into Data Structure which are necessary to implement the software. The whole process includes identifying the data defining specific data types and storage mechanisms and guarantees data integrity by using business rules and other mechanisms.

Data design creates a model of data information that is represented at a high level of abstraction i.e. the customer or user's views of data. The data model is refined into progressively more implementation specific representations which can be easily processed by the computer based system. In many software applications, the architecture of the data will put great influence on the architecture of the software that must process it.

The selection process may take the help of algorithmic analysis of alternative structures in order to determine the most efficient design or it can involve the use of a set of modules that provides operations on some representation of objects. Some principles are observed while specifying the data are given below:

1. The Systematic analysis principle applied to function and behavior should also be applied to data. Lot of June and energy is spent on deriving, reviewing and specifying functional requirements and preliminary design. In addition to it representations of data flow and content should be developed and reviewed; data objects should be identified, alternative data organizations should be considered and the impact of data modeling on software design should be evaluated.
2. All data structures and the operation to be performed on each should be identified. The design of an efficient data structure depends on operations to be performed on data structure upon evaluation of the operations; an abstract data type is defined for use in subsequent software design which simplifies software design considerably.
3. A data dictionary should be established and used to define both data and program design. The specific relationship among data objects and the constraints on the elements of a date structure can be more easily defined by dictionary like data specifications.
4. Low level data design decisions should be deferred until late in the design process. Overall data organization can be defined during requirements analysis, refined during data design work and specified fully during component level design.\
5. The representation of the data structure should be known only to those modules that must make direct use of the data contained within the structure. The quality of a software design is studied by the concept of information hiding and concept of coupling.
6. A library of useful data structures and the operations that may be applied to them should be developed which can reduce both specification and design effort for data.

7. A software design and programming language should support the specification and realization of abstract data types.

There are three levels to view the structure of data:

1. Program component level 2. Application Level 3. Business Level

At the program component level, design of data structures and the algorithms needed to manipulate them are essential for the development of a high quality design.

At the application level, the translation of a data model into a data base is necessary to get the specified business objectives. At the business level, the collection of information stored in different data bases should be reorganized into data ware house which enables data mining that has an effect on the business.

5.9 ARCHITECTURAL DESIGN

An architectural design acts as a preliminary “blue print” from which software is developed. Architecture describes the software top-level structure which identifies its components. IEEE architectural design as ‘the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.’

First of all software requirement is examined to establish a framework and then building a physical model using recognized software engineering methods. Shaw and Garlan (1996) suggest that software architecture is also the first step in producing a software design levels; architecture, code design and executable design.

It is very useful to work from the top-down designing an architecture, then the code design and finally the executable design but Krasner, Curtis and Iscoe (1987) have studied the habit of developers on 19 projects; they present a report and on the basis of other evidences confirm that actually the designers more back and forth from one level to other as they get extra knowledge about the solution and its implication. As the designers explore other aspects of the system, the designers may interact with testers or programmers bringing changes in the design to step up implementation, testability or maintainability.

In Architectural design performs the following functions:

1. The software designers can specify the system behavior (function and performance) after the architectural design provides a level of abstraction.
2. It is really the consciences for a system, a good architectural design guides the process of system enhancement indicating what changes can be made without compromising the integrity of the system.
3. It evaluates all top level designs.
4. It develops and documents top level design for the external and internal interfaces.
5. It develops preliminary versions of user documentation.
6. It defines and documents preliminary test requirements and the schedule for software integration.

5.10 PROCEDURAL DESIGN (FUNCTION ORIENTED DESIGN)

Procedural design techniques are very common and popular and at present they are being used in many software organizations. It is the result of focusing full orientation to the function of the program. This is an approach to software design where the design is decomposed into a set of interacting units where each unit of module performs a specific function Niklans Wirth who is the creator of Pascal and a number of other languages are one of the best known advocates of this method. His special variety to known as step-wise refinement. This stepwise refinement is a top-down strategy where a program is refined as a hierarchy of increasing levels of detail (Mills 1988). Refinement is actually a process of elaboration.

Procedural design is also named as component level design. It occurs after data, architectural and inter face designs have been developed. The aim is to translate the design model into an operational software. The translation from high level of abstraction of existing design into low level of abstraction of operational program is risky because if can bring in subtle errors that are difficult to locate and correct in later shapes of the software process. During this translation, certain design principles must be followed. Regardless of the methodology adopted to represent the procedural design, the date structures, interfaces and algorithms defined should follow to a variety of well-established procedural guiding principles to avoid errors as the procedural design is developed.

1. **Structured Programming:** In the late 1960s, Dijkstra and others gave the idea of using a set of constructs from which any program could be formed. Each construct had a predictable logical structure which was entered at the top and exited at the bottom for the convenience of the reader to follow procedural flow move easily. These constructs are Sequence, Condition and Repetition. The sequence implements processing steps, condition provides facilities for selected processing and Repetition allows for looping. These structured constructs are meant to limit the Procedural Design to a small number of pre-determined operations this contributes more to a human understanding the process called chunking by the psychologists. Any program needs three constructs only for designing and implementation.

5.11 **STRUCTURED DESIGN METHODOLOGY (SDM)**

Structured Design Methodology helps the designer during process. Software is a transformation function that converts given inputs to desired outputs. The main objective of this strategy is to specify functional modules and connections. SDM controls and puts an influence on the final design so that the programs implementing design would have a nice hierarchical structure that has functionally cohesive modules and few inter connections between modules as for as possible. SDM aims to achieve high cohesion but low coupling.

Steps: SDM follows the following procedure:

1. **Restate the problem as Data Flow Diagram (DFD):** The Designer finds full freedom in creating a DFD because the system does not exist yet. DFD will solve the problems stated in SRS.

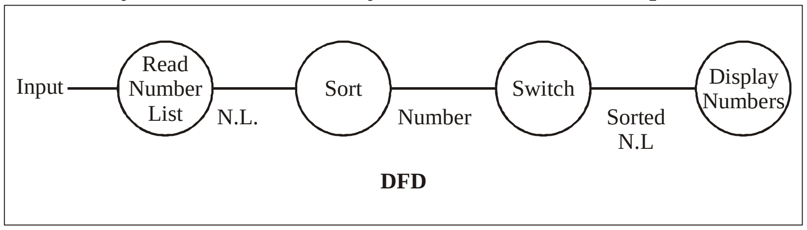


Fig. 5.13: Identification of DFD

Let us take the example of simple sort program that sort the ‘n’ numbers in ascending order. The problem has only one input file i.e. Read Number List. If we want to transform the input to the desired output, all the numbers are stored in a list. Then all numbers list goes to the sort transformation. If numbers are not sorted, Switch Transform changes the place order of number. Sorted Number are then passed to display transformation for printing numbers in ascending order on the screen.

2. **Identify most Abstract Input (MAI) and Most Abstract Out (MAO) data elements:** White going from outputs to inputs, the output of the major transformation is known as MAO and the input of major transformation is known as MAI. In this example of sort program, Number is MAI to switch transform and sorted N.L. is MAC.

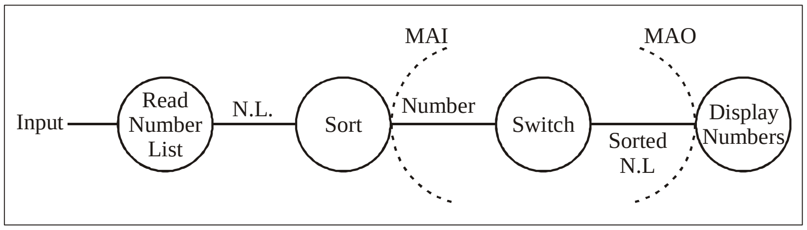
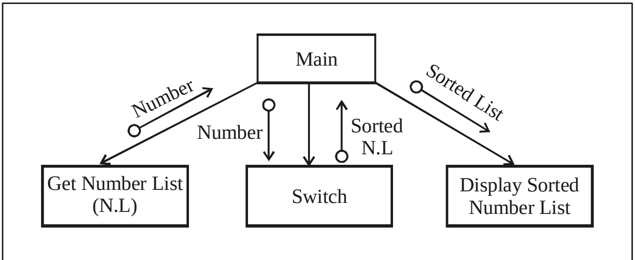


Fig. 5.14: Identification of MAI & MAO

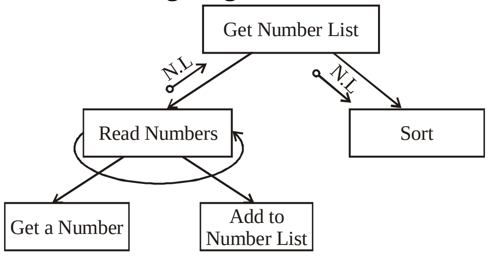
3. **First Level Factoring:** After the identification of central transformation MAI and MAO; the next step is the analysis of these information's. Main Module is specified first of all whose aim is to invoke the subordinates. For each MAI data item, an immediate subordinate module to the main module is specified. Each module is an input module which aims to provide to the main module the most abstract data item for which it is created. Same procedure is adopted for MAO data item, a subordinate module that is an output module that accepts data from the main module is specified.



First Level Factoring

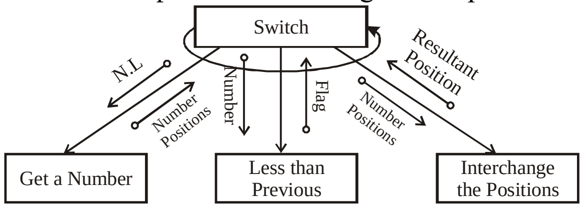
4. **Factoring the input, output and transform branches:** Each module in DFD is factored to move details:

(a) **Factoring the input Module:** The objective of input module is to produce some data to the main module. Do factor as input module, the transform in the DFD that produces number list is treated as a central transform as is shown in the following diagram.



(b) **Factoring the output Module:** It resembles the factoring of input module. The aim is to reach the final output. During the factoring of output module, usually there will be no input module because this task is performed by MAI.

(c) **Factoring the Central Module:** To factor a central module into its subordinate modules; there are no standard guidelines available. It depends on the designer's experience:



Graphical Design Notation

In this a flow chart is quite simple in the form of pictures.

A box indicates processing steps, a diamond represents a logical condition and arrows show the flow of control. Condition (if then else) is represented by decision diamond – if true – it causes 'then part processing' and if false – it invokes 'else part processing'.

Repetition is depicted by two slightly different forms. The do-while tests the condition and executes a loop task again and again as long as the condition holds true. A repeat until executes the loop task first and then tests a condition and performs the task again until the condition fails. The selection construct is the extension if then else successive decisions are taken to test a parameter until a true condition occurs and a case part processing path is executed. The structured constructs may be nested within one another. By nesting constructs in this way, a complex logical scheme may be developed.

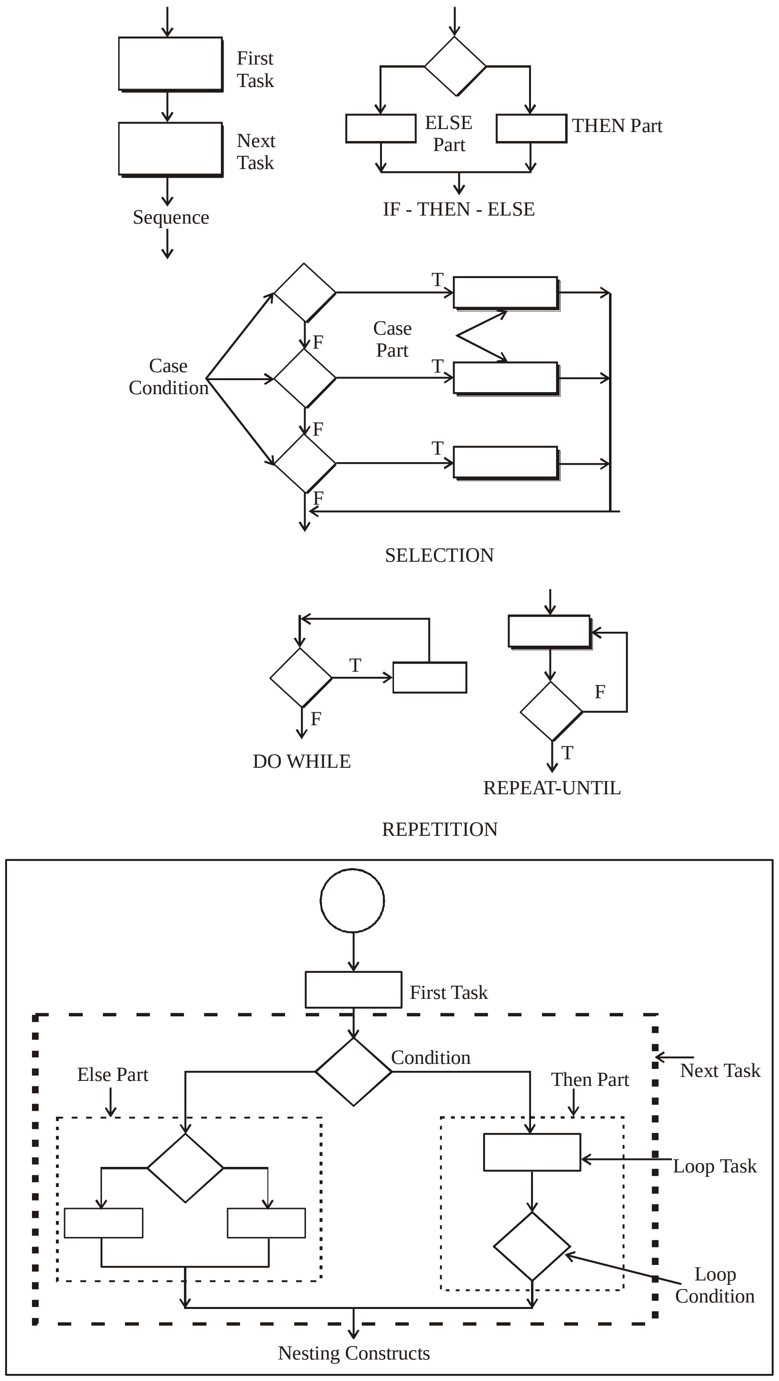


Fig. 5.15: Nesting of Different Structured Constructs

Tabular Design Notation: A module may be required to evaluate a complex combination of conditions and select proper actions based on these conditions. Decision tables provide criteria that translate actions and conditions in a tabular form. The table is not easy to misinterpret and if is possible to be used as a machine readable input to a table driven algorithm. Decision table is divided into four sections. As is clear from the following diagram; the upper left quadrant contains list of all conditions and the lower left hand quadrant contains a list of all actions that are possible depending on combinations of conditions. The right hand quadrants indicate condition combinations and the related actions that will happen for a specific combination.

CONDITIONS	1	2	3	4					n
Conditions # 1	✓			✓	✓				
Conditions # 2		✓		✓					
Conditions # 3			✓		✓				
ACTIONS									
Action # 1	✓			✓	✓				
Action # 2		✓		✓					
Action # 3			✓	✓					
Action # 4			✓	✓	✓				

Fig. 5.16: Decision Table Nomenclature

3. **Program Design Language (PDL):** PDL is also known as structured English or pseudo code. This is a language that uses the vocabulary of a natural language (English) and the overall syntax of a programming language. IEE defines PDL as a specification language with special const and sometimes with special constructs and sometimes verification protocols used to develop, analyze and document a program design. A team of designers and programmers use PDL as a way to ensure that actual programming is likely to match design requirements PDL is mainly used during the detailed design phase and uses to describe algorithms and specify interfaces between modules P&L enables the programmer to concentrate on the algorithms without worrying about all the syntactic details of a particular programming language.

Advantages of using PDL

- It uses English like statements that describe specific operator very precisely.
- It supports the idea of iterative refinement.
- It is expressed in easy-to-read format.
- Reviews became very easy because the source code is no examined.
- The language uses flow chart replacements, program documentation and technical communication at all level.
- Continuous use and refinement of the PDL has established if as the medium of choice for either creating refining a detailed program.

5.12 **DESIGN VERIFICATION**

Software design verification or review is necessary to review or check the design, implementation way, testing and maintenance, members of Development team. These reviews are well documented, comprehensive and systematic evaluation of a design to ensure the adequacy of the design requirements. It is done to evaluate the capability of the design and to identity problems IEEE defines if as a formal meeting at which a system's preliminary or detailed design is presented to the user customer or other parties for comment and approval.

The main aim behind the design review is to detect the errors in the design process which will reflect in code and also in the final system. These verifications are done to check if the design is of good quality or not. These reviews are taken at the end of the design phase to find solutions and resolve issues regarding software related design decisions, architectural design and detailed design. The reviews include study of the development plans, requirements and design specifications, testing plans and procedures, verification results for each stage of the design.

Types of Reviews: The review step generally involves three steps:

1. Preliminary Design Review
2. Critical Design Review

3. Program Design Review

1. **Preliminary Design Review:** It is a formal inspection of the high level architectural design to satisfy whether the design satisfies the functional as well as non-functional requirements of the customers and users. The review team includes customers, moderator, secretary, system designers, other state holders not involved in the project. The main aim is to:

(i) To specify whether effective modularity is achieved or not.
(ii) To define interfaces for modules and external system elements.
(iii) To ensure maintain ability has been considered or not.
(iv) To assess quality factors.
(v) To check discrepancies if any and to resolve it by the review team.

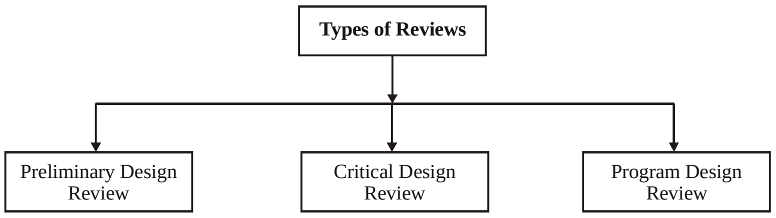


Fig. 5.17: Types of Reviews

2. **Critical Design Review:** Critical Design Review is taken after the successful completion of the preliminary design review. After the customer is satisfied, a critical design review is conducted with the following purpose in mind:

(i) To ensure that designs (conceptual and technical) are without defects.
(ii) To determine whether the design satisfies the requirements specifications or not.
(iii) To assess the functionality and maturity of the final design.
(iv) To justify that the design has clarity, effectiveness and easy to understand. A review is formed to check the design specifications using diagrams and data. Other individuals in addition to the members of the review team are:

(a) System tester: A technical expert.
(b) Analyst: Responsible for writing system documentation.
(c) Program Designer: Who understands the design in detail.

If a minor fault is found, if is resolved by the review team. In case of a major fault, the review team may take decision to revise the proposed technical design.

3. **Program Design Review:** The designers and the developers conduct this review after the completion of Critical Design Review. This is done to get feedback on the designs before implementation begins. The main purpose is to:

(i) Ensure the feasibility of the design.
(ii) Ensure that the interface is consistent with the architectural design.
(iii) Specify whether the design is amenable to implementation language.
(iv) Ensure that structured programming constructs are used throughout Software Design Process.
(v) Ensure that members of the implementation are able to understand the proposed design.

The team of Program Design Review includes program designers, developers, a system tester, moderator, secretary and analyst. A successful Program Design Review presents all plans related to coding plans before coding begins.

BACHELOR OF COMPUTER APPLICATIONS (BCA)
BCA-4-01T: SOFTWARE ENGINEERING

UNIT 6: CODING

6.1 INTRODUCTION

The software coding deals with design of data structure, information hiding, implementation. It is that phase of software development where actual implementation is carried out. The coding is done in a team manner which requires standardization of coding style, coding guidelines and coding documentations. The information hiding is another important aspect which is handled in this phase. In this chapter, we will go though the different coding styles and guidelines.

6.2 CODING

1. After the completion of design phase, coding is undertaken. The objective of the coding phase is to transform the design of a system into high level language code and then to perform unit testing at this code.
2. For adequate and effective coding, programmers have to mandatorily follow the coding standards. Before the testing phase, coding standards should be verified.
3. Coding guidelines provide some general suggestions regarding the coding style to be followed.
4. After the coding of a module, code reviews are carried out. This code review ensures that coding standards are followed during coding of a module. This code review also detects as many errors possible before testing.
5. It is important that the code should be written in such a manner that it could be modified later, if the need be.

6.3 CODING STANDARDS

A coding standard is a list of several rules to be followed during coding like variables should be named, name of module, exception handling mechanism etc. Many organizations develop their own coding standards and guidelines according to their need and type of software product they are going to develop. The some commonly known coding standards are as follows:

1. The organizations should follow the exact format of the header information which is as follows:

- Module’s Name
- Author’s Name
- Module’s Creation Date
- Module’s Synopsis
- Input/output parameters with different functions

2. There are some rules which limits the use of global. These rules explains what type of data can be declared global and what type of data cannot declared global.
3. The method of reporting error conditions and exception conditions should be standardized within an organization.
4. Global variable's name should always start with a capital letter whereas local variable's name should always start with small letter.

6.4 CODING GUIDELINES

Coding guidelines provides some general suggestions regarding the coding style to be followed. Some commonly known coding guidelines are as follows:

1. The function should be small not lengthy because length function is difficult to understand and arises large number of bugs.
2. goto statements should be avoided because it makes program unstructured.
3. The code should be well documented. It is only possible by using comment lines which explain it appropriately.
4. The code should be easy to understand. The complex coding is difficult to understand and maintain.
5. Avoid to use same identifies for multiple purposes. Same identifier for multiple purposes makes future enhancements very difficult and lead to confusion.
6. Variable names should be clear and simple. Variable names should be mnemonic procedure and function name should also be mnemonic.

6.5 PRINCIPLES OF PROGRAMMING/CODING

Programming is the process of writing, testing, debugging and maintaining of a computer program. The following are some basic principles of good programming practice.

1. Before start the programming, developer should understand the problem. He should choose a programming language which meets the needs of the software to be built and the environment in which it will operate.
2. Developer should create a visual layout that aids code understanding.
3. Avoid repetition in programming.
4. Coding should be simple because it takes less time to write, easier to modify and has fewer bugs.
5. Coding should improves code reliability and decrease development time
6. Developer should create a set of unit tests that will be applied once the coding is complete.Perform unit tests and correct the uncovered errors.
7. Developer should conduct a code walkthrough and refactor the code.
8. Developer should write the code for maintainer because any code that is worth writing is worth maintaining in the future, either by him(developer) or by someone else.

6.6 CODE REVIEW/VERIFICATION

1. After the coding of a module, code reviews are carried out. This code review ensures that the coding standards are followed during coding of a module. This code review also detects as many errors possible before testing.
2. After the successful compilation of module, code review takes place. Code review is a cost effective strategy which produce high quality code by eliminating coding errors.
3. Eliminating an error from code involves three main activities—testing, debugging and then correcting errors. Debugging is very laborious and time consuming activity than testing.

4. There are four types of code review which are as follows:

(i) Code Walk through

(ii) Code Inspection

(iii) Clean Room Testing

(iv) Static Analysis

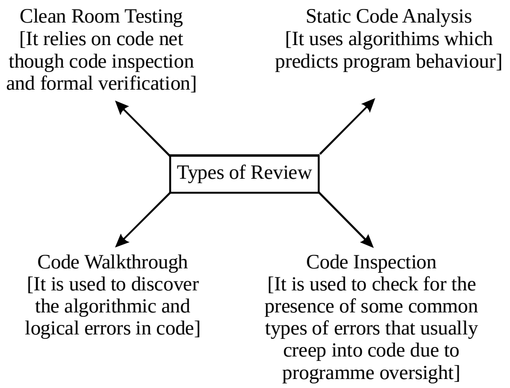

Clean Room Testing
[It relies on code net though code inspection and formal verification]
Static Code Analysis
[It uses algorithims which predicts program behaviour]
Types of Review
Code Walkthrough
[It is used to discover the algorithmic and logical errors in code]
Code Inspection
[It is used to check for the presence of some common types of errors that usually creep into code due to programme oversight]

Fig. 6.1 Types of Code Review/Verification

1. **Code Walkthrough**

- Code walkthrough is an informal code analysis technique.
- Its aim is to discover the logical and algorithmic errors in the code.
- It does not tell how to fix the discovered errors.
- Before the code walkthrough meeting, the members note down their findings to discuss.
- The success of code walkthrough are based on personal experience, common sense and several subjective factors.
- The walkthrough team should consist of three to seven members. The walkthrough team should not be too big or too small.

2. **Code Inspection**

- Code inspection is a formal code analysis technique.
- The aim of the code inspection is to discover common types of errors which are caused due to improper programming.
- Code inspection detects the commonly made errors along with the coding standards.
- It is also used to check whether coding standards are followed during the coding or not.
- Code inspection checks some programming errors like incompatible assignments, improper storage allocation and de-allocation, improper modification of loop variables, non-terminating loops etc.

3. **Clean Room Testing**

- It relies on code walkthrough code inspection and found verification.
- The aim of clean room testing is to produce documentation and code which is more maintainable and reliable.
- Clean room testing involves walkthroughs, inspection and formal verification which is time consuming and hence testing effort is increased.

4. **Static Analysis/Static Code Analysis**

- It is also known as automated code review.
- Static code analysis is Excellent for enforcing compliance to standards
- It Identifies bugs in code and the design and implementation problems
- It uses algorithms which predict program behaviour.

- Static analysis explain the purpose of possible executions of a program. It gives assurance about any execution but this tool spend a lot of effort dealing with developer.
- Static analysis can be performed on modules or unfinished code

BACHELOR OF COMPUTER APPLICATIONS (BCA)
BCA-4-01T: SOFTWARE ENGINEERING

UNIT 7: SOFTWARE TESTING

7.1 INTRODUCTION

Software testing is an activity that has significant impact on the software development process. It is carefully designed process and must be performed effectively. Since allocated resource is limited and time is another constraint, so it is not possible to test any system with all possible inputs. It is the primary quality-control measure used during software development. The testing provides methods and techniques to test a system in reasonable time with assurance to give quality performance. The testing is broadly categorized as Black Box testing and White Testing. There is another way to cauterize the testing method called level testing discussed in detail in this chapter. The chapter includes detail discussion on testing along with advantages and limitations.

7.2 SOFTWARE TESTING

1. Testing is an important and critical activity of software development life cycle. It is the primary quality control measurement activity used during software development.
2. Testing is a set of systematic activities that can be planned in advance and conducted with the intent of finding errors. It make us feel confident that the software works well**.**
3. The goal of testing is to uncover requirement, design, and coding errors in the program. It is critical, expensive and time-consuming activity.

4. The testing process requires proper planning. The testing cannot show the absence of defects. It can show only software errors present.
5. During the testing phase, emphasis should be on the following:
    - Tests should be planned long before testing begins.
    - All tests should be traceable to customer requirements.
    - Tracing should begin “in the small” and progress toward testing “in the large.”
    - Testing must be conducted by the independent third parties.

7.2.1 **Objectives of Software Testing**

The testing objective is to test the code with aim to discover errors. It also expresses that the software functions are working according to software requirements specification with regard to functionality, features, facilities and performance. It is here noticeable, that testing will detect errors in the written code but it will not give you an idea about an error if the code does not address a specific requirement set in the SRS.

The primary objectives of testing are:

1. Testing is a process of executing a program with the intent of finding an error.
2. A good test case is one that has a high probability of finding the undiscovered error.
3. A successful test is one that uncovers an as-yet-undiscovered error.
4. It makes us feel confident that the software works well.

7.2.2 **Principles of Software Testing**

The testing principle is a guide book which helps software engineer to design effective test cases. There are many principles that a software engineer must understand before starting the testing process.

The following are the main principles of testing:

Sr. No.	Principles	Description
1	**All tests should be traceable to customer requirements**	It states that any defects that might cause the program or system to fail to meet the client’s requirements must be uncovered.
2	**The Pareto principle applies to software testing.**	It is also known as 80/20 principle. It states that 20% of the problems lead to 80% of other problems. One should concentrate to 20% of the problems and 80% of the problems will automatically remove.
3	**Tests should be planned long before testing begins.**	According to principle testing phase should be started soon after the requirements model is completed. The detailed test cases can begin as soon as the design model is designed.
4	**Testing should begin “in the small” and progress toward testing “in the large.”**	It states that the first test may be focused on individual components. As testing progresses, focus shifts in an attempt to find errors in integrated components and later on full system.
5	**Testing should be conducted by an independent third party**	The principle states that the testing should be conducted by an independent third party. The software engineer who has developed the system is not the best person to conduct testing
6	**Exhaustive testing is not possible.**	It states that It is possible, however, to adequately cover program logic and to ensure that all conditions in the component-level design have been exercised for testing.

Table 7.1 Principles of Software Testing

7.3 BASIC CONCEPTS

In software testing, there are few terms which generally create confusion. These terms are explained as follows:

1. ***Errors:*** *An error is a discrepancy between the actual value of the output given by the software and the specified (correct) value of the output for that given input.*

Errors occur when any aspect of software product is incomplete or incorrect. It is a mistake committed by the development team during development process. Errors can be classified into following categories:

- ***Implementation Errors:*** These errors made in translating design specifications into source code.
- ***Requirement Errors:*** These errors are due to incomplete and incorrect statements of user requirements. User fails to specify functional and nonfunctional requirements.
- ***Design Errors:*** These are introduced by failure to translate the requirements into complete and correct structures.
- ***Syntax Error:*** A syntax error is a program statement that violates one or more rules of the language in which it is written.
- ***Logic Error:*** A logic error deals with incorrect data fields, out-of-range terms and invalid combinations.

2. ***Fault:*** *A fault is a condition that causes a system to fail in performing its required function. A fault is the basic reason for software malfunction.*

It is also commonly called a bug. Sometime, it is possible that even providing correct input system give wrong results which due to some bug in the system. When it fails then we say that the system has a fault or a bug and it needs repair.

3**.** ***Failure:*** *Failure is the inability of the software to perform a required function to its specification.*

In other words, when software goes ahead in processing without showing error or fault even though certain input and process specification are violated, then it is called a software failure.

4. ***Quality Assurance:*** Quality assurance is the planned and systematic approach. It ensures that the standards, processes and methods are established and followed throughout the software development. Quality assurance is a set of activities which ensure the implementation of processes, procedures and standards in context to verification of developed software and intended requirements.

5. ***Quality Control:*** Quality control is used to verify the quality of the output. Quality control is set of activities which ensure the verification of developed software with respect to documented (or not in some cases) requirements.

6. ***Testing:*** Testing is the sequence of activities which ensure the identification of bugs in the software.

Difference between Testing, Quality Assurance and Quality Control

Quality Assurance	Quality Control	Testing
Quality assurance focuses on processes and procedures rather than conducting actual testing on the system.	Quality control focuses on actual testing by executing software with intend to identify bug/defect through implementation of procedures and process.	Testing focuses on actual testing of the system or unit with suitable test
Quality assurance is process oriented activities.	Quality control is a product oriented activities.	Testing is product oriented activities.
Quality assurance preventive activities.	Quality control is a corrective process.	Testing is a preventive process
It is a subset of Software Test Life Cycle (STLC)	QC can be considered as the subset of Quality Assurance.	Testing is the subset of Quality Control.

Table 7.2 Difference between Testing, Quality Assurance and Quality Control

7. ***Test Oracles:*** The test oracle defines a method to understand the determined and expected behavior of a system in order to test the test case for checking abnormality in behavior. In other words, a

test oracle is a mechanism, different from the program itself, which can be used to check the correctness of the output of the program for the test cases.

The following diagram shows how test oracle is useful to determine the expected behavior of the system

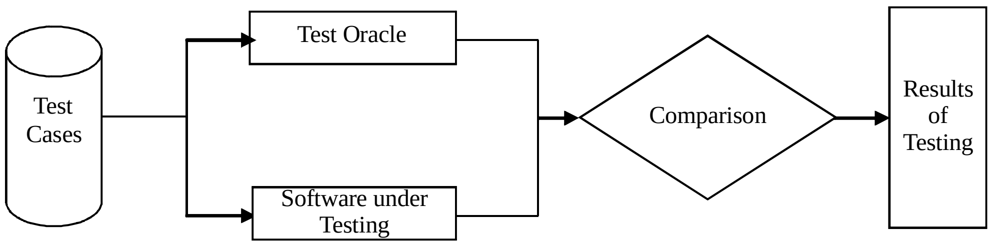


Fig 7.1 Working of Test Oracle

The above diagram shows testing process in which the test cases are given to the test oracle and the program under testing. The output of the two is compared to determine if the program behaved correctly for the test cases.

The human oracles generally use the specifications of the program to decide what the correct behavior of the system should be. To help the oracle to determine the correct behavior, it is important that the behavior of the system be unambiguously specified and the specification itself should be error free. Test oracles are human beings, so they may make mistakes when there is a discrepancy between the oracles and the results of a program. We need to verify the result produced by the oracle before declaring that there is a fault in the program.

8. ***Software Testability:***

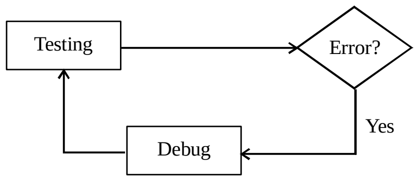
- Software testability is how easily a computer program can be tested.
- It is used to measure how adequately a particular set of tests will cover the product.
- The characteristics which leads to testable software are: operability, observability, decomposability, simplicity, stability and understandability.

9. ***Attributes of a Good Test***: The following are some attributes of good test:

- A good test has higher chances of finding errors.
- A good test is neither too simple nor too complex.
- A good test takes less time and resources.
- A good test is not redundant.

7.4 SOFTWARE DEBUGGING

Testing and debugging are different terms. Debugging is the process of locating and fixing errors in program code. Testing only determine the errors in the code. Testing does not fix those identified code. Whereas debugging is used to locate the errors and fix the located errors.



Difference between Testing and Debugging

The testing involves the identification of bugs in the program without correcting it. Normally people with a quality assurance background are involved in the identification of bugs. Testing is performed in the testing phase. The debugging involves fixing the bugs. The developers who coded the program conduct debugging.

Testing	Debugging
Testing is a process of finding and locating of a defect in a program	The debugging is process of fixing the defects in the program
Testing is done by the testing team	The debugging is done by the development team
Testing is done with intention to find as many defect as possible	The debugging is focused in removing the defects

Table 7.3 Difference between Testing and Debugging

7.5 PLANNING OF TEST

A test plan is a document consisting of different test cases designed for different testing objects and different testing attributes. The plan puts the tests in logical and sequential order per strategy chosen, top-down or bottom-up. The test plan is a matrix of test and test cases listed in order of its execution.

A test plan states:

- The items to be tested.
- At what level they will be tested at.
- The sequence they are to be tested in.
- How the test strategy will be applied to the testing of each item and the test environment.
- The software tools required and estimated hardware utilization.
- What will be the constraints which affect the testing process?

***Test Case*:** A test case is a set of instructions designed to discover a particular type of error or defect in the software system by inducing a failure. The goal of selected test cases is to ensure that there is no error in the program and if there is it then should be immediately depicted. Ideal test casement should contain all inputs to the program. This is often called exhaustive testing.

There are two criteria for the selection of test cases:

- Specifying a criterion for evaluating a set of test cases.
- Generating a set of test cases that satisfy a given criterion.

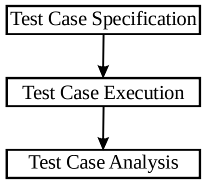


Fig. 7.2 Process of Test Case

1. **Test Case Specification:** Test plan focuses on approach; does not deal with details of testing a unit.Test case specification has to be done separately for each unit. Expected outcome also needs to be specified for each test case. Together the set of test cases should detect most of the defects. Would like the set of test cases to detect any defect, if it exists. The effectiveness and cost of testing depends on the set of test cases. Preparing test case specifications is challenging and time consuming. So for each testing, test case specifications are developed, reviewed, and executed.
2. **Test Case Execution:** Executing test cases may require drivers or stubs to be written; some tests can be automatic, others manual A separate test procedure document may be prepared
3. **Test Case Analysis:** Test summary report is often an output – gives a summary of test cases executed, effort, defects found, etc. the task of test case analysis are monitoring of testing effort is important to ensure that sufficient time is spent, computer time also is an indicator of how testing is proceeding, quality control focuses on removing defectsm and goal of defect prevention is to reduce the defect injection rate in future

10.6 SOFTWARE VERIFICATION AND VALIDATION

The validation and verification are two different functions in software engineering. We need to understand the difference between them.

Verification: All actions are taken at the end of software development phase to check whether we are building the product right. Verification is done at the end of each phase of software development cycle.

Validation: All actions are taken at the end of software development cycle to check whether we are building the right product. Validation is done at the end of complete development of the product.

These terms are often used interchangeably but following table shows how these terms are different:

Sr.No.	Verification	Validation
1	Verification is aimed to answer question like, Are we building the product right?	Validation is aimed to answer question like, Are we building the right product?
2	Verification ensures that the software system meets all the functionality.	Validation ensures that functionalities meet the intended behavior.
3	Verification takes place first and includes the checking for code, documentation etc.	Validation occurs after verification and mainly involves the checking of the overall product.
4	Verification is done by developers who developed that software.	Validation is done by expert team of testers.
5	Verification is a static activity as it includes the reviews, walkthroughs and inspections to verify that software is correct or not.	Validation has dynamic activities as it includes executing the software against the requirements.
6	It is an objective process and no subjective decision should be needed to verify the software.	It is a subjective process and involves subjective decisions on how well the software works.

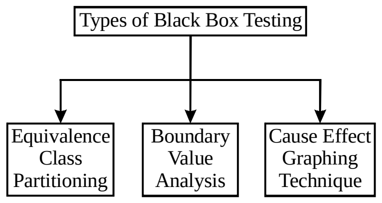
Table 7.4 Difference between verification and validation

7.7 BLACK BOX TESTING

1. Black box testing is also called behavioral testing, focuses on the functional requirements of the software. Black box testing enables the software engineer to derive sets of input conditions that will fully exercise all functional requirements for a program.
2. In black-box testing, the tester only knows the inputs that can be given to the system and what output the system should give. In other words, the basis for deciding test cases in functional testing is the requirements or specifications of the system or module. This form of testing is also called functional or behavioral testing.
3. Black-box testing is not an alternative to white-box techniques. It is acomplementary approach that is likely to uncover a different class of errors thanwhite-box methods.It is applied at the later stages of testing. Black-box testing identifies the following kinds of errors:

- Incorrect or missing functions.
- Interface missing or erroneous.
- Errors in data model.
- Errors in access to external data source.

Types of Black-box Testing:


Types of Black Box Testing
Equivalence Class Partitioning
Boundary Value Analysis
Cause Effect Graphing Technique

7. 3 Types of Black-box Testing

Equivalence class partitioning and boundary value analysis both are test case design strategies in black box testing.

1. ***Equivalence Class Partitioning:***

- The domain of the input values partitioned into set of equivalence class. According to equivalence class partitioning, testing the code with one value from an equivalence class is as good as testing the code with any other value from the same equivalence class.
- In this method the input domain data is divided into different equivalence data classes. It is used to reduce the total number of test cases to a finite set of test cases which covers maximum requirements.
- It is a process of taking all possible test cases and placing them into different classes. One test value is picked from each class while testing.
- Example: If we test for an input box which accepts numbers from 1 to 1000 then there is no use in writing thousand test cases for all 1000 valid input numbers and test cases for invalid data. Using equivalence partitioning method, we can divide our test cases into three equivalence classes of some valid and invalid inputs.

2. ***Boundary Value Analysis:***

- Boundary value analysis is a next part of equivalence class partitioning for designing test cases where test cases are selected at the edges of the equivalence classes.
- This technique is used to identify errors at boundaries rather than finding those exist in center of input domain.More application errors occur at the boundaries of input domain.
- It is often called stress and negative testing.
- Example: If we test for an input box which accepts numbers from 1 to 1000 then test cases are: one test case for exact boundary values of input domains each means 1 and 1000, one test case for just below boundary value of input domains each means 0 and 999, one test case for just above boundary values of input domains each means 2 and 1001.

3. ***Cause Effect Graphing Techniques:***

- The Cause-Effect graph technique restates the requirements specification in terms of logical relationship between the input and output conditions. The causes are the input conditions and effects are the results of those input conditions.
- This technique is used to recognize the root causes, the cause for an exact effect, problem, or outcome.
- In this technique, first we recognize the input conditions (causes) and actions (effect). Then we build up a cause-effect graph and convert cause-effect graph into a decision table. Finally we convert decision table rules to test cases.

Advantages of Black-box Testing

Sr. No	Advantages
1	The test is unbiased because the designer and the tester are independent of each other.
2	The tester does not need knowledge of any specific programming languages.
3	The test is done from the point-of-view of the user, not the designer.
4	Test cases can be designed as soon as the specifications are complete.
5	Well suited and efficient for large code segments.
6	Code Access not required.
7	Clearly separates user's perspective from the developer's perspective through visibly defined roles.
8	Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language or operating systems.

Table 7.5 Advantages of Black-box Testing

Disadvantages of Black-box Testing

Sr. No.	Disadvantage
1	Limited Coverage since only a selected number of test scenarios are actually performed.
2	Inefficient testing, due to the fact that the tester only has limited knowledge about an application.
3	Blind Coverage, since the tester cannot target specific code segments or error prone areas.
4	The test cases are difficult to design.

Table 7.6 Disadvantages of Black-box Testing

7.8 WHITE BOX TESTING

White-box testing is also known by other names, such as glass-box testing, structural testing, clear-box testing, open-box testing, logic-driven testing, and path-oriented testing. A complementary approach to functional or black-box testing is called structural or white-box testing. In this approach, test groups must have complete knowledge of the internal structure of the software. We can say structural testing is an approach to testing where the tests are derived from knowledge of the software's structure and implementation. Structural testing is usually applied to relatively small program units, such as subroutines, or the operations associated with an object. As the name implies, the tester can analyze the code and use knowledge about the structure of a component to derive test data. The analysis of the code can be used to find out how many test cases are needed to guarantee that all of the statements in the program are executed at least once during the testing process. It would not be advisable to release software that contains untested statements as the consequence might be disastrous.
The nature of software defects are:

Sr. No.	Nature of software defects in white box testing
1	Logical errors and incorrect assumptions are inversely proportional to the probability that a program path will be executed.
2	We often believe that a logical path is not to be executed when, in fact, it may be executed on a regular basis.
3	Typographical errors are random. When a program is translated into programming language source code, it is likely that some typing errors will occur.

Table 7.7 Nature of Software Defects in White Box Testing

Different forms of White Box Testing

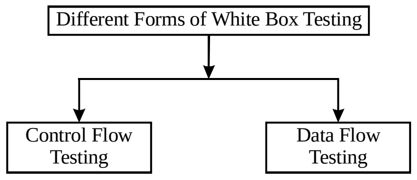


Fig. 7.4 Different forms of White Box Testing

- **Control Flow Testing:** Control flow testing is a structural testing strategy which uses the program's control flow. It is a directed graph where the nodes represent the processing statements like definition, computation and predicates while the edges represent the flow of control between processing statements. It is more effective for unstructured code rather than structured code. But it cannot catch initialization and specification mistakes.Toexamine the patterns, we need to construct a control flow graph of the code.
- **Data Flow Based Testing:** Data flow testing is a technique used to detect improper use of data in a program. It is a white box testing technique thatcan be used to detect improper use of data values due to coding errors. Data-flow testing uses the control flow graph to explore the unreasonable things that can happen to data. Data-flow testing is the set of test strategies based on selecting paths through the program's control flow in order to explore sequences of events related to the status of data objects. Data-

flow testing strategies span the gap between all paths and branch testing.Data-flow testing monitors the lifecycle of a piece ofdata and identifies potential bugs byexamining the patterns in which that piece of data is used.

Reasons Why White-box Testing is performed

Sr. No.	**Reasons**
1	All paths in a process are correctly operational.
2	All logical decisions are executed with true and false conditions.
3	All loops are executed with their limit values tested.

Table 7. 8 Reasons Why White-box Testing is Performed

Types of White Box Testing

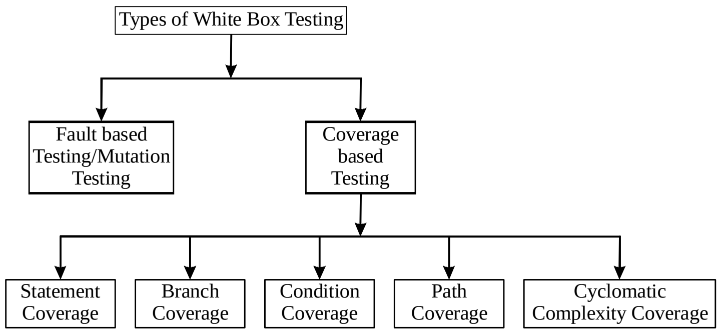

Types of White Box Testing
Fault based Testing/Mutation Testing
Coverage based Testing
Statement Coverage
Branch Coverage
Condition Coverage
Path Coverage
Cyclomatic Complexity Coverage

Fig. 7.5 Types of White Box Testing

1. **Fault Based Testing:** It is used to detect certain types of faults. Its goal is to determine the absence of faults rather than the goal of “finding errors” .The focus is on faults rather than errors. It is not a manual testing. The example of fault based testing is mutation testing.
2. **Coverage Based Testing:** A coverage based testingstrategy aims to execute certain program elements to discover failures. It is of four types:

- ***Statement Coverage:***It aims to design test cases so as to executeall the statements of a program. In this strategy, every statement in a program is to be executed at least once. The disadvantage of this strategy is that after executing a statement for one input value once, it assumes that it behave properly but there is no guarantee that it will behave correctly for all input values.
- ***Branch Coverage:***It is also known as edge testing. It is used to design test cases in such a way that it assume true and false value in return. Branch coverage testing is stronger than statement coverage testing because branch coverage ensures statement coverage but statement coverage does not ensure branch coverage.
- ***Condition Coverage:*** In this testing strategy, test cases are designed to make each component of a conditional expression to assume both true and false values. Condition coverage testing is a stronger testing strategy than branch coverage and Branch testing is a simplified condition testing.
- ***Path Coverage:***It design test cases in such a manner that all linearly independent paths in the program are executed at least once. To understand each linearly independent path, we need to understand the control flow graph (CFG). A control flow graph is a directed graph which consists of set of nodes and edges (N, E). It describes how the control flows through the program.

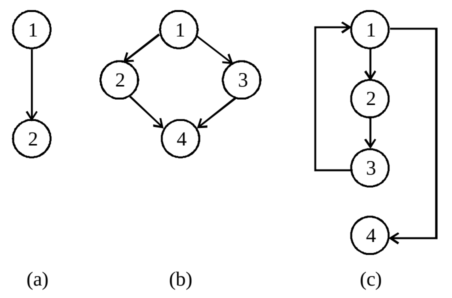


Fig. 7.6 Examples of Path Coverage

- ***Cyclomatic Complex Graph:*** *It tests the complexity of the program. It also estimates the amount of effort required to understand the code. The Cyclomatic complexity is defines as:*

  CC=E-N+P
  CC= Cyclomatic Complexity
  E= Number of Edges
  N= Number of Nodes
  P= Number of Connected Components
  Example:

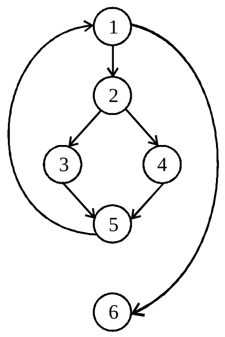


Fig. 7.7 Examples of Cyclomatic Complexity

If number of edges= 7 and number of nodes= 6, then cyclomatic complexity is:
CC= 7-6+2 => CC=3
Cyclomatic complexity helps in estimation of complexity of code, testing effort and program reliability.

Advantages of White-Box Testing:

Sr. No.	Advantages
1	As the tester has knowledge of the source code, it becomes very easy to find out which type of data can help in testing the application effectively.
2	It helps in optimizing the code.
3	Extra lines of code can be removed which can bring in hidden defects.
4	Due to the tester's knowledge about the code, maximum coverage is attained during test scenario writing.

Table 7.9 Advantages of White Box Testing

Disadvantages of White-Box Testing:

Sr. No.	Disadvantages
1	Due to the fact that a skilled tester is needed to perform white box testing, the costs are increased.
2	Sometimes it is impossible to look into every nook and corner to find out hidden errors that may create problems as many paths will go untested.
3	It is difficult to maintain white box testing as the use of specialized tools like code analyzers and debugging tools are required.

Table 7.10 Disadvantages of White Box Testing

7.9 GREY BOX TESTING

Grey Box testing is a technique to test the application with limited knowledge of the internal workings of an application. In software testing, the term "the more you know the better" carries a lot of weight when testing an application.

Mastering the domain of a system always gives the tester an edge over someone with limited domain knowledge. Unlike black box testing, where the tester only tests the application's user interface, in grey box testing, the tester has access to design documents and the database. Having this knowledge, the tester is able to better prepare test data and test scenarios when making the test plan.

Advantages:

1. Offers combined benefits of black box and white box testing wherever possible.
2. Grey box testers don't rely on the source code; instead they rely on interface definition and functional specifications.
3. Based on the limited information available, a grey box tester can design excellent test scenarios especially around communication protocols and data type handling.
4. The test is done from the point of view of the user and not the designer.

Disadvantages:

Since the access to source code is not available, the ability to go over the code and test coverage is limited. The tests can be redundant if the software designer has already run a test case. Testing every possible input stream is unrealistic because it would take an unreasonable amount of time; therefore, many program paths will go untested.

7.10 DIFFERENCE BETWEEN BLACK BOX TESTING, GREY BOX TESTING AND WHITE BOX TESTING

Sr. No.	Black Box Testing	Grey Box Testing	White Box Testing
1.	The Internal Workings of an application are not required to be known	Somewhat knowledge of the internal workings are known	Tester has full knowledge of the Internal workings of the application
2.	Also known as closed box testing, data driven testing and functional testing	Another term for grey box testing is translucent testing as the tester has limited knowledge of the insides of the application	Also known as clear box testing, structural testing or code based testing
3.	Performed by end users and also by testers and developers	Performed by end users and also by testers and developers	Normally done by testers and developers
4.	Testing is based on external expectations Internal behavior of the application is unknown	Testing is done on the basis of high level database diagrams and data flow diagrams	Internal workings are fully known and the tester can design test data accordingly
5.	This is the least time consuming and exhaustive	Partly time consuming and exhaustive	The most exhaustive and time consuming type of testing
6.	Not suited to algorithm testing	Not suited to algorithm testing	Suited for algorithm testing

Table 7.11 Difference Between Black Box Testing, Grey Box Testing and White Box Testing

7.11 LEVELS OF TESTING

There are four levels of testing, i.e., four individual modules in the entire software system.

1. Unit Testing
2. Integration Testing
3. System Testing
4. Acceptance testing

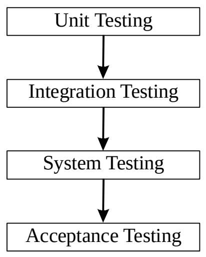


Fig 7. 8 Level of Testing

7.11.1 Unit Testing

The unit test is the lowest level of testing performed during software development, where individual units of software are tested in isolation from other parts of a program. The individual components are tested to ensure that they work correctly. It focuses on the smallest unit of software design where each component is tested independently without other system components. There are number of reasons to do unit testing rather than testing the entire product:

Reasons to perform Unit Testing:

- The size of a single unit is small that is easy to locate error in it.
- The unit is small enough that we can attempt as much as test cases.
- The interactions of multiple errors in it are eliminated.

Benefits of Unit testing:

The unit testing has the following benefits:

1. **Increases confidence:** The unit testing increases confidence in changing/maintaining code. If good unit tests are written and if they are run every time any code is changed, the likelihood of any defects due to the change being promptly caught is very high. If unit testing is not in place, the most one can do is hope for the best and wait till the test results at higher levels of testing are out. Also, if codes are already made less interdependent to make unit testing possible, the unintended impact of changes to any code is less.
2. **Reusability:** The codes are more reusable. In order to make unit testing possible, codes need to be modular.
3. **Fastest development:**The development is faster as the effort required to find and fix defects found during unit testing is easy as comparison to finding defects during system testing or acceptance testing.
4. **Fewer efforts:**The cost of fixing a defect detected during unit testing is lesser in comparison to that of defects detected at higher levels.
5. **Easy Debugging:**Due to unit testing the debugging is easy. When a test fails, only the latest changes need to be debugged.
6. **Error discovery:**Unit testing can discover error like Comparison of different data types, Incorrect logical operators or precedence, Expectation of equality when precision error makes equality unlikely, Incorrect comparison of variables, Improper loop termination, Failure to exit when divergent iteration is encountered, Improperly modified loop variables

Unit-test Procedure:

In unit testing, a module is tested under well-defined environment which includes all relevant code that is required to execute the module. The following diagram shows environment:

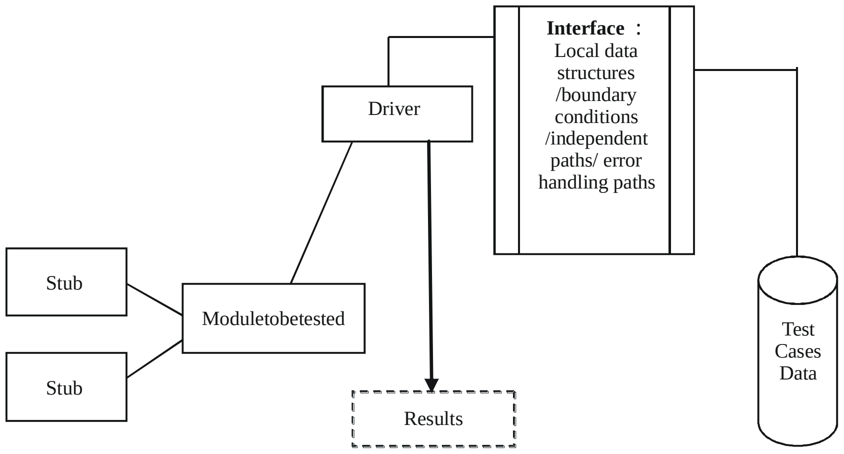

Interface :
Local data structures /boundary conditions /independent paths/ error handling paths
Driver
Stub
Moduletobetested
Stub
Results
Test Cases Data

Fig. 7.9 Unit-test Environment

In the unit test environment, there are four components namely stub, driver, interface, test cases.

Stubs: The Stubs serve to replace modules that are subordinate to the component to be tested.

Driver: A driver is nothing more than a "main program" that accepts test-case data, passes such data to the component (to be tested), and prints relevant results.

Interface: The interface manages takes like integration of local data structure, keep check on boundary condition and possible path to execute the code. The local data structure is examined to ensure that data stored temporarily maintains its integrity during all steps in an algorithm's execution. Boundary conditions are tested to ensure that the module operates properly at the boundaries established to limit. All independent paths through the control structure are exercised to ensure that all statements in a module have been executed at least once.

Test case data: The test case data is collection of all possible input that the can be observed during the testing phase of a particular unit.

7.11.2 **Integration Testing**

The second level of testing is called integration testing. Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing. In this testing many unit-tested modules are combined into subsystems, which are then tested. The goal here is to see if the modules can be integrated properly.

Objective of Integration Testing:

The primary objective of integration testing is to test the module interfaces in order to ensure that there are no errors in the parameter passing, when one module invokes another module. During integration testing, different modules of a system are integrated in a planned manner using an integration plan. The integration plan specifies the steps and the order in which modules are combined to realize the full system. After each integration step, the partially integrated system is tested.

Approaches to Integration Testing:

The various approaches used for integration testing are:

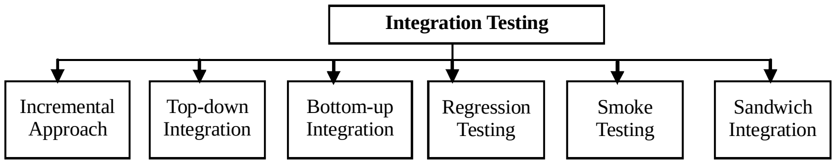


Fig. 7.10 Approaches to integration testing

- **Incremental Approach:** According to the incremental approach, testing starts with first combine only two components together and performs testing on them. If there are some error then removes them first, otherwise combine another component to it and then test again, and so on until the whole system is developed. The following logical diagram shows the same process:

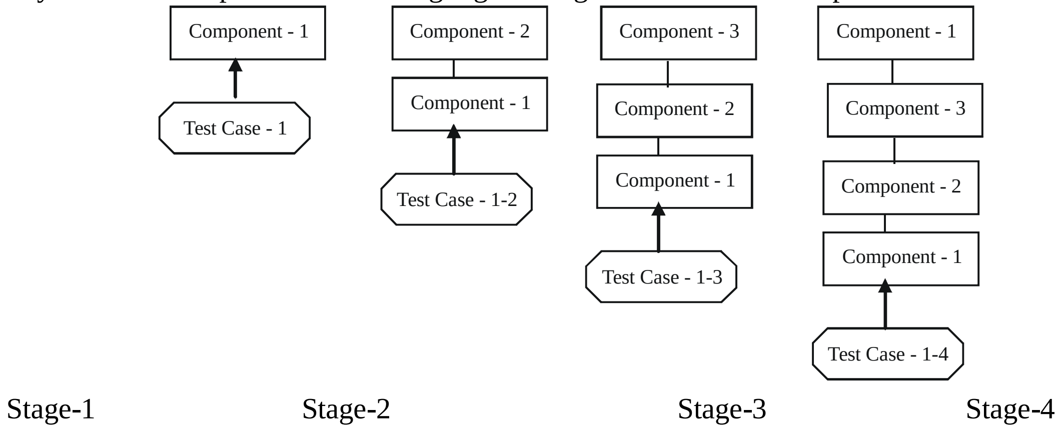


Fig. 7.11 Incremental approach for testing

According to Figure, in incremented testing is performed in four stages where in stage1, component1 is tested if it is working correctly then it is integrated with componets2 and gain test cases1 to 2 are used for testing. If these components are corrected or error free then module componets3 is integrated, i.e. stage3 and then tests case 1 to 3 applied on all the components. The process is repeated till whole system is not integrated and tested.

- **Top-Down Integration Testing**

Top-down integration testing is an incremental approach to construction of program structures. Modules are integrated by moving downward through the control hierarchy beginning with the main control module.

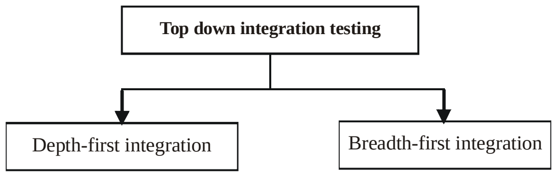
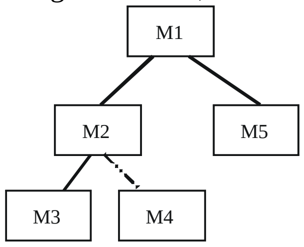


Fig. 7.12 Types of Top down Integration Testing

Depth-first integration: According to depth first integration, we integrate all components on a major control path of the structure. Let us consider example: For example, M1, M2, and M3 would be integrated first. Next, M4, M6 would be integrated. Then, the left side depth control paths are built.



Breadth-first integration: According to breadth first integration all components directly subordinate at each level, moving across the structure horizontally will be incorporated as shown in figure. The components M2, M5 will be integrated first. The next control level incorporate M3, M4.Figure: Top-down integration testing

- **Bottom up Integration Testing**. The bottom-up integration testing begins construction and testing with the components at the lowest level in the program structure. Then low level tested components are integrated and again tested as group. A bottom-up integration strategy may be implemented with the following steps:

1. Low-level components are combined into clusters that perform specific software sub functions.
2. A driver is written to coordinate test case input and output.
3. The cluster is tested.
4. Drivers are removed and clusters are combined moving upward in the program structure.

The bottom up integration is demonstrated in the following diagram:

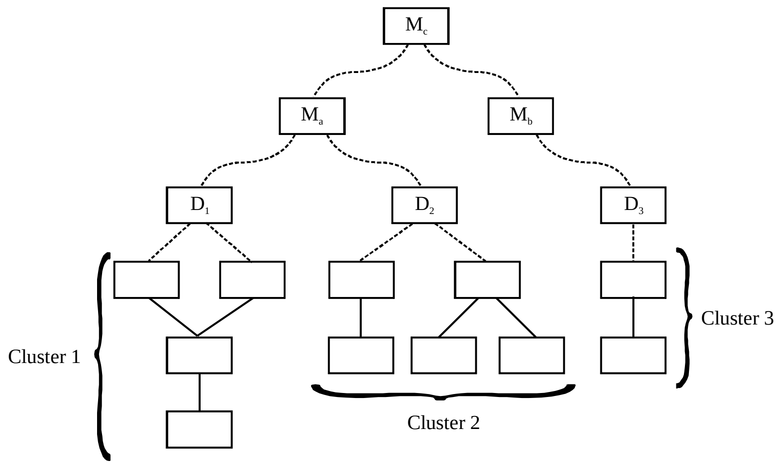


Fig. 7. 13 Bottom-up Integration Testing

The components are combined to form clusters 1, 2, 3. Every cluster is tested using a driver and components in clusters 1 and 2 are subordinate to M*a*. Drivers D1 and D2 are removed and the clusters are interfaced directly to M*a*. Similarly, driver D3 for cluster 3 is removed prior to integration with module M*b*. Both M*a* and M*b* will ultimately be integrated with component M*c*and so forth.

- **Regression Testing:** Regression testing is the activity that helps to ensure that changes (due to testing or for other reasons) do not introduce unintended behavior or additional errors. The regression test suite contains three different classes of test cases:

— Additional tests that focus on software functions.
— A representative sample of tests that will exercise all software functions.
— Tests that focus on the software components that have been changed.

- **Smoke Testing:** Smoke testing is an integration testing approach that is commonly used when “shrink-wrapped” software products are developed. Smoke testing is characterized as a rolling integration approach because the software is rebuilt with new components and testing. Smoke testing encompasses the following activities:

- Software components that have been translated into code are integrated into a “build.” A build includes all data files, libraries, reusable modules, and engineered components that are required to implement one or more product functions.

- A series of tests is designed to expose errors that will keep the build from properly performing its functions. The build is integrated with other builds and the entire product (in its current form) is smoke tested daily.

- **Sandwich Integration Testing.** Sandwich integration testing is the combination of both the top-down and bottom-up approach. So, it is also called mixed integration testing. In it, the whole system is divided into three layers, just like a sandwich: the target is in the middle and one layer is above the target and one is below the target.

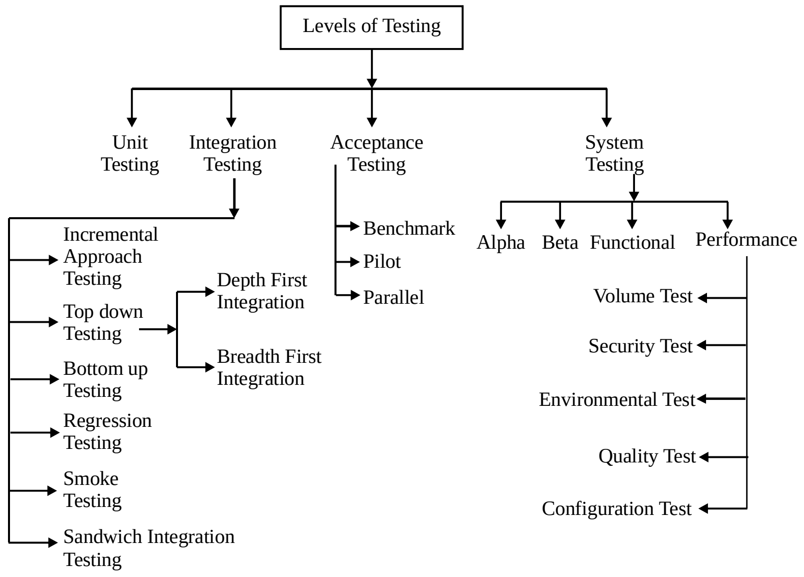


Fig 7.14 Levels of Testing in Detail

7.11.3 System Testing

Once all the components are integrated, the application as a whole is tested thoroughly to see that it meets Quality Standards. This type of testing is performed by a specialized testing team. It is also concerned with validating that the system meets its functional and non-functional requirements. There are essentially three main kinds of system testing:

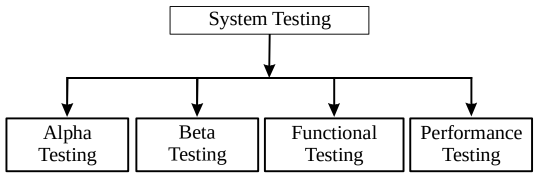


Fig 7.15 Classification of System Testing

- ***Alpha Testing*****:** *Alpha testing refers to the system testing carried out by the test team within the development organization.*

The alpha test is conducted at the developer's site by the customer underthe project team's guidance. In this test, users test the software on thedevelopment platform and point out errors for correction. However, thealpha test, because a few users on the development platform conduct it, has limited ability to expose errors and correct them. Alpha tests are conducted in a controlled environment. It is a simulation of real-life usage. Once the alpha test is complete, the software product is ready for transition to the customer site for implementation and development.

- ***Beta Testing*****:** *Beta testing is the system testing performed by a selected group of friendly customers.*

If the system is complex, the software is not taken for implementation directly. It is installed and all users are asked to use the software in testing mode; this is not live usage. This is called the beta test. Beta tests are conducted at the customer site in an environment where the software is exposed to a number of users.

The developer may or may not be present while the software is in use. So, beta testing is a real-life software experience without actual implementation. In this test, end users record their observations, mistakes, errors, and so on and report them periodically.

In a beta test, the user may suggest a modification, a major change, or a deviation. The development has to examine the proposed change and put it into the change management system for a smooth change from just developed software to a revised, better software. It is standard practice to put all such changes in subsequent version releases.

- ***Functional Testing:*** *Functional Testing of the software is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.*

This is a type of black box testing that is based on the specifications of the software that is to be tested. The application is tested by providing input and then the results are examined that need to conform to the functionality it was intended for. There are five steps that are involved when testing an application for functionality.

1. The determination of the functionality that the intended application is meant to perform.
2. The creation of test data based on the specifications of the application.
3. The output based on the test data and the specifications of the application.
4. The writing of Test Scenarios and the execution of test cases.
5. The comparison of actual and expected results based on the executed test cases.

An effective testing practice will see the above steps applied to the testing policies of every organization and hence it will make sure that the organization maintains the strictest of standards when it comes to software quality.

- ***Performance Testing:*** Performance testing checks the performance objectives set by the customer. It is also known as nonfunctional testing. It measures the accessibility of data, accuracy of the output, speed of response etc. The following are the types of performance testing:

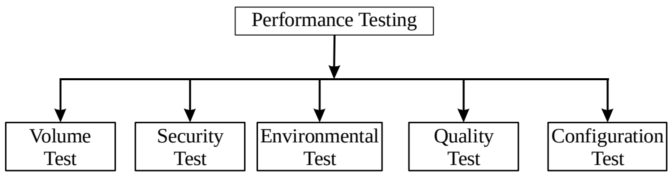


Fig. 7.16 Types of Performance Testing

Volume Test: It address the handling of large amount of data in the system.

Security Test: It ensures the security requirements of the system.

Environmental Test: This test ensures performance of system under the tolerance for heat, motion, portability, moisture and other environmental situations.

Quality Test:It ensures the reliability, availability and maintainability of the system.

Configuration Test: It ensures the various software and hardware configurations explained in the requirements.

7.11.4 **Acceptance Testing**

Acceptance testing is the system testing performed by the customer to determine whether to accept or reject the delivery of the system.

When customer software is built for one customer, a series of acceptancetests are conducted to enable the customer to validate all requirements. Conducted by the end-user rather than the software engineers, an acceptance test can range from an informal 'test drive' to a planned and systematically executed series of tests. In fact, acceptance testing can be conducted over a period of weeks or months, thereby uncovering cumulative errors that might degrade the system over time. Acceptance tests are written, conducted and evaluated by the customers.

Types of acceptance tests:

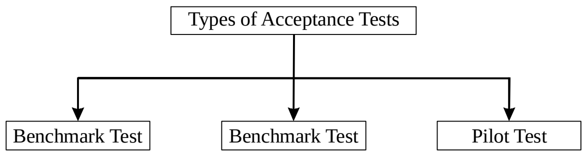


Fig. 7.17 Types of Acceptance Tests

- ***Benchmark Test:***These tests are performed when a user has special requirements. These tests are performed by actual user or a special team.
- ***Pilot Test:***Pilot tests rely on all the functions of everyday working of the system. User exercise the system as if it had been installed permanently.
- ***Parallel Test:*** In this test, the new system installs and operates in parallel with the previous version. It builds the confidence of user in new system with the old system.

BACHELOR OF COMPUTER APPLICATIONS (BCA)
BCA-4-01T: SOFTWARE ENGINEERING

UNIT 8: SOFTWARE MAINTENANCE AND RELIABILTY

8.1 INTRODUCTION

Software maintenance is the activity associated with keeping an operationalcomputer system continuously in tune with the requirements of users and data processing operations. It process is expensive and risky and is very challenging.Maintenance may be classified into the four categories as follows: Corrective,Adaptive,Perfective,and Preventive. The term software reliability is defined as the probability of failure-free operation of a computer program in a specified environment for a specified time. Thedetaileddiscussion about categories of software maintenance along with different components of software maintenance procedures are discussed with appropriate examples.

8.2 SOFTWARE MAINTENANCE

1. After the complete development of software product, software is handed over to the client. It client requires any changes in the software product after the delivery then it is turned as software maintenance. Software maintenance is the most difficult aspect of software development.

Definition

According to ***IEEE***: ***"Software maintenance is a process of modification in a software product after delivery to correct faults, or to improve performance so that software product can adopt the modified environment."***

2. Maximum time is spent on software maintenance as compared to the development of any phase of the software product. So it is very important to kept maintenance in mind throughout the software development.

3. Once an error has been located, the maintenance programmer resolves the errors without adding any error (regression errors). Detailed documentation helps the maintenance programmer in correcting the errors.
4. After receiving a maintenance request, the first step is to identify what type of error is there and what type of maintenance is required.
5. After correcting an error, the fix is tested again to ensure the correct functionality of the software. It also ensure that there should be no other side effects.
6. There are number of software life cycle models which ignore the software maintenance phase and do not provide any adequate information for integration of maintenance into development process.
7. Software development and software maintenance are two processes of software engineering. Software maintenance is based on the information gained from software development. Without this information, software maintenance is very difficult process designing, coding, testing etc.

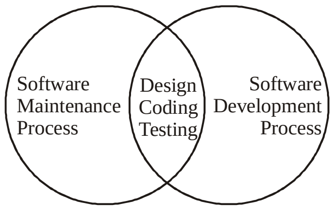


Fig. 8.1 Relationship between Software Maintanance Process & Software Development Process

8. Software maintenance costs are underestimated during the design and implementation phase. But in reality software maintenance costs are the greatest costs.
9. One of the issue regarding maintenance is that software engineers have very poor image of maintenance. They it is unimportant and easy task. That is why organizations allocated this maintenance task into very less skilled and inexperienced staff which results in very high maintenance cost and time.
10. The maintenance cost and time can be reduced by involving experienced and skilled staff early in the software process.
11. In short, software maintenance means modification in a software product after delivery. It mostly occurs at customer’s site. Software maintenance include error detection, error correction and enhancement of capabilities.

8.2.1 Objectives of Software Maintenance

Software maintenance is a set of activities performed in a software product after delivery. Example of software maintenace are changes in code documentation manuals or any part of the product etc. Software maintenance has many objectives which are as follows:

1. Software maintenance enhances the performance of the software.
2. It correct various errors occur during the functionality of the software.
3. It makes changes in the software so that it can adopt the changes in the environment.
4. Software maintenance also update the software product to avoid future problems.

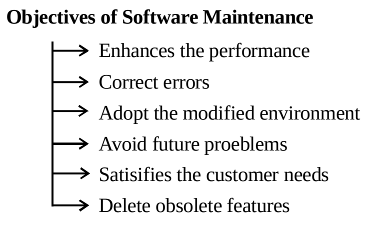

Objectives of Software Maintenance
Enhances the performance
Correct errors
Adopt the modified environment
Avoid future proeblems
Satisifies the customer needs
Delete obsolete features

Fig. 8.2 Objectives of Software Maintenance

5. Maintenance process satisfies the customer needs. After maintenance, software starts working according to user requirements.
6. Software maintenance is used to correct program errors, to add new capabilities into the software to delete obsolete features and to improve performance of the software.

8.2.2 Categories of Software Maintenance

Software maintenance is categorized into four types which are as follows:

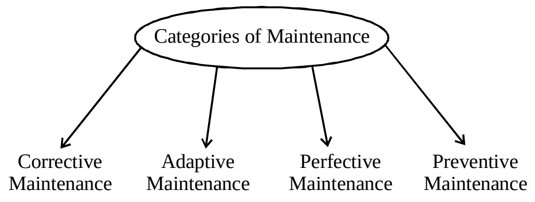


Fig. 8.3 Categories of Maintenance

I. Corrective Maintenance

1. Corrective maintenance is necessary to resolve the bugs occur during the usage of a software product. Bugs can occur at any stage during the development of software for example—bugs can occur during the specifications, design, coding and testing etc.
2. Corrective maintenance changes the software to correct the errors. Some errors are caused by data processing error and system performance errors.
3. An error can result from design errors, logic errors and coding errors which are as follows:

- **Designing errors:** These errors occurs when change are made in the software. These changes are incorrect, incomplete or misunderstood and arises errors.
- **Logic errors:** These errors occurs when invalid tests, incomplete test data of design specification and incorrect implementations are made in the software.
- **Coding errors:**These errors occurs when incorrect use of source code logic and wrong implementation of detailed logic design are made in the software.

II. Adaptive Maintenance

1. Adaptive maintenance makes changes in the software so that it can work in new environment (new platforms) without any error. For example—porting to a new compiler, operating system and/or hardware.
2. The term new environment refers to the totality of all conditions/influences that act from outside on the software.
3. Adaptive maintenance in the software is not the need of the client. It is the need of external environment in which software works.

III. Perfective Maintenance

1. Perfective maintenance needs time and money. It improves the efficiency, effectiveness and maintainability of the software.
2. In perfective maintenance, additions or enhancements are made in the software to improve its functionality and quality.
3. The perfective maintenance is requested by software engineer and the client. The software engineer needs perfective maintenance to improve the status of the product in market by improving its quality. The client needs perfective maintenance to meet the new requirements.

IV. **Preventive Maintenance**

1. Preventivemaintenance performs the activities/operations on the software to update it in anticipation of any feature problems.
2. It performs modifications in a software after delivery, to improve the effectiveness, efficiency and performance so that software does not face any problem (errors) in the future.
3. Preventive Maintenance is basically a modification in the software after the delivery to detect and correct the faults. It ensures the safety of the software.

8.2.3 Components **of Software Maintenance Process**

The components mean those parts which play important role in software maintenance. The important components of the software maintenance process are people, tasks and knowledge of the software.These components are inter-related and interdependent on each other.

1. **People:** It is the most important component of the software maintenance process. This component covers all those persons which are the part of the maintenance process directly or indirectly. It includes users, development staff and maintenance staff.

- Users are those people for whom software is developed and maintained.
- Development staff is a team of those people who takes responsibility for the successful development of the software product
- Maintenance staff includes those people who participated actively in the software product after the complete development of the software to make it up to date.

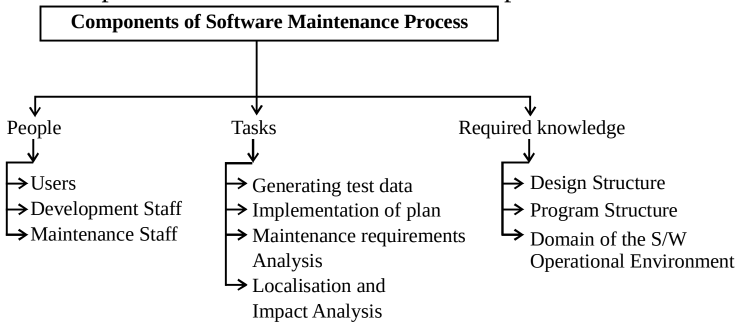

Components of Software Maintenance Process
People
Tasks
Required knowledge
Users
Development Staff
Maintenance Staff
Generating test data
Implementation of plan
Maintenance requirements Analysis
Localisation and Impact Analysis
Design Structure
Program Structure
Domain of the S/W Operational Environment

Fig. 8.4 Components of Software Maintenance Process

2. **Tasks:** Tasks means those activities or operations which are performed during successful maintenance process. During maintenance process, various tasks like implemation plan, maintenance requirements analysis, localization and impact analysis etc. are performed.

- Implementation plan provides the step by step maintenance requirement analysis and the need of maintenance requirement is checked.
- It is used to examine the requirements of the maintenance. Impact analysis include the detailed study of software product after the maintenance process. Performance, efficiency and effectiveness of software product is examined after the successful implementation of maintenance process.

3. **Required knowledge:** This component includes the structured and unstructured knowledge for the software maintenance. Maintenance staff should have the knowledge of the design and program structure of the software product. Maintenance staff should have complete knowledge of domain of the software alongwith the operational environments.

8.2.4 **Software Maintenance Process**

The maintenance process has five steps which are follows:

1. **Understand the existing programs:** In this step, before modification, existing programs are studied for complete understanding. Important attributes which are helpful in better understanding of

existing programs are: less complexity and proper documentation. In this step, objectives are also studied in the detailed manner.

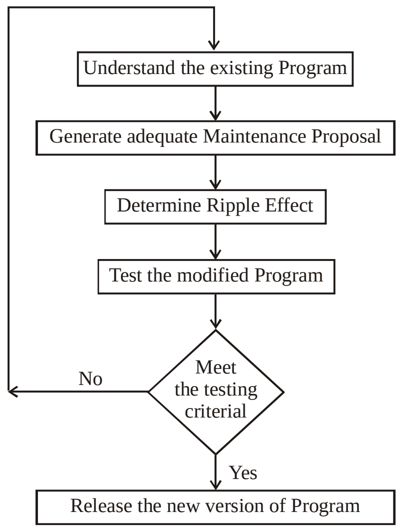

Understand the existing Program
Generate adequate Maintenance Proposal
Determine Ripple Effect
Test the modified Program
Meet the testing criterial
No
Yes
Release the new version of Program

Fig. 8.5 Steps of Software Maintenance Process

2. **Generate adequate maintenance proposal:** In this step, maintenance team generates and propose the adequate maintenance proposal to satisfy the maintenance objectives.

3. **Determine ripple effect:**

- Ripple effect determines the effect of the change (modification) on the system.
- Ripple effect is logical/functional in nature. It concerns the performance of the program.
- The ripple effect is affected by the consequence of a program modification i.e. the stability of the program.
- Program stability is basically a resistance to the amplification of changes in the program.
- Regression testing is performed to ensure that unchanged code is still working efficiently after making the new changes in the software product.

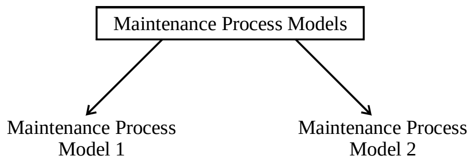
4. **Test the modified program:** In this step, modified program is tested to ensure the reliability level. Modified program should have at least same reliability level as before. Cost effective testing techniques are also performed (during the maintenance process) to measure the required effort to test the modified program, under well-defined testing conditions.

5. **Release the new version of program:** After meeting the testing conditions, modified programs with new versions are ready to install/release.

Maintenance Process Models

No single maintenance process model is suitable for all types of maintenance projects. Maintenance process activities are different for different maintenance projects.

There are two bread categories of maintenance process models are proposed.


Maintenance Process Models
Maintenance Process Model 1
Maintenance Process Model 2

Model 1 preferred for projects where small re-work is requried. In this model, the code is changed directly and the changes are reflected in the relevant documents later.

Model 2 preferred for projects where amount of rework required is significant. In this model, reverse engineering cycle is followed by a forward engineering cycle. This approach is also known as software reengineering.

Fig. 8.6 Maintenance Process Models

1. **Maintenance Process Model 1:** The maintenance process model 1 starts with gathering the requirements for change. This is the first step of Model 1. In this step, all the change requirements are gathered in a detailed manners. In this step 2, these change requirements are analyzed and studied from different angles. In this step 3, after analyzing the change requirements, efficient strategies are formulated and adopted for code change. At this stage, few members of original development team goes a long way in reducing the cycle time, for the projects involving unstructured and inadequate documented code.

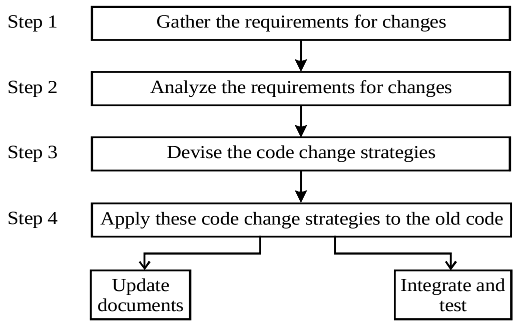

Step 1
Gather the requirements for changes
Step 2
Analyze the requirements for changes
Step 3
Devise the code change strategies
Step 4
Apply these code change strategies to the old code
Update documents
Integrate and test

Fig. 8.7 Maintenance Process Model 1

In this step 4, apply the formulated and adopted code strategies to the old code. Maintenance engineers study the working of the old system and compare the working of their modified system with the old system.

In the end, debugging of the new modified system become easy as after the comparison of the old and new modified system, bugs can be easily localize.In the model 1, the output is update documents and integrate and test the modified program.

2. **Maintenance Process Model 2**

- Maintenance Process Model 2 is basically suitable for those projects where amount of rework is required. In this model reverse engineering cycle is followed by a forward engineering cycle which is known as software re-engineering.
- During the reverse engineering, the old coding is analyzed to extract the module specification. The module specification is further analyzed for producing the designing part. The design is further analyzed to produce the original requirement specification. This process is known as reverse re-engineering because analyzing process is done in reverse order means from bottom to top stages.
- After analyzing the original requirement specifications, change requirements are applied to the original requirement specifications, to produce the new requirement specification. After arriving at the new requirement specification, forward engineering approach is used to produce a new code.

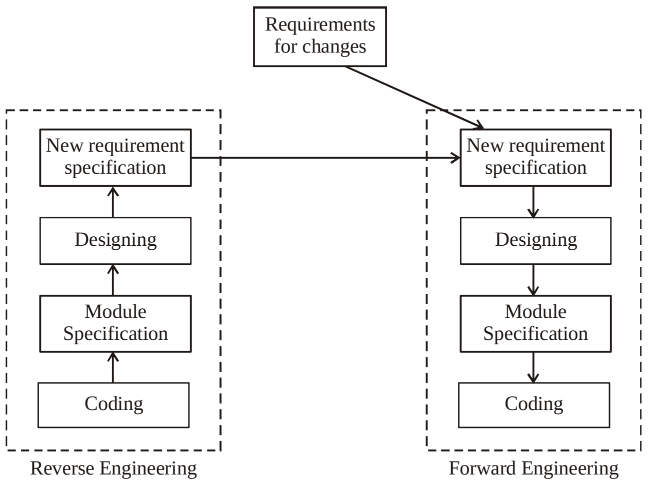


Fig. 8.8 Maintenance Process Model 2

- During the forward engineering approach, new requirement specifications are considered. At the designing, module specification and coding stages, a substantial reuse is made from the reverse engineered products. This approach produces good documentation, more structured design as compared to the original product.

Comparison between Model 1 and Model 2

1. The maintenance process model 2 gives efficient outputs, efficient designs and adequate documentation as compared to maintenance process model 1.
2. The maintenance process model 2 is cost lies than the process model 1.
3. The maintenance process model 2 is preferred only when amount of rework is very high. If amount of rework is not more than 20%, then we use model 1.

8.2.5 **Factors affecting the Software Maintenance**

There are broadly two categories of factors which affects the software maintenance.

I. **Technical Factors:** The technical factors of software maintenance are as follows:

1. **Programming Language:** Programming language plays important role in the software maintenance. Programs written in the high level language are easier to understand and maintained than the programs written in low level languages.
2. **Program Testing:** If a program is adequately validated and tested during the design phase, then there are less chances of occurring errors in the software product. If development staff spends maximum effort and time on the design testing then maintenance cost is reduced because of fewer errors in the program.

   Maximum time and effort in design phase = Less maintenance cost

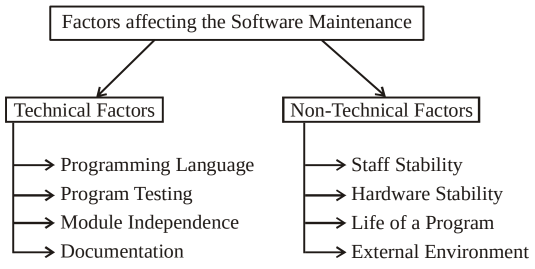


Fig. 8.9 Factors affecting the Software Maintenance

3. **Module Independence:** Module independence means modification of one program unit of a system without affecting any program unit of a system. If one module is being modified, then there should be no effect on other modules of software product.
4. **Documentation:** Documentation is another important factor of software maintenance. If a program is clearly, completely and adequately documented, then it provides the complete understanding of the program.

II. **Non-Technical Factors:** Then non-technical factors of software maintenance are as follows:

1. **Staff Stability:** The stability of staff effects the maintenance cost.

   More stable staff = less maintenance cost

   If an original writer of a program understands and changes a program, then it will be very easy for him to maintain that program and becomes very beneficial for the development company in tums of cost and time. It is very difficult for individual to understand and changes in a program written by another person (a new programmer).
2. **Hardware Stability:** The program must be modified to use new hardware. The new hardware replaces the old (obsolete) equipment. Sometimes a program is designed for a particular hardware configuration and the chances of the change of that particular hardware configuration is very less during the program's whole life. In this situation there is no maintenance cost, but it is rarest case.
3. **Life of a Program:** As the life of a program increases, the maintenance cost increases. The life of a program depends upon the application used. When the application becomes obsolete, program becomes obsolete and maintenance cost increases very rapidly because of conversion of original obsolete hardware into new hardware.

   Life of Program = Maintenance cost and time
4. **External Environment:** Sometimes, program depends upon the external environments. According to the external environments, program needs changes and maintenance cost increases. Whole maintenance process and maintenance cost depends upon the external environment.

   Changes in external environment αHigher maintenance cost and time.

8.3 SOFTWARE RELIABILITY

1. Software reliability is an important concern for users. Software reliability is an attribute of software quality and it is independent from time.
2. More the software reliability, higher the software quality.
3. If software reliability increases, then the performance usability, functionality and stability of software increases.
4. According to ***IEEE: "Software reliability is defined as the probability of failure free software operation for a specified period of time in a specified environment."***
5. Reliability of a software product denotes the twist-worthiness, software reliability refers to as the probability of the product working correctly over a given time duration.
6. Sometimes, it is very difficult to measure accurately the reliability of a software product because of following two reasons:

(i) Reliability is observer dependent. It depends upon the observer's new point.

(ii) Reliability of a software product keeps changing as errors are detected and fixed. Because of frequent changing nature of software product, problems occur during the accurate measurement of the reliability of the software product.

8.3.1 Factors affecting Software Reliability

There are many factors which affects the growth of software reliability. These factors increases or decreases the growth of software reliability. These factors are as follows:

1. **Number of Errors:** Software reliability depends upon the number. of errors in the software product. Reliability is inversely proportional to the number of errors (defects) occur in the software product.

$$\text{Reliability } \alpha \frac{1}{\text{No. of Errors}}$$

   Reliability increases when no. of errors decreases in a particular software product. Reliability decreases when no. of errors increases in a particular software product.

2. **Exact Location of Errors:** Software reliability increases when maintenance staff finds out the exact location of error. If exact location of error is finds out, then it becomes very easy for maintenance staff to remove that error and enhance the reliability and quality of a particular software product.
3. **How Product is used:** Software reliability also depends upon the way in which product is being used. If proper and correct way is used then it increases the reliability of a particular software product.
4. **Time:** Software reliability is not a function of time and different from hardware failure. With the passage of time, hardware components become old and obsolete. Hardware failure parts can be replaced with rate of hardware reliability increases. But in the case of software reliability, software cannot wear out. It can be upgraded intentionally by identifying and resolving errors and failure rate decreases. But when the software become obsolete, no more error correction occurs and failure rate remains unchanged.

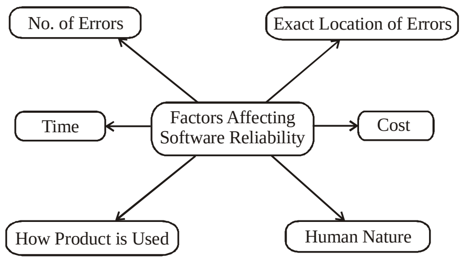


Fig. 8.10 Factors affecting Software Reliability

5. **Cost:** Software reliability depends on cost. Testing is a costly process. When more testing is performed on any software product to improve its reliability, cost automatically increases.
6. **Human Nature:** Software reliability also depends upon the human nature and observer's view point. Level of reliability various with the different nature of different people.

8.3.2 Comparison of Software Reliability and Hardware Reliability

Measuring the software reliability is more difficult as compared to measuring the hardware reliability. The difference between hardware reliability and software reliability are as follows:

Sr. No	Hardware Reliability	Software Reliability
1	Hardware reliability is concerned with stability.	Software reliability is concerned with reliability growth.
2	Hardware components fail because of wear and tear.	Software components fail because of bugs
3	To fix a hardware fault, one has to repair or replace the failed parts.	To fix a software fault, one has to fix the bug, code is changed to fix the bug.
4	When the hardware failure is repaired its reliability would be maintained at the level that existed before the failure occurred.	When the software failure is repaired, its reliability may increase or decrease.
5	Hardware reliability is a function of time and different from software failure.	Software reliability is not a function of time and different from hardware failure.

Table 8.1 Comparison of Software Reliability and Hardware Reliability

8.3.3 Uses of Reliability Studies

1. It estimate and predict the reliability behavior of software during its development and operation.
2. It answer the various question like what is the expected duration between successive failures.
3. It helps software engineering methodologies to develop the code.
4. It use formal methods during code development.
5. It compares the software engineering technologies in terms of cost and quality.
6. It aims at fault-free performance of software systems.
7. It concerns itself with how well the software functions to meet the requirements of the user.
8. It measures the progress of system testing.
9. It gives us a better insight into the development processes.

8.3.4 Software Reliability Metrics

Software reliability metrics are used to express quantitatively the reliability of the software product. No single metric is present which explains the reliability of all software products. Different software products may have different reliability requirements. The following are some reliability metrics:

1. **Rate of Occurrence of Failure (ROCOF):** It measures the frequency of the occurrences of failure. ROCOF can be obtained by observing the operational behavior of a software product over a specified time interval.

$$\text{ROCOF} = \frac{\text{Total number of observed failures}}{\text{Duration of observation}}$$

2. **Mean Time to Failure (MTTF):** MTTF is mean time between two successive failures. MTTF is measured by recording the failure data for n failures.

$$\text{MTTF} = \sum_{i-1}^{n} \frac{t_{i+1} - t_i}{(n-1)}$$

3. **Mean Time to Repair (MTTR):** After the occurrence of failure, some time is required to fix the error to solve that failure.

MTTF is the average time to track and fix the errors.

4. **Mean Time between Failure (MTBF):** MTBF metric is a combination of MTTF and MTTR

MTBF = MTTF + MTTR

5. **Availability (Avail):** It is a measure of how likely would the system be available for use over a given period of time. Availability is a measure of time during the system is available over long time periods.

$$\text{Availability} = \frac{\text{MTBF}}{(\text{MTBF} + \text{MTTR})}$$

8.3.5 Software Reliability Growth Models

A software reliability growth model is a mathematical model which helps in detection of errors, repair those errors and improves the software reliability. It also provides information when to stop testing to attain given reliability growth level.

The two software reliability growth models are as follows:

1. Jelinski and Moranda Model
2. Littlewood Model

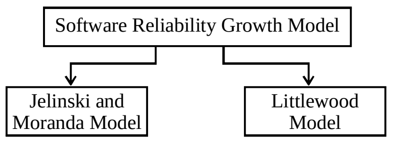


Fig. 8.11 Software Reliability Growth Models

1. **Jelinski and MorandaModel:**It is the simplest reliability growth model. This model assumes that reliability increases each time when an error is detected and repaired. Reliability is directly proportional to error detection and fixing.

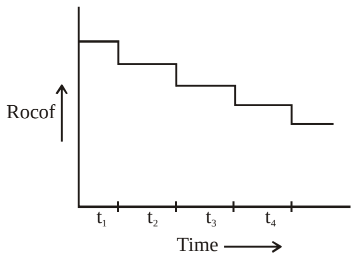


Fig. 8.12 Jelinski and Moranda Model

2. **LittlewoodModel:**It is the complex reliability growth model. This model assumes that there is a chance of introducing additional errors when an error is detected and repaired.

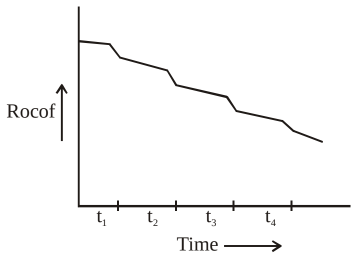


Fig. 8.13 Littlewood Model