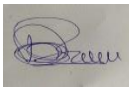
COURSE PACK FOR

Subject Title: - Software Engineering (CBCS 2018 course)

COURSE CODE : 302
COURSE **:** BCA
SEMESTER **:** III
YEAR **:** 2020-21

Course Instructor: Dr. Daljeet Singh Bawa

Course Leader: Dr. Daljeet Singh Bawa 

(Dr. A.K. Srivastava) **(Dr.VikasNath)**

Forwarded by:HOD **ApprovedBy:Director(I/C)**

BharatiVidyapeeth(Deemed to be University), Institute of Management & Research, New Delhi
An ISO 9001: 2015 & 14001:2015 Certified Institute
A-4, PaschimVihar, New Delhi-110063(Ph:011-25284396,25285808, Fax No. 011-25286442)

Note: "Strictly for Internal academic use only"

TABLE OF CONTENT

BHARATI VIDYAPEETH (DEEMED TO BE) UNIVERSITY

INSTITUTE OF MANAGEMENT AND RESEARCH, NEW DELHI

COURSE OUTLINE

Course: BCA	**Semester:III**	**Academic Year: 2020-21**
Course Code: 302	**Credits: 04**	**No. of teaching Hours: 40+**

1. **Course Title:** Software Engineering

2. **Course Overview:**

Unit 1 begins with the Definition of Software Engineering, Software Engineering Fundamentals, discussion on Software Development Models, Program Vs Software, Importance & Principles of Software Engineering, Difference between Software Engineering and Software Programming, Software Project Management Concepts followed by the concept of Software Metrics and Application of PERT and GANTT charts.

Unit 2 discusses about Software Development Cycle, General Software development life cycle, Comparison between waterfall, prototyping and spiral model, Comparative Study of Incremental Model & RAD Model, Component Based Development, Feasibility Study and Cost Benefit Analysis.

Unit 3 covers the concept of Requirement Engineering, Types of Requirements and Requirement Elicitation Techniques., the concept of **SRS,** Completeness, Unambiguity, Inconsistency, IEEE SRS,

Unit 4 covers DFD, ERD, Structure Chart, Data Dictionaries, UML Introduction, Use Case Diagrams, decision trees, decision tables and Class Diagrams, coupling and cohesion, structure charts

Unit 5 covers Testing Techniques and Types of test cases & test plans, SCM and introduces the Software Quality Concepts (What is Quality, Quality Control, Quality Assurance, Cost of Quality, Software Quality Assurance) and Software Reviews (Formal Technical Reviews, The Review Meeting, Review Reporting, Record Keeping, Review Guidelines, Formal Approaches to SQA, Statistical Quality Assurance, Software Reliability and SQA Plan)

Unit 6 discusses about Software Maintenance, its Problems, Corrective Maintenance, Adaptive Maintenance, Perfective Maintenance and Preventive Maintenance, Potential Solutions to Maintenance and Maintenance Process & Models

3. Course Objectives:

The objective of the course is to introduce the current methodologies involved in the development and maintenance of software over its entire life cycle. It revolves around concepts like when computer software succeeds – when it meets the needs of the people who use it, what it can and does change things for better, but when software fails - when its users are dissatisfied, when it is error prone, when it is difficult to change. To succeed and overcome failures, we need discipline when software is designed and built- an engineering approach. Software Engineering will help to:

1. Introduce the methodology involved in the development and maintenance of software over its entire life cycle.
2. Understand life cycle models, Requirement elicitation techniques, understand the concept of Analysis and Design of software.
3. Implement software engineering concepts in software development to develop quality software which can work on any real machine.

Following are the course objectives:

1. Knowledge of basic S/W engineering methods and practices and their appropriate application.
2. Describe software engineering layered technology and Process frame work.
3. A general understanding of software process models such as the waterfall and evolutionary models.
4. Understanding of software requirements, software elicitation techniques and requirements engineering process
5. Understanding of the SRS document, function orientated design and object oriented design.
6. Describe data models, object models, context models and behavioural models.
7. Understanding of the role of project management including planning, scheduling, risk management, etc.
8. Understanding of approaches to verification and validation including static analysis, and reviews.
9. Understanding of software testing approaches such as unit testing and integration testing.
10. Describe software measurement and software risks.
11. Understanding on quality control and how to ensure good quality software.
12. Understanding of software maintenance, solutions to software maintenance and maintenance process and models.

4. Course Outcomes:

After undergoing this course, the student will have:

CO1: Basic knowledge and understanding of the analysis and design of complex systems. Knowledge of basic S/W engineering methods and practices and their appropriate application.

CO2: Ability to apply software engineering principles and techniques. A general understanding of software process models such as the waterfall and evolutionary models

CO3: Implementing software elicitation techniques, requirements engineering process and developing the SRS document.

CO4: Develop various object oriented and function oriented designs.

CO5: Apply various testing techniques and develop test cases.

CO6: Apply the concepts of Quality Control, Quality Assurance, Formal technical reviews and making SQA plan.

CO7: Knowledge of implementation of software maintenance process and S/w maintenance models.

CO8: Ability to develop, maintain and evaluate - efficient, reliable, robust and cost-effective software solutions.

5. List of Topics/ Modules/Syllabus:

Topic/ Module	Contents/ Concepts
Module I: Introduction to Software Engineering	• **Software Development Models:** Program Vs Software • Definition of Software Engineering • Importance & Principles of Software Engineering • Difference between Software Engineering and Software Programming • Members involved in software development.
Module II:Software Process& Feasibility Study	• **Software Development Cycle** • General Software development life cycle • Comparison between waterfall, prototyping and spiral model • Comparative Study of Incremental Model & RAD Model • Component Based Development • Fourth Generation Techniques **Feasibility Study** • Need of Feasibility Study • Types of Feasibility **Cost Benefit Analysis** • Why Cost Benefit Analysis? • Cost Benefit Analysis Process
Module III: Requirement Engineering Concepts and Methods	• What is Requirement Engineering? • Types of Requirements • Requirement Elicitation Techniques • Traditional and Modern Methods • Verification and Validation Process • Principles of requirement specification • **Characteristics of good SRS:** • Completeness, correct, Unambiguity, consistent , modifiable, traceable, understandable, IEEE SRS •
Module IV: Analysis and structured system design tools	• DFD, ERD, Structure Chart, decision tree, decision table, Data Dictionaries, pseudo code, input and output design • **Modules concept, types of modules** • **Qualities of good design** • **Coupling – types of coupling** • **Cohesion – types of cohesion**

	•
Module V: Software testing and software quality assurance	**Testing Techniques** • Different testing techniques with examples • Development and Execution of Test Cases: Debugging, Testing Tools & Environments • Types of test cases and test plans • **Software Quality Concepts** • **Quality Concepts** • What is Quality, Quality Control, Quality Assurance, Cost of Quality, Software Quality Assurance, Software Reviews • **Formal Technical Reviews** • The Review Meeting • Review Reporting • Record Keeping • Review Guidelines • Formal Approaches to SQA • Statistical Quality Assurance • Software Reliability • SQA Plan • **SCM Process – Identification of objects, version control and change control.**
Module VI: Software Maintenance	What is Software Maintenance? Problems during Software Maintenance **Categories of Software Maintenance:** Corrective Maintenance,Adaptive Maintenance Perfective Maintenance and Preventive Maintenance **Potential Solutions to Maintenance:** Budget and efforts reallocation, complete replacement, maintenance of existing system **Maintenance Process and Models:** Maintenance Process, Fix Model, Iterative Enhancement Model, Reuse Oriented Model, Boehm Model and Taute's Models

6. Evaluation Criteria:

Component	Description	Weight age
First Internal Examination	First internal question paper will be based on first 3 unit of syllabus.	10marks
Second Internal Examination	Second internal question paper will be based on last 3 unit of syllabus.	10marks
CES 1 Quiz	Moodle MCQs based on the concepts of Software Engineering	5 marks
CES 2Quiz	Moodle MCQs based on the concepts of Software Engineering	5 marks
CES 3 Quiz	Moodle MCQs based on the concepts of Software Engineering	5 marks
Attendance	Above 75% - 10 marks Below 75% - 0 mark	10 marks
Note : All three CES will be mandatory. If any student misses anyone CES in that case the weightage of each CES would be 3.33 marks and if a student attempts all three CES then his/her best two CES will be considered, in that case the weightage would be 5 marks each.		

7. Recommended/ Reference Text Books and Resources:

Text Books	Text Book1: A:SOFTWARE ENGINEERING A PRACTITIONERS APPROACH Sixth Edition BY Roger S. Pressman, McGraw Hill International Edition B: SOFTWARE ENGINEERING A PRACTITONERS APPROACH Seventh Edition BY Rogers S. Pressman, McGraw Hill International Edition
Course Reading / Reference Books	• Software Engineering By Sommerville, Pearson Education, 7th Edition • Software Engineering by K.K. Aggarwal &Yogesh Singh, New Age International Publishers
Internet Resource:	www.beknowledge.com/wp-content/uploads/2010/.../8f14etx427_11.pdf http://www.tutorialspoint.com/software_engineering/software_engineering_tutorial.pdf

8. Suggested MOOC: Please refer to following websites for MOOCS:

NPTEL/Swayam

www.edx.com

www.coursera.com

9. Session Plan:

Unit	Lecture	Topic Details	Ref. Book	Learning Outcomes
Unit: 1 Software Engineering Concepts:	1	Program Vs Software	Text Book 1 Chapter1,2	Basic knowledge and understanding of the analysis and design of complex systems. LO1
	2	Definition of Software, Software types, Software components	Text Book 1 Chapter1,2	Basic knowledge and understanding of the analysis and design of complex systems.LO1
	3	Definition of Software engineering, Importance	Text Book 1 Chapter1,2	Basic knowledge and understanding of the analysis and design of complex systems. LO1
	4	Principles of Software engineering	Text Book 1 Chapter1,2	Basic knowledge and understanding of the analysis and design of complex systems. LO1
	5	Difference b/w Software engineering and Software programming	Text Book 1 Chapter1,2	Basic knowledge and understanding of the analysis and design of complex systems. LO1
	6	Members involved in Software development	Text Book 1 Chapter1,2	Basic knowledge and understanding of the analysis and design of complex systems. LO1
	7	Qualities required for project Manager/ Team Leader Software team organizations	Text Book 1 Chapter1,2	Basic knowledge and understanding of the analysis and design of complex systems. LO1
	8	Generic view of Software engineering.	Text Book 1 Chapter1,2	Basic knowledge and understanding of the analysis and design of complex systems. LO1
Unit 2: Software	9	**Software Development Cycle**	Text Book 1	Ability to apply software engineering

Process and Feasibility Study:		General Software development life cycle	Chapter 2	principles and techniques LO1,LO2
	10	Comparison between waterfall, prototyping	Text Book 1 Chapter 2	Ability to apply software engineering principles and techniques LO1, LO2
	11	Comparison between waterfall, prototyping and spiral model	Text Book 1 Chapter 2	Ability to apply software engineering principles and techniques LO1, LO2
	12	Comparative Study of Incremental Model & RAD Model	Text Book 1 Chapter 2	Ability to apply software engineering principles and techniques LO1, LO2
	13	Component Based Development	Text Book 1 Chapter 2	Ability to apply software engineering principles and techniques LO1, LO2
	14	Fourth Generation Techniques	Text Book 1 Chapter 2	Ability to apply software engineering principles and techniques LO1, LO2
	15	**Feasibility Study** Need of Feasibility Study Types of Feasibility	Text Book 1 Chapter 2 Handout	Ability to apply software engineering principles and techniques LO1, LO2
	16	**Cost Benefit Analysis** Why Cost Benefit Analysis? Cost Benefit Analysis Process	Text Book 1 Chapter 2 Handout	To understand the CBA LO1, LO2, LO3, LO4
	17	**Class Test**		
Unit 3: Requirement Engineering concepts and methods	18	What is Requirement Engineering? Types of Requirements	Text Book 1 Chapter 5,6,7	Implementing software elicitation techniques, requirements engineering process and developing the

			Handout	SRS document LO3
	19	Requirement Elicitation Techniques	Text Book 1 Chapter 5,6,7	Implementing software elicitation techniques, requirements engineering process and developing the SRS document LO3
	20	Traditional and Modern Methods Verification and Validation Process	Text Book 1 Chapter 5,6,7 Handout	Implementing software elicitation techniques, requirements engineering process and developing the SRS document LO3
	21,22,23	Principles of requirements specification, SRS, Characteristics of Good SRS, Completeness, correct, Unambiguity, consistent , modifiable, traceable, understandable, IEEE SRS	Text Book 1 Chapter 5,6,7 Handout	Implementing software elicitation techniques, requirements engineering process and developing the SRS document LO3
	24	**Quiz**		
Unit 4: Analysis and structured system design tools.	24	**Function Oriented Modelling:** DFD, ERD,	Text Book 1 Chapter 5,6,7 Handout	Develop various object oriented and function oriented designs. LO3, LO4
	25	Structure Chart, Data Dictionaries	Text Book 1 Chapter 5,6,7 Handout	Develop various object oriented and function oriented designs. LO3, LO4
	26	decision tree, decision table,	Text Book 1 Chapter 5,6,7 Handout	Develop various object oriented and function oriented designs. LO3, LO4

	27	Constructing Solution to Problem Identifying Components and their interaction	Text Book 1 Chapter 5,6,7 Handout	Develop various object oriented and function oriented designs. LO3, LO4
	28	Visualizing the Solution Characteristics of a good function Oriented design (Coupling, Cohesion etc.).	Text Book 1 Chapter 5,6,7 Handout	Develop various object oriented and function oriented designs. LO3, LO4
	29	**Object Oriented Design** Identification & Specification problem domain static objects Working out the application logic objects	Text Book 1 Chapter 5,6,7 Handout	Develop various object oriented and function oriented designs. LO3, LO4
	30	Identification of necessary utility objects Methodology of identification of objects, Case Study	Text Book 1 Chapter 5,6,7 Handout	Develop various object oriented and function oriented designs. LO3, LO4
Unit 5: **S/w Testing and S/W Quality assurance**	31	**Testing Techniques** Different testing techniques with examples	Text Book 1 Chapter 17,18,19 Handout	Apply various testing techniques and develop test cases. LO5
	32	Development and Execution of Test Cases: Debugging, Testing Tools & Environments Types of test cases and test plans	Text Book 1 Chapter 17,18,19 Handout	Apply various testing techniques and develop test cases. LO5
	33	**Software Quality Concepts** **Quality Concepts** What is Quality, Quality Control, Quality Assurance, Cost of Quality, Software Quality Assurance,	Text Book 1 Chapter 14,15,16 Handout	Apply the concepts of Quality Control, Quality Assurance, Formal technical reviews and making SQA plan. LO6

		Software Reviews		
	34	**Formal Technical Reviews** The Review Meeting Review Reporting	Text Book 1 Chapter 14,15,16 Handout	Apply the concepts of Quality Control, Quality Assurance, Formal technical reviews and making SQA plan. LO6
	35	Record Keeping Review Guidelines	Text Book 1 Chapter 14,15,16 Handout	Apply the concepts of Quality Control, Quality Assurance, Formal technical reviews and making SQA plan. LO6
	36	Formal Approaches to SQA Statistical Quality Assurance Software Reliability SQA Plan	Text Book 1 Chapter 14,15,16 Handout	Apply the concepts of Quality Control, Quality Assurance, Formal technical reviews and making SQA plan. LO6
	37	SCM Process – Identification of objects, version control and change control.	Text Book 1 Chapter 14,15,16	Apply the concepts of Quality Control, Quality Assurance, Formal technical reviews and making SQA plan. LO6
	38	**Class Test**		
Unit 6: Software Maintenance	39	What is Software Maintenance? Problems during Software Maintenance	Text Book 1 Chapter 29	Knowledge of implementation of software maintenance process and S/w maintenance models LO7, LO8
	40	**Categories of Software Maintenance:** Corrective Maintenance, Adaptive Maintenance Perfective Maintenance and Preventive Maintenance	Text Book 1 Chapter 29	Knowledge of implementation of software maintenance process and S/w maintenance models LO7, LO8
	41	**Potential Solutions to Maintenance:** Budget and efforts reallocation,	Text Book 1	Knowledge of implementation of

		complete replacement, maintenance of existing system	Chapter 29	software maintenance process and S/w maintenance models LO7, LO8
	42	**Maintenance Process and Models:** Maintenance Process, Fix Model, Iterative Enhancement Model	Text Book 1 Chapter 29	Knowledge of implementation of software maintenance process and S/w maintenance models LO7, LO8
	43	Reuse Oriented Model, Boehm Model and Taute's Models	Text Book 1 Chapter 29	Knowledge of implementation of software maintenance process and S/w maintenance models LO7, LO8
	44	Revision		
	45	Revision		

10. Brief Profile and Contact Details

Dr. DALJEET SINGH BAWA

Mobile: 9582035733

E-Mail: daljeetsinghbawa@gmail.com

Dr. Daljeet Singh Bawa is PhD in Computer Science and is presently working as Assistant Professor in IT Department at BharatiVidyapeeth University Institute of Management and Research, New Delhi. He has completed M.Phil (Computer Science) as well and loves experimenting with new softwares. His areas of specialization are Software Engineering, Operating Systems, Computer Organization and Architecture, e-learning and e-assessment and has a rich experience of working with live software projects. His research work revolves around e-learning, blended learning and e-assessment and has 24 research papers to his credit. He can be contacted at daljeetsinghbawa@gmail.com.

Name of the Instructor:	Dr. Daljeet Singh Bawa
Office Location:	Faculty Cabin – 208, Second Floor, BVIMR, New Delhi
Telephone:	**011-25285808, Extn.: 258**
Email:	daljeetsinghbawa@gmail.com
Teaching Venue:	*Assigned classroom as per timetable*
Website:	www.bvimr.com
Office Hours:	09:00 am to 05:00 pm

Study Notes

Unit – 1

- **Software Development Models:**
- **Program Vs Software**
- **Definition of Software Engineering**
- **Importance & Principles of Software Engineering**
- **Difference between Software Engineering and Software Programming**
- **Members involved in software development**

Software Overview

LetusunderstandwhatSoftwareEngineeringstandsfor.Thetermismadeoftwo words, software andengineering.

Software is more than just a program code. A program is an executable code, whichservessomecomputationalpurpose.Softwareisconsideredtobecollection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called **softwareproduct.**

Engineering on the other hand, is all about developing products, using well- defined, scientific principles and methods.

Software engineering is an engineering branch associated with developmentof softwareproductusingwell-definedscientificprinciples,methodsandprocedures. Theoutcomeofsoftwareengineeringisanefficientandreliablesoftwareproduct.

Definitions

IEEE defines software engineering as:

> (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering tosoftware.
>
> (2) The study of approaches as in the abovestatement.

Fritz Bauer, a German computer scientist, defines software engineering as:

"Software engineering is the establishment and use of sound engineering principlesinordertoobtaineconomicallysoftwarethatisreliableandwork efficiently on realmachines."

What Is A Computer Program?

A computer program is a collection of instructions that performs a specific task when executed by a computer. Most computer devices require programs to function properly. A computer program is usually written by a computer programmer in a programming language. Once it is written, the programmer uses a compiler to turn it into a language that the computer can understand.

A computer program is stored as a file on the computer's hard drive. When the user runs the program, the file is read by the computer, and the processor reads the data in the file as a list of instructions. Then the computer does what the program tells it to do.

What You Need To Know About Program

1. Program is a set of instructions written in a programming language used to execute for a specific task or particular function.
2. A program does not have further categorization.
3. A program cannot be software.
4. A program consists of a set of instructions which are coded in a programming language like c, C++, PHP, Java etc.
5. Programs do not have a user interface.
6. A program is developed and used by either a single programmer or a group of programmers.
7. A program is compiled every time when we need to generate some output from it.
8. Program has limited functionality and less features.
9. Program functionality is dependent on compiler.
10. A program takes less time to build/make.
11. Program development approach is un-procedural, un-organized and unplanned.
12. The size of a program ranges from kilobytes (Kb) to megabytes (Mb).

What Is A Computer Software?

Computer software popularly referred to as software, is a set of instructions, data or programs used to operate computers and execute specific tasks. It includes all programs on a computer such as applications and the operating system. Software is often divided into three categories:

- **Application software**. This is intended to perform certain tasks. Examples of application software include office suites, gaming applications, database systems and educational software.
- **Programming Software**. Programming software is a set of tools to aid developers in writing programs. The various tools available are compilers, linkers, debuggers, interpreters and text editors.
- **System Software.** System software act as a base for application software. Examples include device drivers, operating systems, compilers, disk formatters, text editors and utilities helping the computer to operate more efficiently. System software is usually written in C programming language.

What You Need To Know About Software

1. Software is a collection of several programs and other procedures and documentation.
2. Software can be categorized into three types: application software, system software and utilities.
3. Software can be a program.
4. Software consists of bundles of programs and data files. Programs in specific software use these data files to perform a dedicated type of tasks.
5. Every software has a dedicated user interface. The user interface of software may be in the form of a command prompt or in a graphical format.
6. Software is developed by either a single programmer or a group of programmers.
7. Whole software is compiled, tested and debugged during the development process.
8. Software has lots of functionality and features such as GUI, input/output data, process etc.
9. Software functionality is dependent on the operating system.
10. Software takes relatively more time to build/make when compared to program.
11. Software development approach is systematic, organized and very well planned.
12. The size of a software ranges from megabytes (Mb) to Gigabytes (Gb).
13. Examples of software include: Microsoft Word, Microsoft Excel, VLC media player, Firefox, Adobe Reader, Windows, Linux, Unix, Mac etc.

Software is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called software product.

Engineering on the other hand, is all about developing products, using well-defined, scientific principles and methods.

Software engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

Definitions

IEEE defines software engineering as:

(1) The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software.

(2) The study of approaches as in the above statement.

Fritz Bauer, a German computer scientist, defines software engineering as:

Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and work efficiently on real machines.

Need of Software Engineering

The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working.

- Large software - It is easier to build a wall than to a house or building, likewise, as the size of software become large engineering has to step to give it a scientific process.
- Scalability- If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one.
- Cost- As hardware industry has shown its skills and huge manufacturing has lower down he price of computer and electronic hardware. But the cost of software remains high if proper process is not adapted.
- Dynamic Nature- The always growing and adapting nature of software hugely depends upon the environment in which user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where software engineering plays a good role.
- Quality Management- Better process of software development provides better and quality software product.

Characteristics of good software

A software product can be judged by what it offers and how well it can be used. This software must satisfy on the following grounds:

- Operational
- Transitional
- Maintenance

Well-engineered and crafted software is expected to have the following characteristics:

Operational

This tells us how well software works in operations. It can be measured on:

- Budget
- Usability
- Efficiency
- Correctness

Functionality

Dependability

Security

Safety

Transitional

This aspect is important when the software is moved from one platform to another:

Portability

Interoperability

Reusability

Adaptability

Maintenance

This aspect briefs about how well a software has the capabilities to maintain itself in the ever-changing environment:

Modularity

Maintainability

Flexibility

Scalability

In short, Software engineering is a branch of computer science, which uses well-defined engineering concepts required to produce efficient, durable, scalable, in-budget and on-time software products.

The Importance of Software Engineers

Software engineers of all kinds, full-time staff, vendors, contracted workers, or part-time workers, are important members of the IT community.

What do software engineers do? Software engineers apply the principles of software engineering to the design, development, maintenance, testing, and evaluation of software. There is much discussion about the degree of education and or certification that should be required for software engineers.

According to StackOverflow Survey 2018, software engineers are lifelong learners; almost 90% of all developers say they have taught themselves a new language, framework, or tool outside of their formal education.

Software engineers are well versed in the software development process, though they typically need input from IT leader regarding software requirements and what the end result needs to be. Regardless of formal education, all software engineers should work within a specific set of best practices for software engineering so that others can do some of this work at the same time. Software engineering almost always includes a vast amount of teamwork. Designers, writers, coders, testers, various team members, and the entire IT team need to understand the code.

Software engineers should understand how to work with several common computer languages, including Visual Basic, Python, Java, C, and C++. According to Stackoverflow, for the sixth

year in a row, JavaScript is the most commonly used programming language. Python has risen in the ranks, surpassing C# this year, much like it surpassed PHP last year. Python has a solid claim to being the fastest-growing major programming language.

Software engineering is important because specific software is needed in almost every industry, in every business, and for every function. It becomes more important as time goes on – if something breaks within your application portfolio, a quick, efficient, and effective fix needs to happen as soon as possible.

Whatever you need software engineering to do – it is something that is vitally important and that importance just keeps growing. When you work with software engineers, you need to have a check and balance system to see if they are living up to their requirements and meeting KPIs.

SoftwareEvolution

The process of developing a software product using software engineering principles and methods is referred to as **Software Evolution.** This includes the initial development of software and its maintenance and updates, till desired software product is developed, which satisfies the expected requirements.

Evolution starts from the requirement gathering process. After which developers create a prototype of the intended software and show it to the users to get their feedback at the early stage of the software product development. The users suggestchanges,onwhichseveralconsecutiveupdatesandmaintenancekeepon changing too. This process changes to the original software, till the desired software isaccomplished.

Even after the user has the desired software in hand, the advancing technology andthechangingrequirementsforcethesoftwareproducttochangeaccordingly. Re-creating software from scratch and to go one-on-one with the requirementis

not feasible. The only feasible and economical solution is to update the existing software so that it matches the latest requirements.

Software EvolutionLaws

Lehmanhasgivenlawsforsoftwareevolution.Hedividedthesoftwareintothree differentcategories:

1. **Static-type (S-type) -** This is a software, which works strictly according to defined specifications and solutions. The solution and the method to achieve it, both are immediately understood before coding. The s-type software is least subjected to changes hence this is the simplest of all. For example, calculator program for mathematicalcomputation.
2. **Practical-type (P-type) -** This is a software with a collection of procedures.This is defined by exactly what procedures can do. In this software, the specifications can be described but the solution is not obviously instant. For example, gamingsoftware.
3. **Embedded-type (E-type) -** This software works closely as the requirement of real-world environment. This software has a high degreeof evolution as there are various changes in laws, taxes etc. in the real world situations. For example, Online tradingsoftware.

E-Type softwareevolution

Lehman has given eight laws for E-Type software evolution -

1. **Continuing change -** An E-type software system must continue to adapt to the real world changes, else it becomes progressively lessuseful.
2. **Increasing complexity -** As an E-type software system evolves, its complexity tends to increase unless work is done to maintain or reduceit.
3. **Conservation of familiarity -** The familiarity with the software or the knowledge about how it was developed, why was it developed in that particular manner etc., must be retained at any cost, to implement the changes in thesystem.
4. **Continuing growth-** In order for an E-type system intended to

resolve some business problem, its size of implementing the changes grows according to the lifestyle changes of thebusiness.

5. **Reducing quality -** An E-type software system declines in quality unless rigorously maintained and adapted to a changing operationalenvironment.
6. **Feedback systems-** The E-type software systems constitute multi-loop, multi-levelfeedbacksystemsandmustbetreatedassuchtobesuccessfully modified orimproved.
7. **Self-regulation -** E-type system evolution processes are self-regulating with the distribution of product and process measures close tonormal.
8. **Organizational stability -** The average effective global activity rate inan evolving E-type system is invariant over the lifetime of the product.

Software Paradigms

Software paradigms refer to the methods and steps, which are taken while designingthesoftware.Therearemanymethodsproposedandareimplemented. But, we need to see where in the software engineering concept, these paradigms stand. These can be combined into various categories, though each of them is contained in oneanother:

Programming paradigm is a subset of Software design paradigm which is further a subset of Software development paradigm.

Software Development Paradigm

This paradigm is known as software engineering paradigms; where all the engineering concepts pertaining to the development of software are applied. It includes various researches and requirement gathering which helps the software product to build. It consists of –

- Requirementgathering
- Softwaredesign
- Programming

Software Design Paradigm

This paradigm is a part of Software Development and includes –

- Design
- Maintenance
- Programming

Programming Paradigm

This paradigm is related closely to programming aspect of software development. This includes –

- Coding
- Testing
- Integration

Need of SoftwareEngineering

The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working. Following are some of the needs stated:

- **Large software -** It is easier to build a wall than a house or building, likewise,asthesizeofthesoftwarebecomeslarge,engineeringhastostep to give it a scientificprocess.
- **Scalability-** If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existingone.
- **Cost-** As hardware industry has shown its skills and huge manufacturing haslowerdownthepriceofcomputerandelectronichardware.But,costof the software remains high if proper process is notadapted.
- **Dynamic Nature-** Always growing and adapting nature of the software hugely depends upon the environment in which the user works. If the

natureofsoftwareisalwayschanging,newenhancementsneedtobedone in the existing one. This is where the software engineering plays a good role.

- **Quality Management-** Better process of software development provides better and quality softwareproduct.

Characteristics of goodsoftware

A software product can be judged by what it offers and how well it can be used. This software must satisfy on the following grounds:

- Operational
- Transitional
- Maintenance

Well-engineered and crafted software is expected to have the following characteristics:

Operational

This tells us how well the software works in operations. It can be measured on:

- Budget
- Usability
- Efficiency
- Correctness
- Functionality
- Dependability
- Security
- Safety

Transitional

This aspect is important when the software is moved from one platform to another:

- Portability
- Interoperability
- Reusability
- Adaptability

Maintenance

This aspect briefs about how well the software has the capabilities to maintain itself in the ever-changing environment:

- Modularity

- Maintainability
- Flexibility
- Scalability

In short, Software engineering is a branch of computer science, which uses well- defined engineering concepts required to produce efficient, durable, scalable, in- budget, and on-time software products.

Software Characteristics Softwareis:

- Instructions (computer programs) that when executed provide desired function and performance.
- Data structures that enable the programs to adequately manipulateinformation.
- Documents that describe the operation and use of theprograms.

To gain an understanding of software, it is important to examine the characteristics of software that make it different from other things that human beings build. Software isa logical rather than a physical system element. Therefore, software has characteristics that are considerably different than those ofhardware:

- Software is developed orengineered.
- Software doesn't "wearout."
- Although the industry is moving toward component-based assembly, most software continues to be custombuilt.

1. Software is developed or engineered; it is not manufactured in the classicalsense.

Although some similarities exist between software development and hardware manufacture, the two activities are fundamentally different. In both activities, high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems that are nonexistent (or easily corrected) for software. Both activities are dependent on people, but the relationship between people applied and work accomplished is entirely different. Both activities require the construction of a —productǁ, but the approaches are different. Software costs are concentrated in engineering. This means that software projects cannot be managed as if they were manufacturingprojects.

2. Software doesn't "wearout."

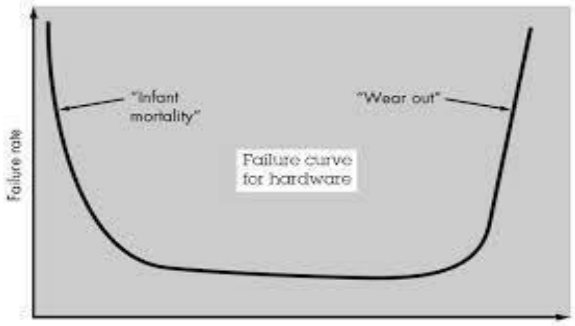


Fig: 1.2 Failure Curve for Hardware

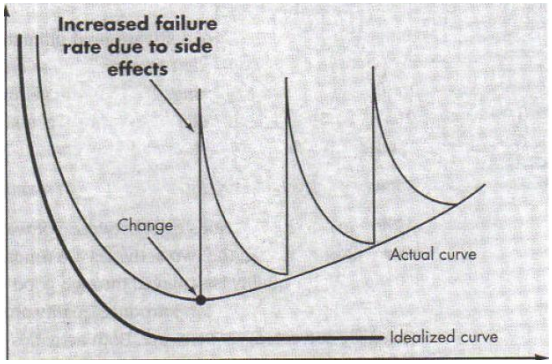


Fig: 1.3 Failure Curve for Software

The relationship, often called the "bathtub curve," indicates that hardware exhibits relatively high failure rates early in its life (these failures are often attributable to design or manufacturing defects); defects are corrected and the failure rate drops to a steady-state level (ideally, quite low) for some period of time. As time passes, however, the failure rate rises again as hardware components suffer from the cumulative effects of dust, vibration, abuse, temperature extremes, and many other environmental maladies. Stated simply, the hardware begins to wear out. Software is not susceptible to the environmental maladies that cause hardware to wear out

3. Although the industry is moving toward component-based assembly, most software continues to be custombuilt.

Consider the manner in which the control hardware for a computer-based product is designed and built. The design engineer draws a simple schematic of the digital circuitry, does some fundamental analysis to assure that proper function will be achieved, and then goes to the shelf where catalogs of digital components exist. Each integrated circuit (called an IC or a chip) has a part number, a defined

and validated function, a well-defined interface, and a standard set of integration guidelines. After each component is selected, it can be ordered off theshelf.

Principles of Software Engineering

Separation of Concerns

Separation of concerns is a recognition of the need for human beings to work within a limited context. As descibed by G. A. Miller [Miller56], the human mind is limited to dealing with approximately seven units of data at a time. A unit is something that a person has learned to deal with as a whole - a single abstraction or concept. Although human capacity for forming abstractions appears to be unlimited, it takes time and repetitive use for an abstraction to become a useful tool; that is, to serve as a unit.

When specifying the behavior of a data structure component, there are often two concerns that need to be dealt with: basic functionality and support for data integrity. A data structure component is often easier to use if these two concerns are divided as much as posible into separate sets of client functions. It is certainly helful to clients if the client documentation treats the two concerns separately. Further, implementation documentation and algorithm descriptions can profit from separate treatment of basic algorithms and modifications for data integrity and exception handling.

There is another reason for the importance of separation of concerns. Software engineers must deal with complex values in attempting to optimize the quality of a product. From the study of algorithmic complexity, we can learn an important lesson. There are often efficient algorithms for optimizing a single measurable quantity, but problems requiring optimization of a combination of quantities are almost always NP-complete. Although it is not a proven fact, most experts in complexity theory believe

that NP-complete problems cannot be solved by algorithms that run in polynomial time.

In view of this, it makes sense to separate handling of different values. This can be done either by dealing with different values at different times in the software development process, or by structuring the design so that responsibility for achieving different values is assigned to different components.

As an example of this, run-time efficiency is one value that frequently conflicts with other software values. For this reason, most software engineers recommend dealing with efficiency as a separate concern. After the software is design to meet other criteria, it's run time can be checked and analysed to see where the time is being spent. If necessary, the portions of code that are using the greatest part of the runtime can be modified to improve the runtime. This idea is described in depth in Ken Auer and Kent Beck's article "Lazy optimization: patterns for efficient smalltalk programming" in [VCK96, pp 19-42].

Modularity

The principle of modularity is a specialization of the principle of separation of concerns. Following the principle of modularity implies separating software into components according to functionality and responsibility. Parnas [Parnas72] wrote one of the eariest papers discussing the considerations involved in modularization. A more recent work, [WWW90], describes a responsibility-driven methodology for modularization in an object-oriented context.

Abstraction

The principle of abstraction is another specialization of the principle of separation of concerns. Following the principle of abstraction implies separating the behavior of software components from their implementation. It requires learning to look at software and software components from two points of view: what it does, and how it does it.

Failure to separate behavior from implementation is a common cause of unnecessary coupling. For example, it is common in recursive algorithms to introduce extra parameters to make the recursion work. When this is done, the recursion should be called through a non-recursive shell that provides the proper initial values for the

extra parameters. Otherwise, the caller must deal with a more complex behavior that requires specifying the extra parameters. If the implementation is later converted to a non-recursive algorithm then the client code will also need to be changed.

Design by contract is an important methodology for dealing with abstraction. The basic ideas of design by contract are sketched by Fowler and Scott [FS97]. The most complete treatment of the methodology is given by Meyer [Meyer92a].

Anticipation of Change

Computer software is an automated solution to a problem. The problem arises in some context, or domain that is familiar to the users of the software. The domain defines the types of data that the users need to work with and relationships between the types of data.

Software developers, on the other hand, are familiar with a technology that deals with data in an abstract way. They deal with structures and algorithms without regard for the meaning or importance of the data that is involved. A software developer can think in terms of graphs and graph algorithms without attaching concrete meaning to vertices and edges.

Working out an automated solution to a problem is thus a learning experience for both software developers and their clients. Software developers are learning the domain that the clients work in. They are also learning the values of the client: what form of data presentation is most useful to the client, what kinds of data are crucial and require special protective measures.

The clients are learning to see the range of possible solutions that software technology can provide. They are also learning to evaluate the possible solutions with regard to their effectiveness in meeting the clients needs.

If the problem to be solved is complex then it is not reasonable to assume that the best solution will be worked out in a short period of time. The clients do, however, want a timely solution. In most cases, they are not willing to wait until the perfect solution is worked out. They want a reasonable solution soon; perfection can come later. To develop a timely solution, software developers need to know the

requirements: how the software should behave. The principle of acticipation of change recognizes the complexity of the learning process for both software developers and their clients. Preliminary requirements need to be worked out early, but it should be possible to make changes in the requirements as learning progresses.

Coupling is a major obstacle to change. If two components are strongly coupled then it is likely that changing one will not work without changing the other.

Cohesiveness has a positive effect on ease of change. Cohesive components are easier to reuse when requirements change. If a component has several tasks rolled up into one package, it is likely that it will need to be split up when changes are made.

Generality

The principle of generality is closely related to the principle of anticipation of change. It is important in designing software that is free from unnatural restrictions and limitations. One excellent example of an unnatural restriction or limitation is the use of two digit year numbers, which has led to the "year 2000" problem: software that will garble record keeping at the turn of the century. Although the two-digit limitation appeared reasonable at the time, good software frequently survives beyond its expected lifetime.

For another example where the principle of generality applies, consider a customer who is converting business practices into automated software. They are often trying to satisfy general needs, but they understand and present their needs in terms of their current practices. As they become more familiar with the possibilities of automated solutions, they begin seeing what they need, rather than what they are currently doing to satisfy those needs. This distinction is similar to the distinction in the principle of abstraction, but its effects are felt earlier in the software development process.

Incremental Development

Fowler and Scott [FS97] give a brief, but thoughtful, description of an incremental software development process. In this process, you build the software in small increments; for example, adding one use case at a time.

An incremental software development process simplifies verification. If you develop software by adding small increments of functionality then, for verification, you only need to deal with the added portion. If there are any errors detected then they are already partly isolated so they are much easier to correct.

A carefully planned incremental development process can also ease the handling of changes in requirements. To do this, the planning must identify use cases that are most likely to be changed and put them towards the end of the development process.

Consistency

The principle of consistency is a recognition of the fact that it is easier to do things in a familiar context. For example, coding style is a consistent manner of laying out code text. This serves two purposes. First, it makes reading the code easier. Second, it allows programmers to automate part of the skills required in code entry, freeing the programmer's mind to deal with more important issues.

At a higher level, consistency involves the development of idioms for dealing with common programming problems. Coplien [Coplien92] gives an excellent presentation of the use of idioms for coding in C++.

Consistency serves two purposes in designing graphical user interfaces. First, a consistent look and feel makes it easier for users to learn to use software. Once the basic elements of dealing with an inteface are learned, they do not have to be relearned for a different software application. Second, a consistent user interface promotes reuse of the interace components. Graphical user interface systems have a collection of frames, panes, and other view components that support the common look. They also have a collection of controllers for responding to user input, supporting the common feel. Often, both look and feel are combined, as in pop-up menus and buttons. These components can be used by any program.

Meyer [Meyer94c] applies the principle of consistency to object-oriented class libraries. As the available libraries grow more and more complex it is essential that they be designed to present a consistent interface to the client. For example, most data collection structures support adding new data items. It is much easier to learn to use the collections if the name add is always used for this kind of operation.

- **Difference Between S/w Programming and S/w Engineering**

Programming is primarily a personal activity, mostly done as an individual	Software engineering involves team activity
The programmer writes the entire program or develops code	The Software Engineer architects the software components, combined with components to build a system
Specializes in coding and has required technical skills to create a merchandise	Specializes in a scientific method of operations with the stakeholders and develop solutions
Programming has a wide approach towards the concepts of computer and applications	Crafts the Product with Quality Cautious attitude
Works on just one aspect of software development	Works on Large software systems needs to be developed, which is similar to other modern Engineering practices

- **Software Engineering Team Members**

1 PROJECT SPONSOR
2 SUBJECT MATTER EXPERTS (SME)
3 PRODUCT OWNER
4 PROJECT MANAGER (PM)
5 TECHNICAL LEAD
6 SOFTWARE DEVELOPERS
7 SOFTWARE TESTERS
8 USER ACCEPTANCE TESTERS

Software projects are difficult and they all take careful planning, a talented development team and collaboration of a project's team members, both internally within the company and externally with the software development company.

Software projects can only move forward when the key stakeholders are all in place.

One of the keys to a successful software project is identifying and documenting the software project roles and responsibilities for your project. You'll need to ensure that you define the key stakeholders within your business that will be involved in the delivery of the software solution.

Get the right people. Then no matter what all else you might do wrong after that, the people will save you. That's what management is all about.
-– Tom DeMarco

Among the key stakeholders of a software project are the following eight key roles in software development and their corresponding responsibilities.

PROJECT SPONSOR

Project Sponsors play a critical role in all projects. Project sponsors have the bandwidth to take on the Project Sponsor role, their day job and no other project role, therefore Project Sponsors are not Project Managers, Scrum Masters or Product Owners.

Unengaged sponsor sinks the ship.
-– Angela Waner

The Project Sponsor is the person or group that provides direction and resources, including financial resources for the software project. The Project Sponsor works with the project management team, aiding with wider project matters such as scope clarification, progress, monitoring, and influencing others in order to benefit the software project.

The Project Sponsor leads the project through the software supplier selection process until it is formally authorised. For issues that are beyond the control of the Product Owner, the Project Sponsor serves as an escalation path.

The Project Sponsor may also be involved in other important issues such as authorising changes in scope, phase-end reviews, and go/no-go decisions when the stakes of the project are particularly high.

Typically sponsors of projects tend to be senior management or director level executives.

SUBJECT MATTER EXPERTS (SME)

A Subject Matter Expert (SME) or Domain Expert is a person who is an authority in a particular area or topic. A Subject Matter Expert has superior (expert) knowledge of a discipline, technology, product, business process or entire business area.

The SME role and responsibilities in software development could be summarised as follows: they are normally the people from whom technical requirements are captured.

If everyone is thinking alike, someone isn't thinking.
-– General George Patton Jr.

Subject Matter Experts are the accountants, finance controllers, salespeople, production managers and so on who roll up their sleeves each day. They know their roles inside and out and are rarely technical.

However, their lack of technical knowledge is their strength, as they are not bogged down in technicalities and instead are purely focused on business outcomes.

It's imperative that discussions are held with Subject Matter Experts at the same time as the software product vision statement is being created. Feedback from this group of experts can save a lot of back and forth down the line.

However, given that Subject Matter Experts tend not to be technical the right amount and type of engagement are necessary so as not to overwhelm them. One of the ways to get them involved is to have them contribute to the creation of early-stage wireframes and prototypes.

PRODUCT OWNER

Product Owner is a software development role for a person who represents the business or end-users and is responsible for working with the user group to determine what features will be in the product release.

The Product Owner is also responsible for the prioritised backlog and maximising the return on investment (ROI) of the software project. Part of this role's responsibility includes documenting user stories or requirements for the software project.

They act as the main point of contact for all decisions concerning the project and as such, need to be empowered to perform their responsibilities without the need to seek too much prior authorisation from the Project Sponsors.

Appointing the right person to this role, with the appropriate delegated authority to progress the project, is fundamental to the success of the project, especially if an agile methodology approach is undertaken.

In particular, the Product Owner is responsible for:

- ensuring that the software product vision statement is adhered to
- making the final decision on all scope related decisions
- maintaining and updating the product backlog on a continuous basis by
    - refining new requirements
    - removing requirements that fall out of scope
    - adding new requirements identified as being required to achieve the software product vision statement
    - reviewing and setting the priorities assigned to the product backlog and heading up all project planning meetings
- resolving any disputes either with the software development team or internally

Failure to have a Product Owner in place usually means that the software project will execute in fits and starts whilst the software developers are on hold waiting for crucial feedback.

A slowdown in the momentum of a software project can have long-term consequences, not least of missed milestones and deadlines. Don't ever underestimate the importance of the Product Owner role in the success of your software development project.

PROJECT MANAGER (PM)

The Project Manager (PM) is responsible for knowing the "who, what, where, when and why" of the software project. This means knowing the stakeholders of the project and being able to effectively communicate with each of them.

The Project Manager is also responsible for creating and managing the project budget and schedule as well as processes including scope management, issues management and risk management.

Some of the Project Manager duties can include:

- Developing a software project plan
- Manage deliverables according to the software project plan
- Recruiting software project staff
- Leading and managing the software project team
- Determining the methodology used on the project
- Establishing a project schedule and determine each phase
- Assigning tasks to project team members
- Providing regular updates to senior management

It doesn't matter if you are using an agile methodology or the waterfall method, once deliverables are defined, it is critical that the Project Manager starts to actively exercise change management. Change should not be perceived as negative or something to be avoided.

Change is inevitable and is acceptable in a software project as long as it is managed. The impact of any change needs to be assessed, measured and communicated. The

major factors are typically timeline and budget. If the impact is deemed acceptable by the Project Sponsor, then the change can be incorporated.

The Project Manager also oversees software testing, delivery and formal acceptance by the customer. Then the Project Manager performs a project review with the software development team to document any lessons learned from the software development processes.

TECHNICAL LEAD

This person translates the business requirements into a technical solution. Because of this responsibility, it is beneficial to have the Technical Lead involved in the planning phase to hear the business requirements from the customer's point of view and ask questions.

The Technical Lead is the development team leader and works with the developers to provide technical details and estimates for the proposed solution. This information is used by the Project Manager to create the Statement of Work and the Work Breakdown Structure documents for the software project.

It is critical that the Technical Lead can effectively communicate the status of the software project to the Project Manager so that issues or variances can be effectively addressed as soon as possible.

The Technical Lead is also responsible for establishing and enforcing standards and practices with the software development team.

SOFTWARE DEVELOPERS

The Software Developers (front-end and back-end) are responsible for using the technical requirements from the Technical Lead to create cost and timeline estimates.

The Software Developers are also responsible for building the deliverables and communicating the status of the software project to the Technical Lead or Project Manager.

It is critical that the other team members effectively communicate the technical requirements to the Software Developers to reduce project risk and provide the software project with the greatest chance of success.

SOFTWARE TESTERS

The Software Testers ensure that the software solution meets the business requirements and that it is free of bugs, errors and defects.

In the test planning and preparation phases of the software testing, Software Testers should review and contribute to test plans, as well as be analysing, reviewing and assessing technical requirements and design specifications.

Software Testers are involved in identifying test conditions and creating test designs, test cases, test procedure specifications and test data, and may automate or help to automate the tests.

Some of the Software Testers duties can include:

- They often set up the test environments or assist system administration and network management staff in doing so
- As test execution begins, the number of testers often increases, starting with the work required to implement tests in the test environment
- Testers execute and log the tests, evaluate the results and document problems found
- They monitor the testing and the test environment, often using tools for this task, and often gather performance metrics
- Throughout the software testing life cycle, they review each other's work, including test specifications, defect reports and test results

USER ACCEPTANCE TESTERS

You should expect your software solution provider to carry out a wide array of software testing to ensure that your new software solution meets various quality assurance (QA) criteria.

On from that, representatives of your company will need to perform the final checks to ensure that the software works for the business across a number of real-world scenarios.

User Acceptance Testing (UAT) is the final step prior to a new software solution being released to production (live). It's absolutely essential that you have the resources to tackle user acceptance testing in a timely and organised fashion, as it is often UAT that creates the bottleneck between the software solution being completed and released to the business.

It's often the case that the aforementioned Subject Matter Experts defined how the new software solution should work and, given their close proximity to the actual work, they can make excellent User Acceptance Testers.

When end users get involved in the final stages of testing, light bulbs go on, and they often have an "aha" moment. Unfortunately, that is often too late.
-– Frank R. Parth

It's an excellent idea to ensure that those employees participating in UAT are brought in from the start, and understand, or perhaps better still contribute to, the design of the new software solution.

This emotional buy-in and understanding of the software solution's objectives reduces the friction that might otherwise exist in attempting to move end-users from the existing software systems they know, love and use every day.

SOFTWARE DEVELOPMENT ROLES – CONCLUSION

Whilst it's important that your software solution provider has the necessary resources in place to operate your project, it is equally as important that you as the customer understand the roles and responsibilities required within your team to bring your project to successful completion.

The key to project success is clear and effective communication. A critical portion of this communication is identifying the stakeholders and their roles.

Whatever labels you apply to the software project roles above, clear communication of expectations and status to the stakeholders throughout the life of the software project will increase the chances of your project's success.

Unit – 2

- **Software Development Cycle**
- General Software development life cycle
- Comparison between waterfall, prototyping and spiral model
- Comparative Study of Incremental Model & RAD Model
- Component Based Development
- Fourth Generation Techniques

Feasibility Study

- Need of Feasibility Study
- Types of Feasibility

Cost Benefit Analysis

- Why Cost Benefit Analysis?
- Cost Benefit Analysis Process

Software Development Life Cycle

Software Development Life Cycle, SDLC for short, is a well-defined, structured sequence of stages in software engineering to develop the intended software product.

SDLC Activities

SDLC provides a series of steps to be followed to design and develop a software product efficiently. SDLC framework includes the following steps:

Communication

This is the first step where the user initiates the request for a desired software product. The user contacts the service provider and tries to negotiate the terms, submits the request to the service providing organization in writing.

Requirement Gathering

This step onwards the software development team works to carry on the project. The team holds discussions with various stakeholders from problem domain and tries to bring

out as much information as possible on their requirements. The requirements are contemplated and segregated into user requirements, system requirements and functional requirements. The requirements are collected using a number of practices as given -

- studying the existing or obsolete system andsoftware,
- conducting interviews of users anddevelopers,
- referring to the databaseor
- collecting answers from thequestionnaires.

Feasibility Study

After requirement gathering, the team comes up with a rough plan of software process. At this step the team analyzes if a software can be designed to fulfill all requirementsoftheuser,andifthereisanypossibilityofsoftwareb eingnomore useful. It is also analyzed if the project is financially, practically, and technologicallyfeasiblefortheorganizationtotakeup.Thereare manyalgorithms available, which help the developers to conclude the feasibility of a software project.

System Analysis

Atthisstepthedevelopersdecidearoadmapoftheirplanandtrytob ringupthe best software model suitable for the project. System analysis includes understanding of software product limitations, learning system related problems orchangestobedoneinexistingsystemsbeforehand,identifyinga ndaddressing theimpactofprojectonorganizationandpersonneletc.Theprojec tteamanalyzes the scope of the project and plans the schedule and resourcesaccordingly.

Software Design

Next step is to bring down whole knowledge of requirements and analysis on the desk and design the software product. The inputs from users and information gathered in requirement gathering phase are the inputs of this step. The output ofthisstepcomesintheformoftwodesigns;logicaldesign,andphy sicaldesign. Engineers produce meta-data and data dictionaries, logical diagrams, data-flow diagrams, and in some cases pseudocodes.

Coding

This step is also known as programming phase. The implementation of software design starts in terms of writing program code in the suitable programming language and developing error-free executable programs efficiently.

Testing

An estimate says that 50% of whole software development process should be tested. Errors may ruin the software from critical level to its own removal. Software testing is done while coding by the developers and thorough testing is conducted by testing experts at various levels of code such as moduletesting,
program testing, product testing, in-house testing, and testing the product at user's end. Early discovery of errors and their remedy is the key to reliable software.

Integration

Software may need to be integrated with the libraries, databases, and other program(s). This stage of SDLC is involved in the integration of software with outer world entities.

Implementation

This means installing the software on user machines. At times, software needs post-installation configurations at user end. Software is tested for portability and adaptability and integration related issues are solved during implementation.

Operation and Maintenance

This phase confirms the software operation in terms of more efficiency and less errors. If required, the users are trained on, or aided with the documentation on how to operate the software and how to keep the software operational. The software is maintained timely by updating the code according to the changes taking place in user end environment or technology. This phase may face challenges from hidden bugs and real-world unidentified problems.

Software DevelopmentParadigm

The software development paradigm helps a developer to

select a strategy to developthesoftware.Asoftwaredevelopmentparadigmhasitso wnsetoftools, methods, and procedures, which are expressed clearly and defines software development life cycle. A few of software development paradigms or process models are defined asfollows:

Waterfall Model

Waterfall model is the simplest model of software development paradigm. All the phasesofSDLCwillfunctiononeafteranotherinlinearmanner.Tha tis,whenthe first phase is finished then only the second phase will start and soon.

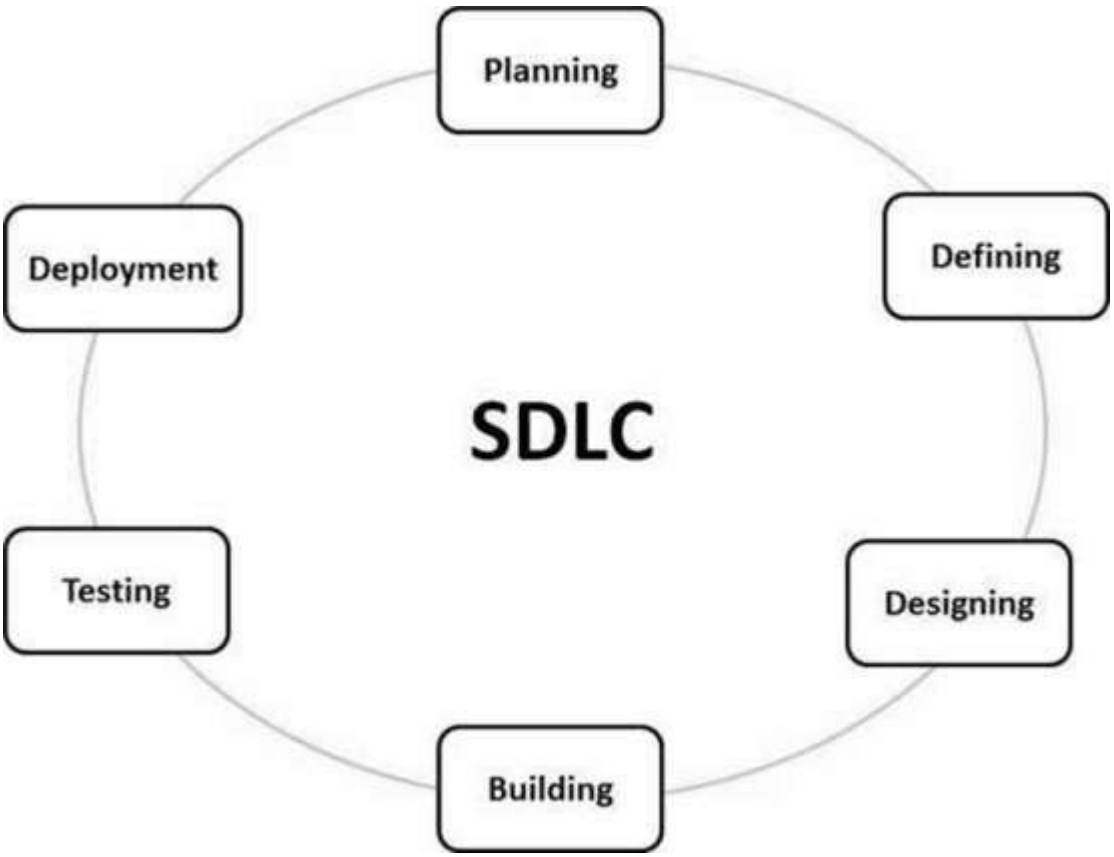
Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality softwares. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

- SDLC is the acronym of Software Development Life Cycle.
- It is also called as Software Development Process.
- SDLC is a framework defining tasks performed at each step in the software development process.
- ISO/IEC 12207 is an international standard for software life-cycle processes. It aims to be the standard that defines all the tasks required for developing and maintaining software.

What is SDLC?

SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.

The following figure is a graphical representation of the various stages of a typical SDLC.



A typical Software Development Life Cycle consists of the following stages −

Stage 1: Planning and Requirement Analysis

Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational and technical areas.

Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.

Stage 2: Defining Requirements

Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through an **SRS (Software Requirement Specification)** document which consists of all the product requirements to be designed and developed during the project life cycle.

Stage 3: Designing the Product Architecture

SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification.

This DDS is reviewed by all the important stakeholders and based on various parameters as risk assessment, product robustness, design modularity, budget and time constraints, the best design approach is selected for the product.

A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third party modules (if any). The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of the details in DDS.

Stage 4: Building or Developing the Product

In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.

Developers must follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers, etc. are used to generate the code. Different high level programming languages such as C, C++, Pascal, Java and PHP are used for coding. The programming language is chosen with respect to the type of software being developed.

Stage 5: Testing the Product

This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However, this stage refers to the testing only stage of the product where product defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS.

Stage 6: Deployment in the Market and Maintenance

Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that organization. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing).

Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base.

SDLC Models

There are various software development life cycle models defined and designed which are followed during the software development process. These models are also referred as Software Development Process Models". Each process model follows a Series of steps unique to its type to ensure success in the process of software development.

Following are the most important and popular SDLC models followed in the industry −

- Waterfall Model
- Iterative Model
- Spiral Model

- V-Model
- Big Bang Model

Other related methodologies are Agile Model, RAD Model, Rapid Application Development and Prototyping Models.

SDLC - Waterfall Model

The Waterfall Model was the first Process Model to be introduced. It is also referred to as a **linear-sequential life cycle model**. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.

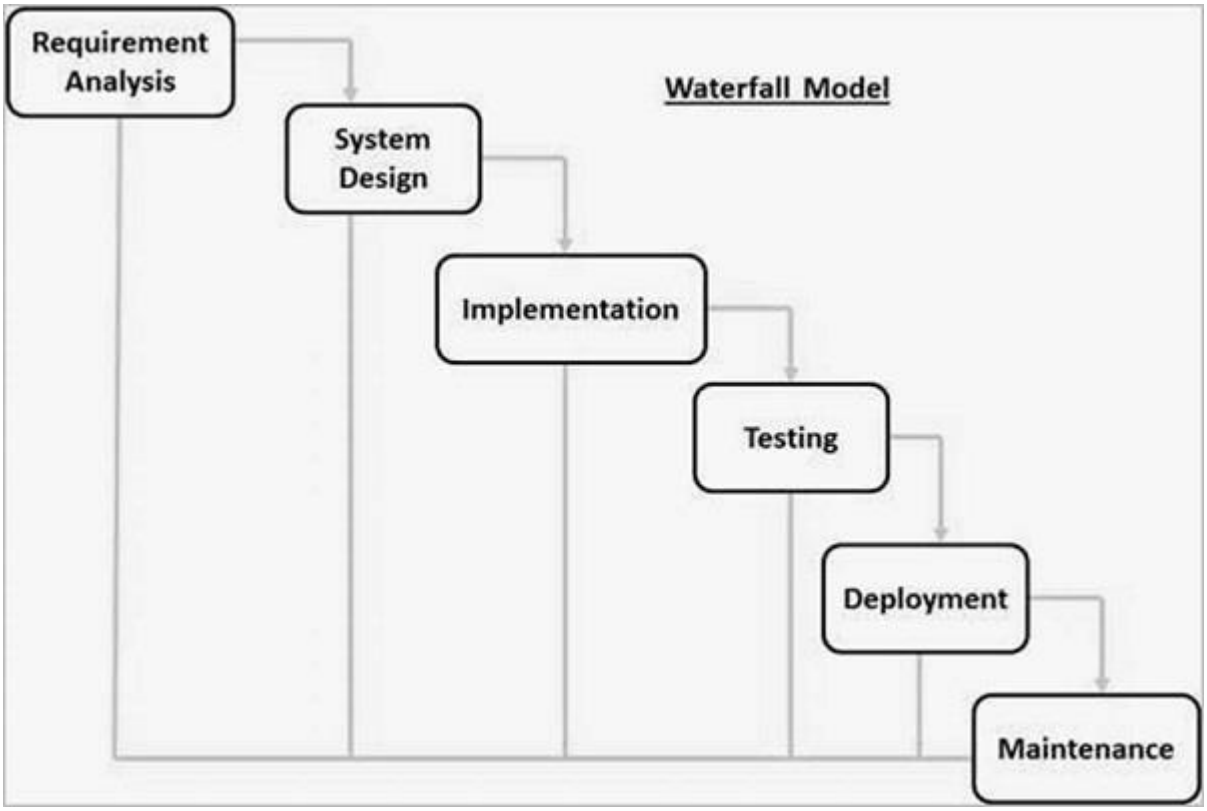
The Waterfall model is the earliest SDLC approach that was used for software development.

The waterfall Model illustrates the software development process in a linear sequential flow. This means that any phase in the development process begins only if the previous phase is complete. In this waterfall model, the phases do not overlap.

Waterfall Model - Design

Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

The following illustration is a representation of the different phases of the Waterfall Model.



The sequential phases in Waterfall model are −

- **Requirement Gathering and analysis** − All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.
- **System Design** − The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.
- **Implementation** − With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.
- **Integration and Testing** − All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
- **Deployment of system** − Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.
- **Maintenance** − There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model". In this model, phases do not overlap.

Waterfall Model - Application

Every software developed is different and requires a suitable SDLC approach to be followed based on the internal and external factors. Some situations where the use of Waterfall model is most appropriate are −

- Requirements are very well documented, clear and fixed.
- Product definition is stable.
- Technology is understood and is not dynamic.
- There are no ambiguous requirements.
- Ample resources with required expertise are available to support the product.
- The project is short.

Waterfall Model - Advantages

The advantages of waterfall development are that it allows for departmentalization and control. A schedule can be set with deadlines for each stage of development and a product can proceed through the development process model phases one by one.

Development moves from concept, through design, implementation, testing, installation, troubleshooting, and ends up at operation and maintenance. Each phase of development proceeds in strict order.

Some of the major advantages of the Waterfall Model are as follows −

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.

Waterfall Model - Disadvantages

The disadvantage of waterfall development is that it does not allow much reflection or revision. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-documented or thought upon in the concept stage.

The major disadvantages of the Waterfall Model are as follows −

- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.
- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- Adjusting scope during the life cycle can end a project.
- Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

SDLC - Iterative Model

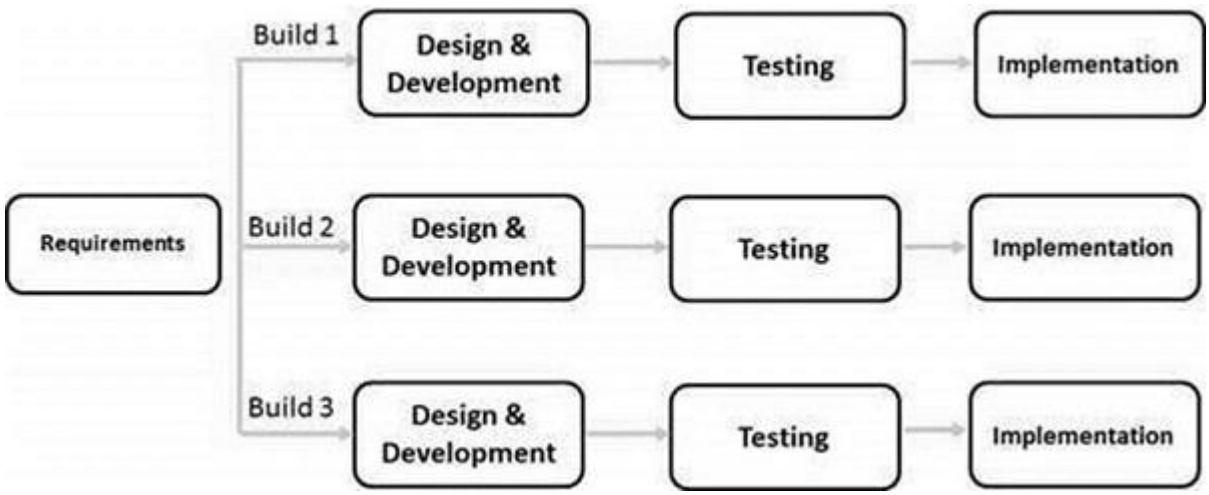
In the Iterative model, iterative process starts with a simple implementation of a small set of the software requirements and iteratively enhances the evolving versions until the complete system is implemented and ready to be deployed.

An iterative life cycle model does not attempt to start with a full specification of requirements. Instead, development begins by specifying and implementing just part of the software, which is then reviewed to identify further requirements. This process is then repeated, producing a new version of the software at the end of each iteration of the model.

Iterative Model - Design

Iterative process starts with a simple implementation of a subset of the software requirements and iteratively enhances the evolving versions until the full system is implemented. At each iteration, design modifications are made and new functional capabilities are added. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental).

The following illustration is a representation of the Iterative and Incremental model −



Iterative and Incremental development is a combination of both iterative design or iterative method and incremental build model for development. "During software development, more than one iteration of the software development cycle may be in progress at the same time." This process may be described as an "evolutionary acquisition" or "incremental build" approach."

In this incremental model, the whole requirement is divided into various builds. During each iteration, the development module goes through the requirements, design, implementation and testing phases. Each subsequent release of the module adds function to the previous release. The process continues till the complete system is ready as per the requirement.

The key to a successful use of an iterative software development lifecycle is rigorous validation of requirements, and verification & testing of each version of the software against those requirements within each cycle of the model. As the software evolves through successive cycles, tests must be repeated and extended to verify each version of the software.

Iterative Model - Application

Like other SDLC models, Iterative and incremental development has some specific applications in the software industry. This model is most often used in the following scenarios −

- Requirements of the complete system are clearly defined and understood.
- Major requirements must be defined; however, some functionalities or requested enhancements may evolve with time.
- There is a time to the market constraint.
- A new technology is being used and is being learnt by the development team while working on the project.

- Resources with needed skill sets are not available and are planned to be used on contract basis for specific iterations.
- There are some high-risk features and goals which may change in the future.

Iterative Model - Pros and Cons

The advantage of this model is that there is a working model of the system at a very early stage of development, which makes it easier to find functional or design flaws. Finding issues at an early stage of development enables to take corrective measures in a limited budget.

The disadvantage with this SDLC model is that it is applicable only to large and bulky software development projects. This is because it is hard to break a small software system into further small serviceable increments/modules.

The advantages of the Iterative and Incremental SDLC Model are as follows −

- Some working functionality can be developed quickly and early in the life cycle.
- Results are obtained early and periodically.
- Parallel development can be planned.
- Progress can be measured.
- Less costly to change the scope/requirements.
- Testing and debugging during smaller iteration is easy.
- Risks are identified and resolved during iteration; and each iteration is an easily managed milestone.
- Easier to manage risk - High risk part is done first.
- With every increment, operational product is delivered.
- Issues, challenges and risks identified from each increment can be utilized/applied to the next increment.
- Risk analysis is better.
- It supports changing requirements.
- Initial Operating time is less.
- Better suited for large and mission-critical projects.
- During the life cycle, software is produced early which facilitates customer evaluation and feedback.

The disadvantages of the Iterative and Incremental SDLC Model are as follows −

- More resources may be required.
- Although cost of change is lesser, but it is not very suitable for changing requirements.
- More management attention is required.
- System architecture or design issues may arise because not all requirements are gathered in the beginning of the entire life cycle.

- Defining increments may require definition of the complete system.
- Not suitable for smaller projects.
- Management complexity is more.
- End of project may not be known which is a risk.
- Highly skilled resources are required for risk analysis.
- Projects progress is highly dependent upon the risk analysis phase.

SDLC - Spiral Model

The spiral model combines the idea of iterative development with the systematic, controlled aspects of the waterfall model. This Spiral model is a combination of iterative development process model and sequential linear development model i.e. the waterfall model with a very high emphasis on risk analysis. It allows incremental releases of the product or incremental refinement through each iteration around the spiral.

Spiral Model - Design

The spiral model has four phases. A software project repeatedly passes through these phases in iterations called Spirals.

Identification

This phase starts with gathering the business requirements in the baseline spiral. In the subsequent spirals as the product matures, identification of system requirements, subsystem requirements and unit requirements are all done in this phase.

This phase also includes understanding the system requirements by continuous communication between the customer and the system analyst. At the end of the spiral, the product is deployed in the identified market.

Design

The Design phase starts with the conceptual design in the baseline spiral and involves architectural design, logical design of modules, physical product design and the final design in the subsequent spirals.

Construct or Build

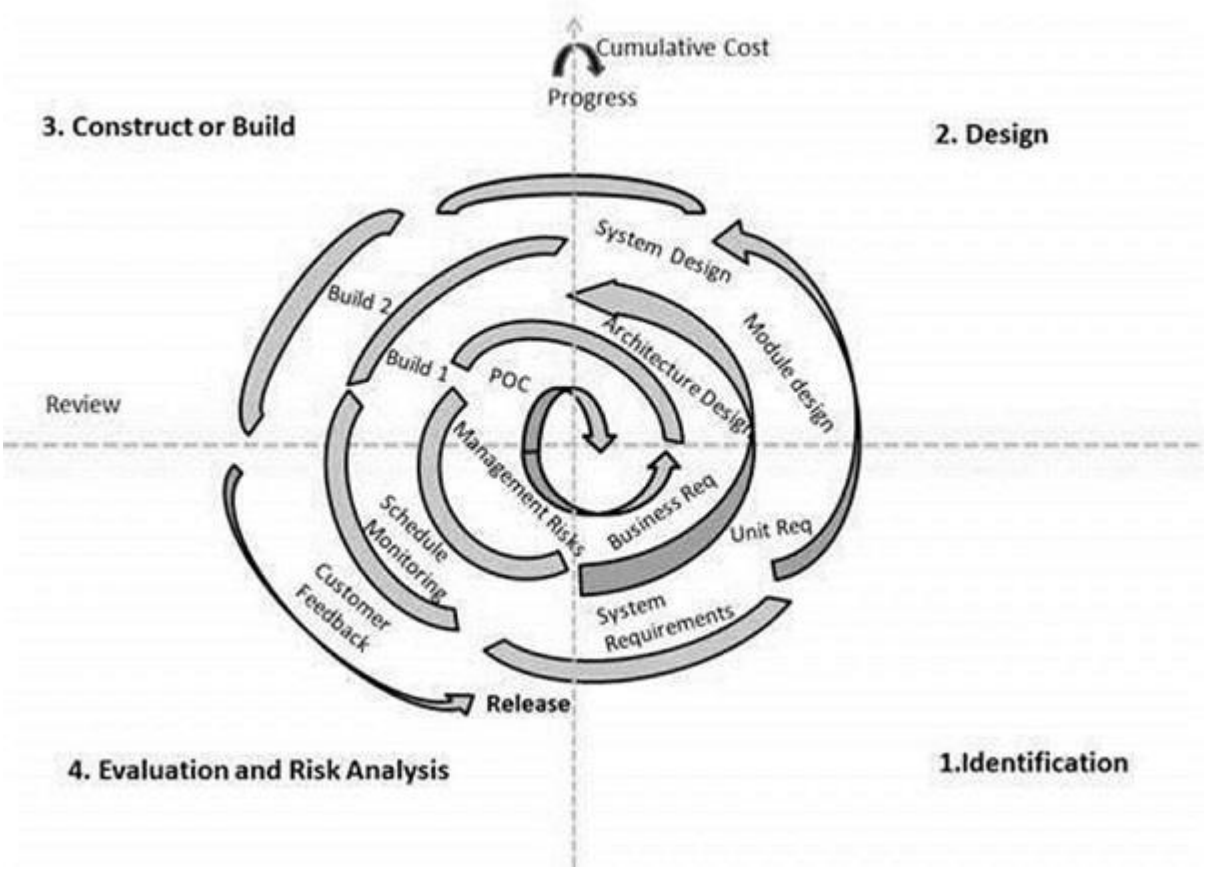
The Construct phase refers to production of the actual software product at every spiral. In the baseline spiral, when the product is just thought of and the design is being developed a POC (Proof of Concept) is developed in this phase to get customer feedback.

Then in the subsequent spirals with higher clarity on requirements and design details a working model of the software called build is produced with a version number. These builds are sent to the customer for feedback.

Evaluation and Risk Analysis

Risk Analysis includes identifying, estimating and monitoring the technical feasibility and management risks, such as schedule slippage and cost overrun. After testing the build, at the end of first iteration, the customer evaluates the software and provides feedback.

The following illustration is a representation of the Spiral Model, listing the activities in each phase.



Based on the customer evaluation, the software development process enters the next iteration and subsequently follows the linear approach to implement the feedback suggested by the customer. The process of iterations along the spiral continues throughout the life of the software.

Spiral Model Application

The Spiral Model is widely used in the software industry as it is in sync with the natural development process of any product, i.e. learning with maturity which involves minimum risk for the customer as well as the development firms.

The following pointers explain the typical uses of a Spiral Model −

- When there is a budget constraint and risk evaluation is important.
- For medium to high-risk projects.
- Long-term project commitment because of potential changes to economic priorities as the requirements change with time.
- Customer is not sure of their requirements which is usually the case.
- Requirements are complex and need evaluation to get clarity.

- New product line which should be released in phases to get enough customer feedback.
- Significant changes are expected in the product during the development cycle.

Spiral Model - Pros and Cons

The advantage of spiral lifecycle model is that it allows elements of the product to be added in, when they become available or known. This assures that there is no conflict with previous requirements and design.

This method is consistent with approaches that have multiple software builds and releases which allows making an orderly transition to a maintenance activity. Another positive aspect of this method is that the spiral model forces an early user involvement in the system development effort.

On the other side, it takes a very strict management to complete such products and there is a risk of running the spiral in an indefinite loop. So, the discipline of change and the extent of taking change requests is very important to develop and deploy the product successfully.

The advantages of the Spiral SDLC Model are as follows −

- Changing requirements can be accommodated.
- Allows extensive use of prototypes.
- Requirements can be captured more accurately.
- Users see the system early.
- Development can be divided into smaller parts and the risky parts can be developed earlier which helps in better risk management.

The disadvantages of the Spiral SDLC Model are as follows −

- Management is more complex.
- End of the project may not be known early.
- Not suitable for small or low risk projects and could be expensive for small projects.
- Process is complex
- Spiral may go on indefinitely.
- Large number of intermediate stages requires excessive documentation.

SDLC - V-Model

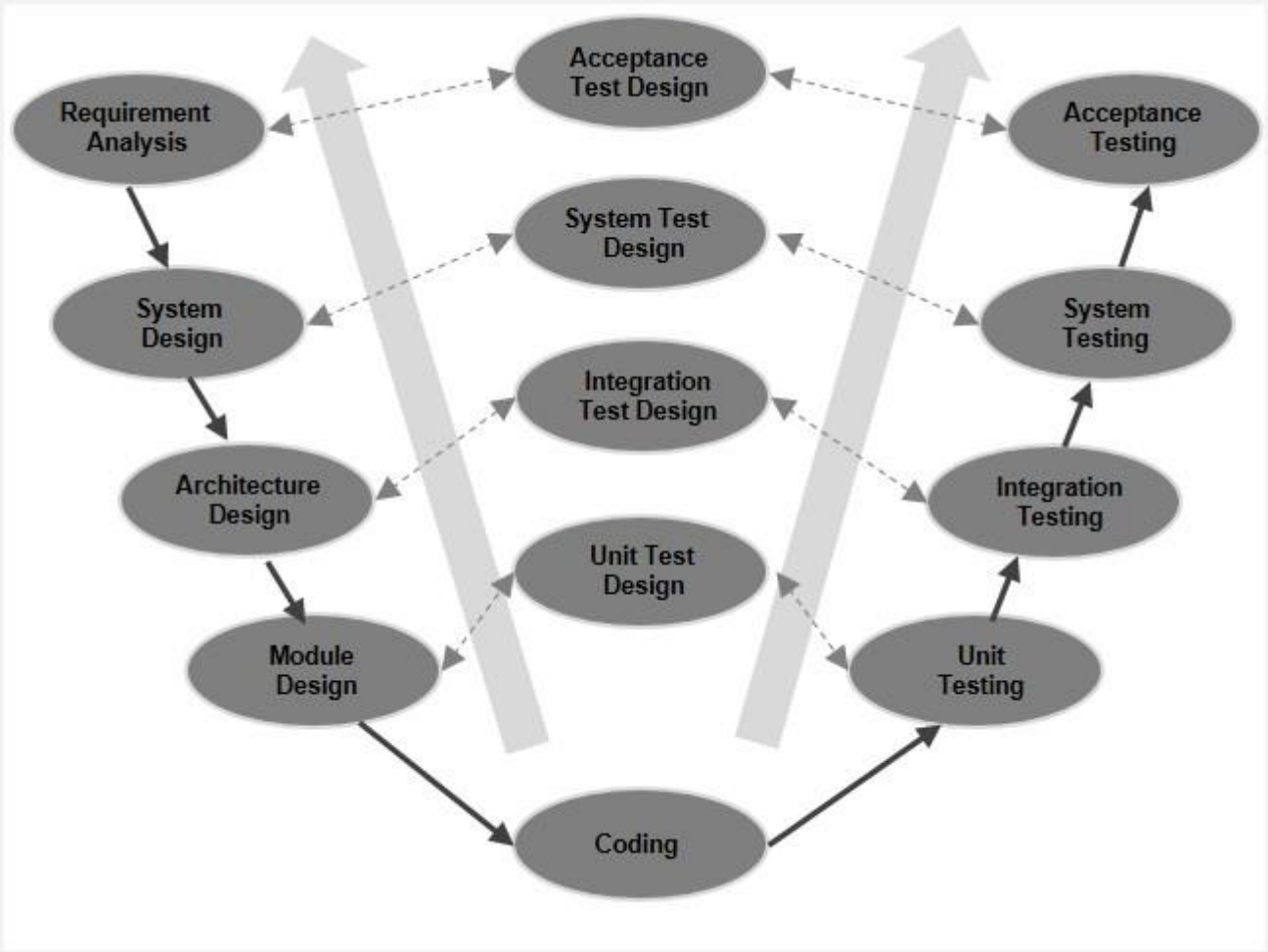
The V-model is an SDLC model where execution of processes happens in a sequential manner in a V-shape. It is also known as **Verification and Validation model**.

The V-Model is an extension of the waterfall model and is based on the association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle, there is a directly associated testing phase. This is a highly-disciplined model and the next phase starts only after completion of the previous phase.

V-Model - Design

Under the V-Model, the corresponding testing phase of the development phase is planned in parallel. So, there are Verification phases on one side of the ‘V’ and Validation phases on the other side. The Coding Phase joins the two sides of the V-Model.

The following illustration depicts the different phases in a V-Model of the SDLC.



V-Model - Verification Phases

There are several Verification phases in the V-Model, each of these are explained in detail below.

Business Requirement Analysis

This is the first phase in the development cycle where the product requirements are understood from the customer’s perspective. This phase involves detailed communication with the customer to understand his expectations and exact requirement. This is a very important activity and needs to be managed well, as most of the customers are not sure about what exactly they need. The **acceptance test design planning** is done at this stage as business requirements can be used as an input for acceptance testing.

System Design

Once you have the clear and detailed product requirements, it is time to design the complete system. The system design will have the understanding and detailing the complete hardware and communication setup for the product under development. The

system test plan is developed based on the system design. Doing this at an earlier stage leaves more time for the actual test execution later.

Architectural Design

Architectural specifications are understood and designed in this phase. Usually more than one technical approach is proposed and based on the technical and financial feasibility the final decision is taken. The system design is broken down further into modules taking up different functionality. This is also referred to as **High Level Design (HLD)**.

The data transfer and communication between the internal modules and with the outside world (other systems) is clearly understood and defined in this stage. With this information, integration tests can be designed and documented during this stage.

Module Design

In this phase, the detailed internal design for all the system modules is specified, referred to as **Low Level Design (LLD)**. It is important that the design is compatible with the other modules in the system architecture and the other external systems. The unit tests are an essential part of any development process and helps eliminate the maximum faults and errors at a very early stage. These unit tests can be designed at this stage based on the internal module designs.

Coding Phase

The actual coding of the system modules designed in the design phase is taken up in the Coding phase. The best suitable programming language is decided based on the system and architectural requirements.

The coding is performed based on the coding guidelines and standards. The code goes through numerous code reviews and is optimized for best performance before the final build is checked into the repository.

Validation Phases

The different Validation Phases in a V-Model are explained in detail below.

Unit Testing

Unit tests designed in the module design phase are executed on the code during this validation phase. Unit testing is the testing at code level and helps eliminate bugs at an early stage, though all defects cannot be uncovered by unit testing.

Integration Testing

Integration testing is associated with the architectural design phase. Integration tests are performed to test the coexistence and communication of the internal modules within the system.

System Testing

System testing is directly associated with the system design phase. System tests check the entire system functionality and the communication of the system under development with external systems. Most of the software and hardware compatibility issues can be uncovered during this system test execution.

Acceptance Testing

Acceptance testing is associated with the business requirement analysis phase and involves testing the product in user environment. Acceptance tests uncover the compatibility issues with the other systems available in the user environment. It also discovers the non-functional issues such as load and performance defects in the actual user environment.

V- Model – Application

V- Model application is almost the same as the waterfall model, as both the models are of sequential type. Requirements have to be very clear before the project starts, because it is usually expensive to go back and make changes. This model is used in the medical development field, as it is strictly a disciplined domain.

The following pointers are some of the most suitable scenarios to use the V-Model application.

- Requirements are well defined, clearly documented and fixed.
- Product definition is stable.
- Technology is not dynamic and is well understood by the project team.
- There are no ambiguous or undefined requirements.
- The project is short.

V-Model - Pros and Cons

The advantage of the V-Model method is that it is very easy to understand and apply. The simplicity of this model also makes it easier to manage. The disadvantage is that the model is not flexible to changes and just in case there is a requirement change, which is very common in today's dynamic world, it becomes very expensive to make the change.

The advantages of the V-Model method are as follows −

- This is a highly-disciplined model and Phases are completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Simple and easy to understand and use.
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.

The disadvantages of the V-Model method are as follows −

- High risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.

- Not suitable for the projects where requirements are at a moderate to high risk of changing.
- Once an application is in the testing stage, it is difficult to go back and change a functionality.
- No working software is produced until late during the life cycle.

SDLC - Big Bang Model

The Big Bang model is an SDLC model where we do not follow any specific process. The development just starts with the required money and efforts as the input, and the output is the software developed which may or may not be as per customer requirement. This Big Bang Model does not follow a process/procedure and there is a very little planning required. Even the customer is not sure about what exactly he wants and the requirements are implemented on the fly without much analysis.

Usually this model is followed for small projects where the development teams are very small.

Big Bang Model – Design and Application

The Big Bang Model comprises of focusing all the possible resources in the software development and coding, with very little or no planning. The requirements are understood and implemented as they come. Any changes required may or may not need to revamp the complete software.

This model is ideal for small projects with one or two developers working together and is also useful for academic or practice projects. It is an ideal model for the product where requirements are not well understood and the final release date is not given.

Big Bang Model - Pros and Cons

The advantage of this Big Bang Model is that it is very simple and requires very little or no planning. Easy to manage and no formal procedure are required.

However, the Big Bang Model is a very high risk model and changes in the requirements or misunderstood requirements may even lead to complete reversal or scraping of the project. It is ideal for repetitive or small projects with minimum risks.

The advantages of the Big Bang Model are as follows −

- This is a very simple model
- Little or no planning required
- Easy to manage
- Very few resources required
- Gives flexibility to developers
- It is a good learning aid for new comers or students.

The disadvantages of the Big Bang Model are as follows −

- Very High risk and uncertainty.
- Not a good model for complex and object-oriented projects.

- Poor model for long and ongoing projects.
- Can turn out to be very expensive if requirements are misunderstood.

SDLC - Agile Model

Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product. Agile Methods break the product into small incremental builds. These builds are provided in iterations. Each iteration typically lasts from about one to three weeks. Every iteration involves cross functional teams working simultaneously on various areas like −

- Planning
- Requirements Analysis
- Design
- Coding
- Unit Testing and
- Acceptance Testing.

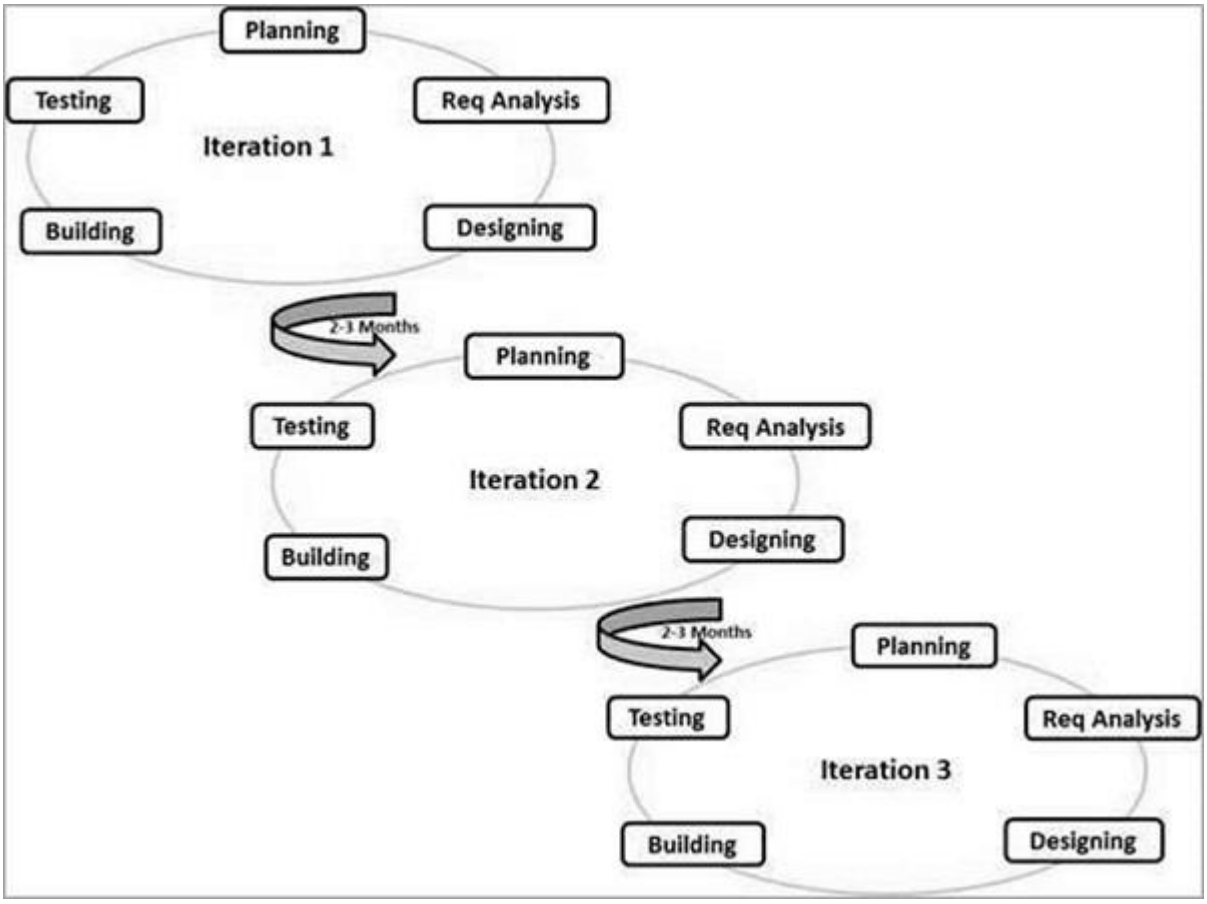
At the end of the iteration, a working product is displayed to the customer and important stakeholders.

What is Agile?

Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements. In Agile, the tasks are divided to time boxes (small time frames) to deliver specific features for a release.

Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer.

Here is a graphical illustration of the Agile Model −



The Agile thought process had started early in the software development and started becoming popular with time due to its flexibility and adaptability.

The most popular Agile methods include Rational Unified Process (1994), Scrum (1995), Crystal Clear, Extreme Programming (1996), Adaptive Software Development, Feature Driven Development, and Dynamic Systems Development Method (DSDM) (1995). These are now collectively referred to as **Agile Methodologies**, after the Agile Manifesto was published in 2001.

Following are the Agile Manifesto principles −

- **Individuals and interactions** − In Agile development, self-organization and motivation are important, as are interactions like co-location and pair programming.
- **Working software** − Demo working software is considered the best means of communication with the customers to understand their requirements, instead of just depending on documentation.
- **Customer collaboration** − As the requirements cannot be gathered completely in the beginning of the project due to various factors, continuous customer interaction is very important to get proper product requirements.
- **Responding to change** − Agile Development is focused on quick responses to change and continuous development.

Agile Vs Traditional SDLC Models

Agile is based on the **adaptive software development methods**, whereas the traditional SDLC models like the waterfall model is based on a predictive approach. Predictive teams in the traditional SDLC models usually work with detailed planning and have a

complete forecast of the exact tasks and features to be delivered in the next few months or during the product life cycle.

Predictive methods entirely depend on the **requirement analysis and planning** done in the beginning of cycle. Any changes to be incorporated go through a strict change control management and prioritization.

Agile uses an **adaptive approach** where there is no detailed planning and there is clarity on future tasks only in respect of what features need to be developed. There is feature driven development and the team adapts to the changing product requirements dynamically. The product is tested very frequently, through the release iterations, minimizing the risk of any major failures in future.

Customer Interaction is the backbone of this Agile methodology, and open communication with minimum documentation are the typical features of Agile development environment. The agile teams work in close collaboration with each other and are most often located in the same geographical location.

Agile Model - Pros and Cons

Agile methods are being widely accepted in the software world recently. However, this method may not always be suitable for all products. Here are some pros and cons of the Agile model.

The advantages of the Agile Model are as follows −

- Is a very realistic approach to software development.
- Promotes teamwork and cross training.
- Functionality can be developed rapidly and demonstrated.
- Resource requirements are minimum.
- Suitable for fixed or changing requirements
- Delivers early partial working solutions.
- Good model for environments that change steadily.
- Minimal rules, documentation easily employed.
- Enables concurrent development and delivery within an overall planned context.
- Little or no planning required.
- Easy to manage.
- Gives flexibility to developers.

The disadvantages of the Agile Model are as follows −

- Not suitable for handling complex dependencies.
- More risk of sustainability, maintainability and extensibility.
- An overall plan, an agile leader and agile PM practice is a must without which it will not work.
- Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.

- Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.
- There is a very high individual dependency, since there is minimum documentation generated.
- Transfer of technology to new team members may be quite challenging due to lack of documentation.

SDLC - RAD Model

The **RAD (Rapid Application Development)** model is based on prototyping and iterative development with no specific planning involved. The process of writing the software itself involves the planning required for developing the product.

Rapid Application Development focuses on gathering customer requirements through workshops or focus groups, early testing of the prototypes by the customer using iterative concept, reuse of the existing prototypes (components), continuous integration and rapid delivery.

What is RAD?

Rapid application development is a software development methodology that uses minimal planning in favor of rapid prototyping. A prototype is a working model that is functionally equivalent to a component of the product.

In the RAD model, the functional modules are developed in parallel as prototypes and are integrated to make the complete product for faster product delivery. Since there is no detailed preplanning, it makes it easier to incorporate the changes within the development process.

RAD projects follow iterative and incremental model and have small teams comprising of developers, domain experts, customer representatives and other IT resources working progressively on their component or prototype.

The most important aspect for this model to be successful is to make sure that the prototypes developed are reusable.

RAD Model Design

RAD model distributes the analysis, design, build and test phases into a series of short, iterative development cycles.

Following are the various phases of the RAD Model −

Business Modelling

The business model for the product under development is designed in terms of flow of information and the distribution of information between various business channels. A complete business analysis is performed to find the vital information for business, how it can be obtained, how and when is the information processed and what are the factors driving successful flow of information.

Data Modelling

The information gathered in the Business Modelling phase is reviewed and analyzed to form sets of data objects vital for the business. The attributes of all data sets is identified and defined. The relation between these data objects are established and defined in detail in relevance to the business model.

Process Modelling

The data object sets defined in the Data Modelling phase are converted to establish the business information flow needed to achieve specific business objectives as per the business model. The process model for any changes or enhancements to the data object sets is defined in this phase. Process descriptions for adding, deleting, retrieving or modifying a data object are given.

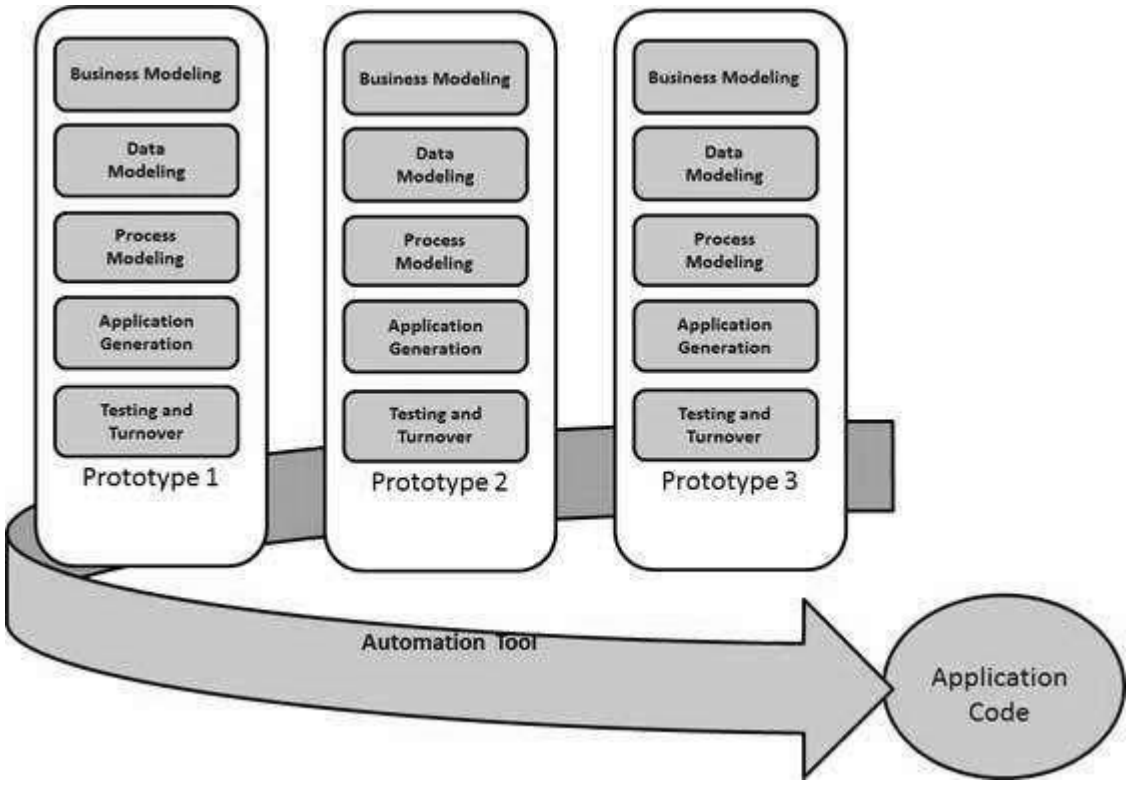
Application Generation

The actual system is built and coding is done by using automation tools to convert process and data models into actual prototypes.

Testing and Turnover

The overall testing time is reduced in the RAD model as the prototypes are independently tested during every iteration. However, the data flow and the interfaces between all the components need to be thoroughly tested with complete test coverage. Since most of the programming components have already been tested, it reduces the risk of any major issues.

The following illustration describes the RAD Model in detail.



RAD Model Vs Traditional SDLC

The traditional SDLC follows a rigid process models with high emphasis on requirement analysis and gathering before the coding starts. It puts pressure on the customer to sign off the requirements before the project starts and the customer doesn't get the feel of the product as there is no working build available for a long time.

The customer may need some changes after he gets to see the software. However, the change process is quite rigid and it may not be feasible to incorporate major changes in the product in the traditional SDLC.

The RAD model focuses on iterative and incremental delivery of working models to the customer. This results in rapid delivery to the customer and customer involvement during the complete development cycle of product reducing the risk of non-conformance with the actual user requirements.

RAD Model - Application

RAD model can be applied successfully to the projects in which clear modularization is possible. If the project cannot be broken into modules, RAD may fail.

The following pointers describe the typical scenarios where RAD can be used −

- RAD should be used only when a system can be modularized to be delivered in an incremental manner.
- It should be used if there is a high availability of designers for Modelling.
- It should be used only if the budget permits use of automated code generating tools.
- RAD SDLC model should be chosen only if domain experts are available with relevant business knowledge.
- Should be used where the requirements change during the project and working prototypes are to be presented to customer in small iterations of 2-3 months.

RAD Model - Pros and Cons

RAD model enables rapid delivery as it reduces the overall development time due to the reusability of the components and parallel development. RAD works well only if high skilled engineers are available and the customer is also committed to achieve the targeted prototype in the given time frame. If there is commitment lacking on either side the model may fail.

The advantages of the RAD Model are as follows −

- Changing requirements can be accommodated.
- Progress can be measured.
- Iteration time can be short with use of powerful RAD tools.
- Productivity with fewer people in a short time.
- Reduced development time.
- Increases reusability of components.
- Quick initial reviews occur.

- Encourages customer feedback.
- Integration from very beginning solves a lot of integration issues.

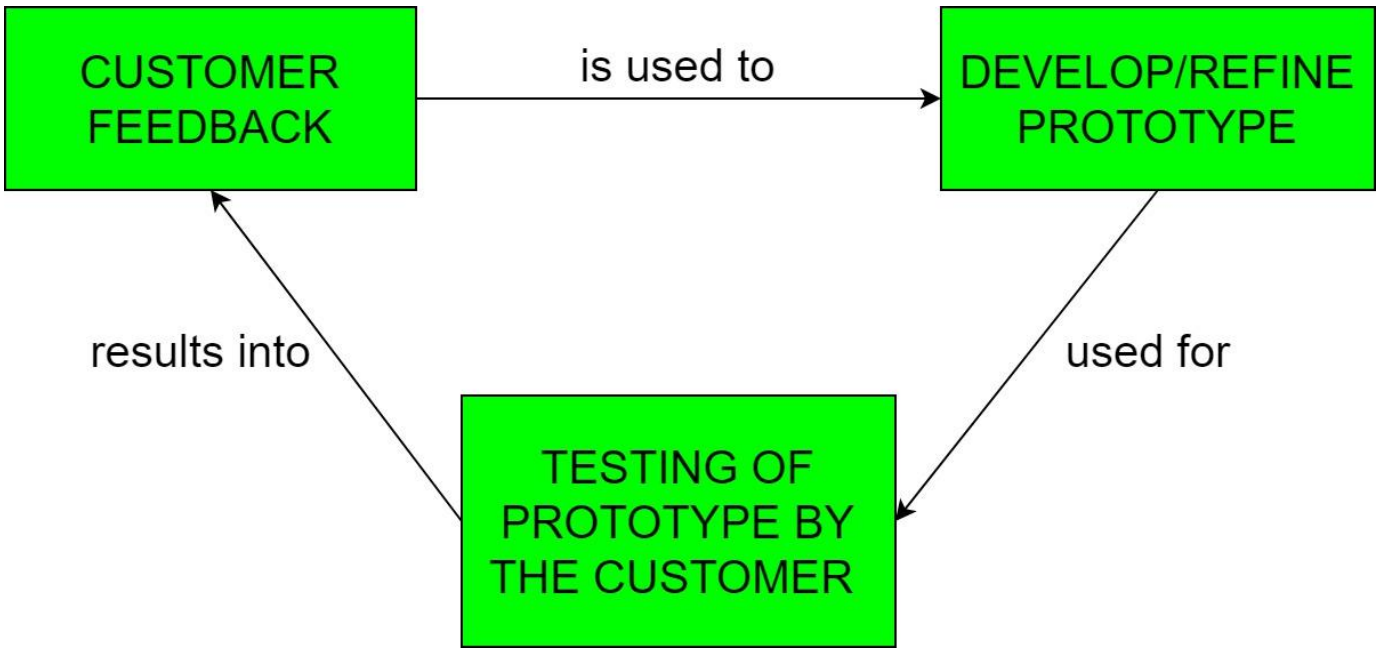
The disadvantages of the RAD Model are as follows −

- Dependency on technically strong team members for identifying business requirements.
- Only system that can be modularized can be built using RAD.
- Requires highly skilled developers/designers.
- High dependency on Modelling skills.
- Inapplicable to cheaper projects as cost of Modelling and automated code generation is very high.
- Management complexity is more.
- Suitable for systems that are component based and scalable.
- Requires user involvement throughout the life cycle.
- Suitable for project requiring shorter development times.

SDLC - Software Prototype Model

The Software Prototyping refers to building software application prototypes which displays the functionality of the product under development, but may not actually hold the exact logic of the original software.

Software prototyping is becoming very popular as a software development model, as it enables to understand customer requirements at an early stage of development. It helps get valuable feedback from the customer and helps software designers and developers understand about what exactly is expected from the product under development.



What is Software Prototyping?

Prototype is a working model of software with some limited functionality. The prototype does not always hold the exact logic used in the actual software application and is an extra effort to be considered under effort estimation.

Prototyping is used to allow the users evaluate developer proposals and try them out before implementation. It also helps understand the requirements which are user specific and may not have been considered by the developer during product design.

Following is a stepwise approach explained to design a software prototype.

Basic Requirement Identification

This step involves understanding the very basics product requirements especially in terms of user interface. The more intricate details of the internal design and external aspects like performance and security can be ignored at this stage.

Developing the initial Prototype

The initial Prototype is developed in this stage, where the very basic requirements are showcased and user interfaces are provided. These features may not exactly work in the same manner internally in the actual software developed. While, the workarounds are used to give the same look and feel to the customer in the prototype developed.

Review of the Prototype

The prototype developed is then presented to the customer and the other important stakeholders in the project. The feedback is collected in an organized manner and used for further enhancements in the product under development.

Revise and Enhance the Prototype

The feedback and the review comments are discussed during this stage and some negotiations happen with the customer based on factors like – time and budget constraints and technical feasibility of the actual implementation. The changes accepted are again incorporated in the new Prototype developed and the cycle repeats until the customer expectations are met.

Prototypes can have horizontal or vertical dimensions. A Horizontal prototype displays the user interface for the product and gives a broader view of the entire system, without concentrating on internal functions. A Vertical prototype on the other side is a detailed elaboration of a specific function or a sub system in the product.

The purpose of both horizontal and vertical prototype is different. Horizontal prototypes are used to get more information on the user interface level and the business requirements. It can even be presented in the sales demos to get business in the market. Vertical prototypes are technical in nature and are used to get details of the exact functioning of the sub systems. For example, database requirements, interaction and data processing loads in a given sub system.

Software Prototyping - Types

There are different types of software prototypes used in the industry. Following are the major software prototyping types used widely −

Throwaway/Rapid Prototyping

Throwaway prototyping is also called as rapid or close ended prototyping. This type of prototyping uses very little efforts with minimum requirement analysis to build a prototype. Once the actual requirements are understood, the prototype is discarded and the actual system is developed with a much clear understanding of user requirements.

Evolutionary Prototyping

Evolutionary prototyping also called as breadboard prototyping is based on building actual functional prototypes with minimal functionality in the beginning. The prototype developed forms the heart of the future prototypes on top of which the entire system is built. By using evolutionary prototyping, the well-understood requirements are included in the prototype and the requirements are added as and when they are understood.

Incremental Prototyping

Incremental prototyping refers to building multiple functional prototypes of the various sub-systems and then integrating all the available prototypes to form a complete system.

Extreme Prototyping

Extreme prototyping is used in the web development domain. It consists of three sequential phases. First, a basic prototype with all the existing pages is presented in the HTML format. Then the data processing is simulated using a prototype services layer. Finally, the services are implemented and integrated to the final prototype. This process is called Extreme Prototyping used to draw attention to the second phase of the process, where a fully functional UI is developed with very little regard to the actual services.

Software Prototyping - Application

Software Prototyping is most useful in development of systems having high level of user interactions such as online systems. Systems which need users to fill out forms or go through various screens before data is processed can use prototyping very effectively to give the exact look and feel even before the actual software is developed.

Software that involves too much of data processing and most of the functionality is internal with very little user interface does not usually benefit from prototyping. Prototype development could be an extra overhead in such projects and may need lot of extra efforts.

Software Prototyping - Pros and Cons

Software prototyping is used in typical cases and the decision should be taken very carefully so that the efforts spent in building the prototype add considerable value to the final software developed. The model has its own pros and cons discussed as follows.

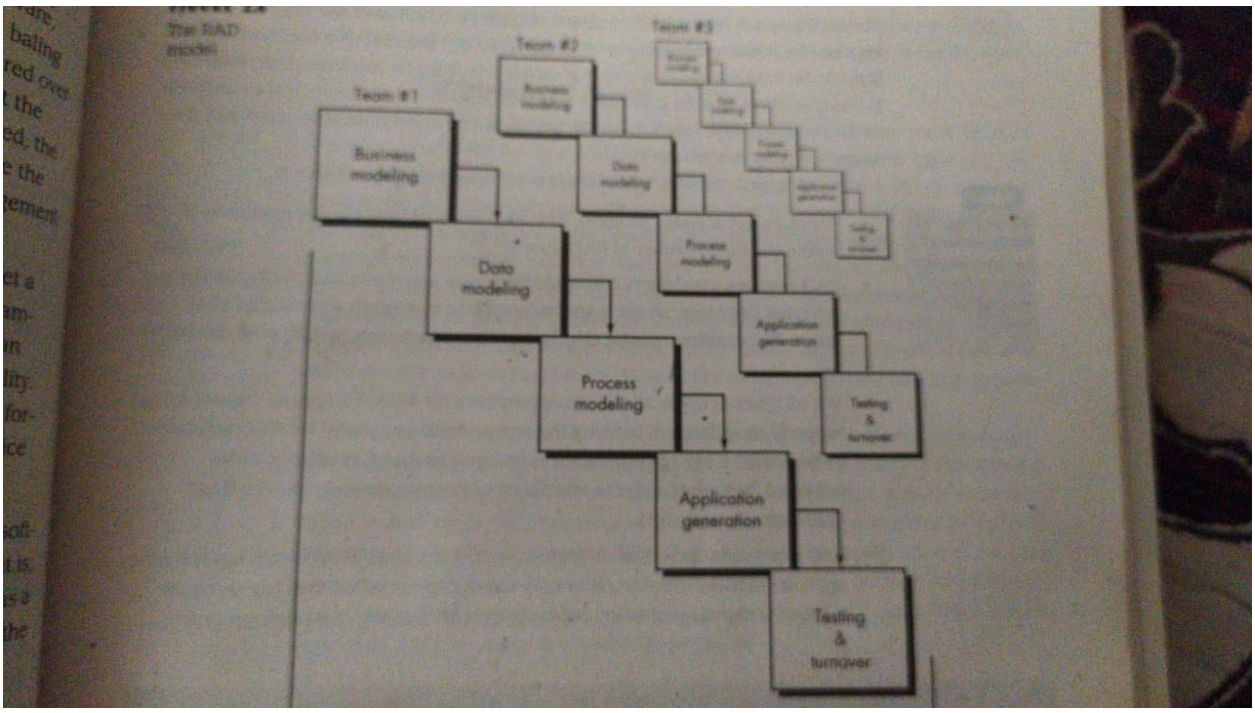
The advantages of the Prototyping Model are as follows −

- Increased user involvement in the product even before its implementation.
- Since a working model of the system is displayed, the users get a better understanding of the system being developed.
- Reduces time and cost as the defects can be detected much earlier.
- Quicker user feedback is available leading to better solutions.
- Missing functionality can be identified easily.
- Confusing or difficult functions can be identified.

The Disadvantages of the Prototyping Model are as follows −

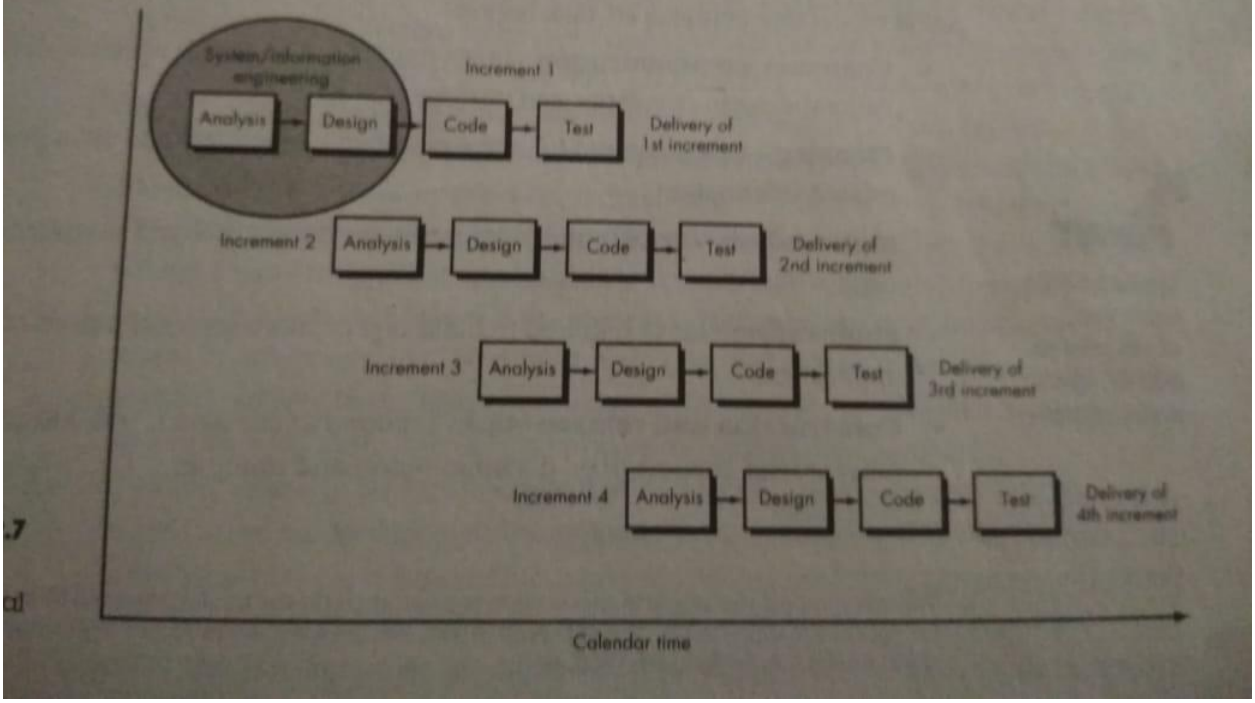
- Risk of insufficient requirement analysis owing to too much dependency on the prototype.
- Users may get confused in the prototypes and actual systems.
- Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.
- Developers may try to reuse the existing prototypes to build the actual system, even when it is not technically feasible.
- The effort invested in building prototypes may be too much if it is not monitored properly.

RAD Model Diagram



Incremental Model Diagram

developed for the next increment. The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality. This process is repeated following the delivery of each increment, until the complete product is produced.



Component Based Development

development model composes applications from prepackaged software components (called *classes*).

The engineering activity begins with the identification of candidate classes. This is accomplished by examining the data to be manipulated by the application and the algorithms that will be applied to accomplish the manipulation.[12] Corresponding data and algorithms are packaged into a class.

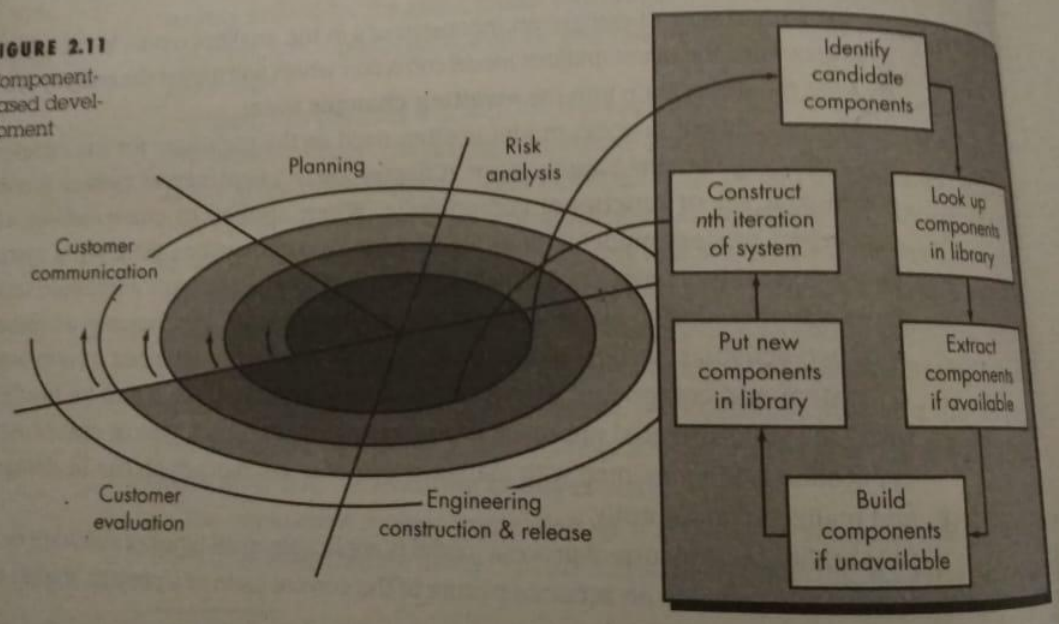
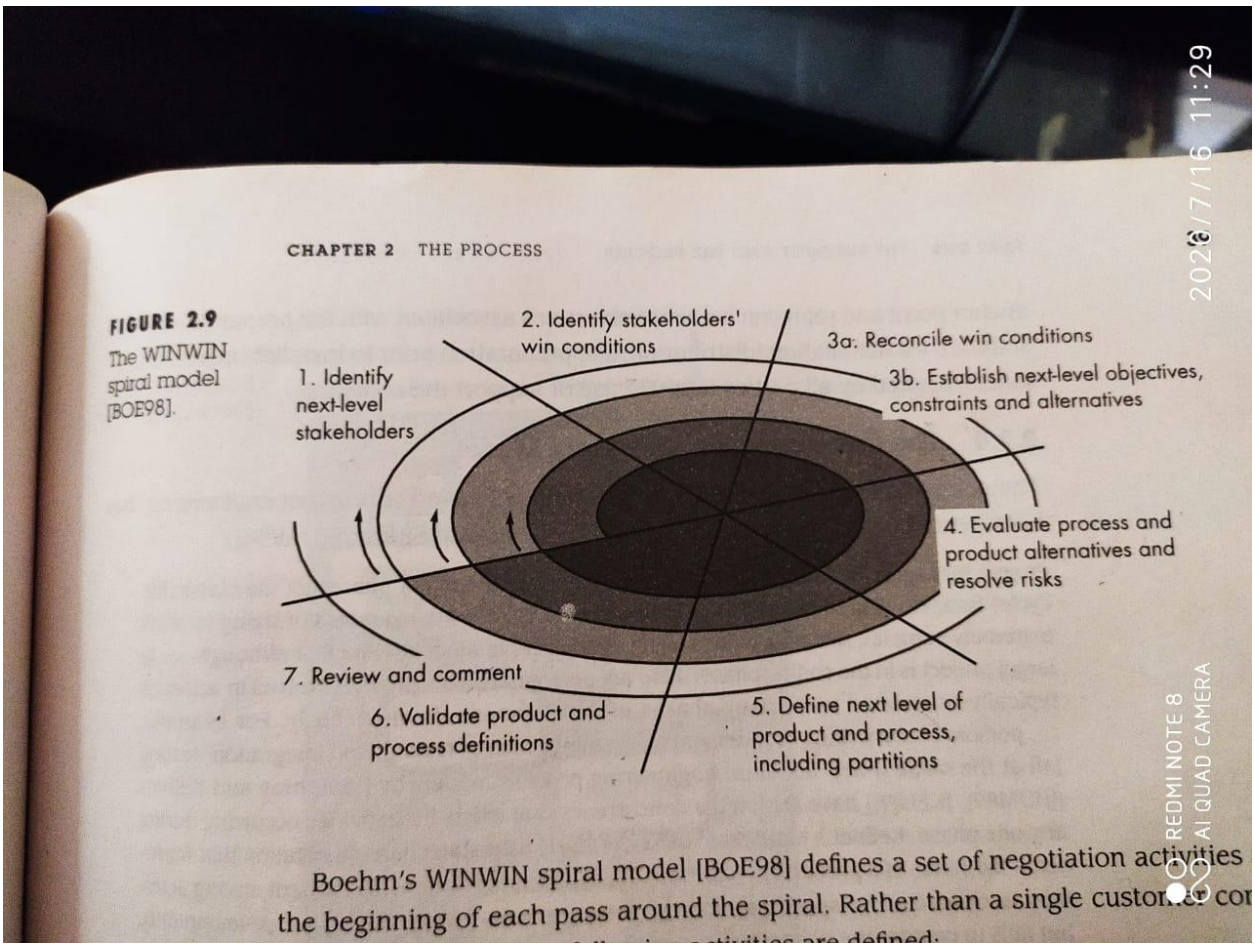
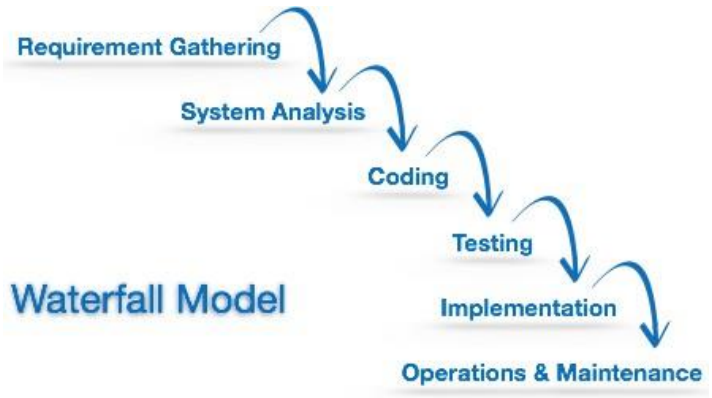


FIGURE 2.11 Component-based development

12 This is a simplified description of class definition. For a more detailed discussion, see Chapter 2

Win-Win Spiral Model



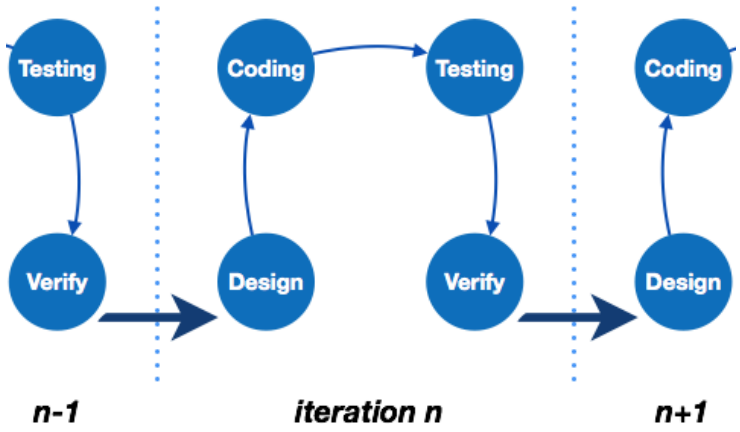


This model assumes that everything is carried out and taken place perfectly as planned in the previous stage and there is no need to think about the past issues that may arise in the next phase. This model does not work smoothly if there are some issues left at the previous step. The sequential nature of model does not allow us to go back and undo or redo our actions.

This model is best suited when developers already have designed and developed similar software in the past and are aware of all its domains.

Iterative Model

This model leads the software development process in iterations. It projects the processofdevelopmentincyclicmannerrepeatingeverystepaftereverycycleof SDLCprocess.

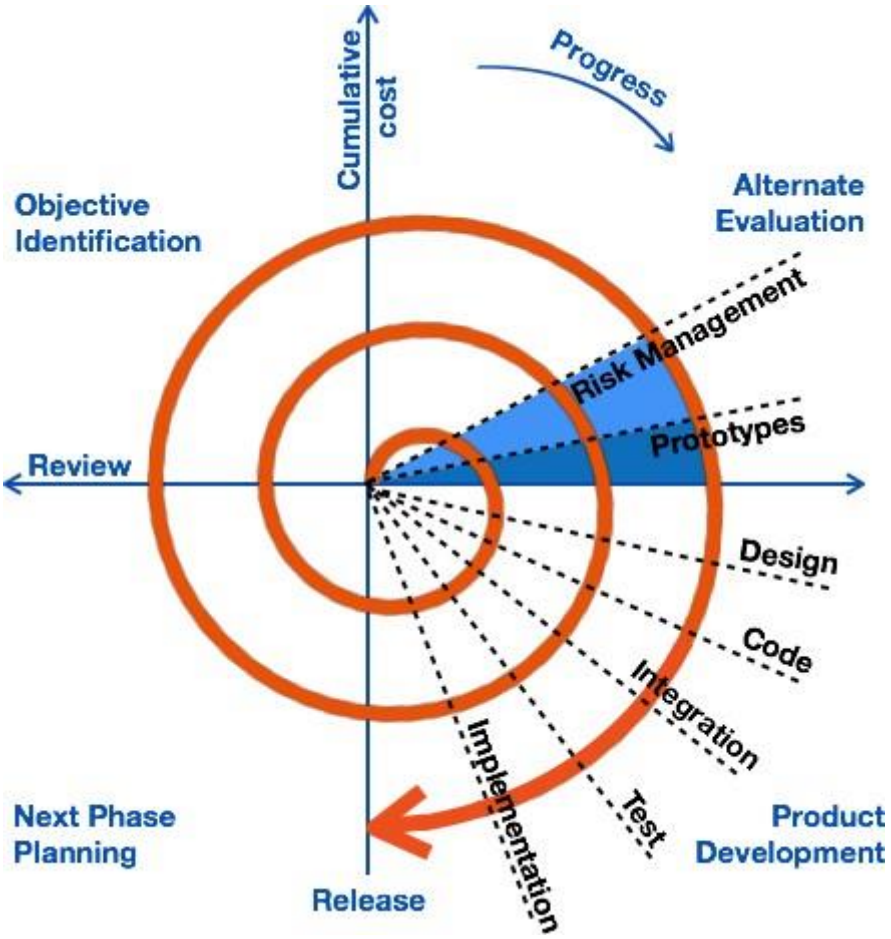


The software is first developed on very small scale and all the steps are followed which are taken into consideration. Then, on every next iteration, more features andmodulesaredesigned,coded,tested,andaddedtothesoftware.Everycycle produces a software, which is complete in itself and has more features and capabilities than that of the previousone.

Aftereachiteration,themanagementteamcandoworkonriskmanagementand prepareforthenextiteration.Becauseacycleincludessmallportionofwhole softwareprocess,itiseasiertomanagethedevelopmentprocessbutitconsumes moreresources.

Spiral Model

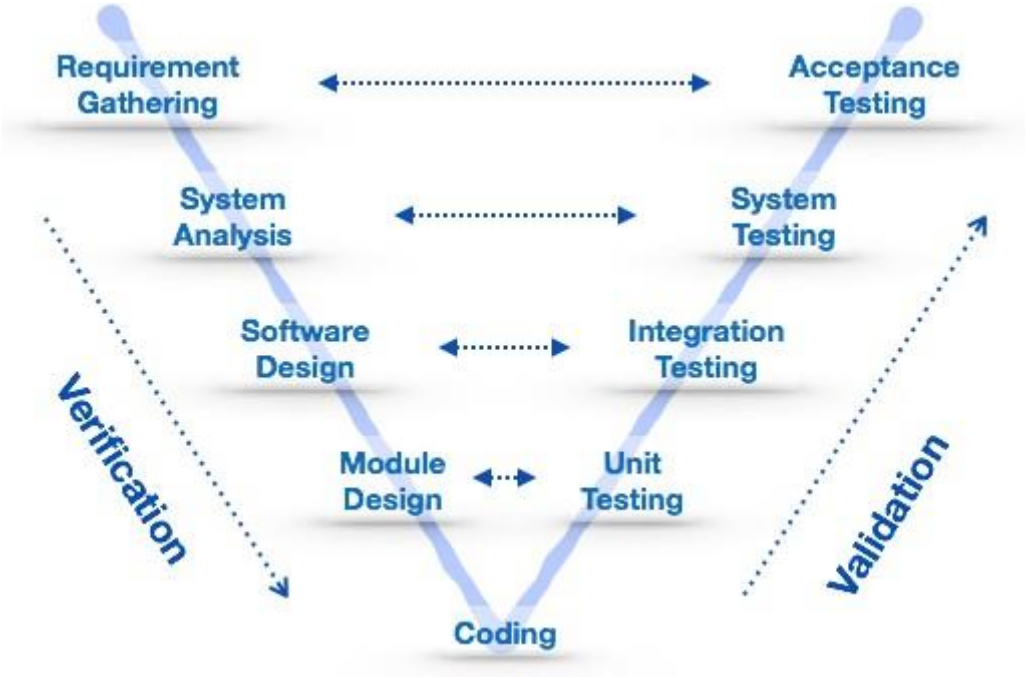
Spiralmodelisacombinationofboth,iterativemodelandoneoftheSDLCmodel. It can be seen as if you choose one SDLC model and combined it with cyclic process (iterativemodel).



Thismodelconsidersrisk,whichoftengoesun-noticedbymostothermodels.The model starts with determining objectives and constraints of the software at the startofoneiteration.Nextphaseisofprototypingthesoftware.Thisincludesrisk analysis. Then one standard SDLC model is used to build the software. In the fourth phase of the plan of next iteration isprepared.

V – model

The major drawback of waterfall model is we move to the next stage only when the previous one is finished and there was no chance to go back if something is
found wrong in later stages. V-Model provides means of testing of software at each stage in reverse manner.



At every stage, test plans and test cases are created to verify and validate the product according to the requirement of that stage. For example, in requirement gatheringstagethetestteampreparesallthetestcasesincorrespondencetothe requirements. Later, when the product is developed and is ready for testing, test casesofthisstageverifythesoftwareagainstitsvaliditytowardsrequirementsat thisstage.

Thismakesbothverificationandvalidationgoinparallel.Thismodelisalsoknown as verification and validationmodel.

Big Bang Model

This model is the simplest model in its form. It requires little planning, lots of programmingandlotsoffunds.Thismodelisconceptualizedaroundthebigbang of universe. As scientists say that after big bang lots of galaxies, planets, and stars evolved just as an event. Likewise, if we put together lots of programming and funds, you may achieve the best softwareproduct.

For this model, very small amount of planning is required. It does not follow any process, or at times the customer is not sure about the requirements and future needs. So the input requirements are arbitrary.

This model is not suitable for large software projects but good one for learning and experimenting.

The Linear SequentialModel:

The **waterfall model** is also called the classic life cycle, suggests a systematic sequential approach to software development that begins with customer specification of requirements and progress through planning, modeling, construction and deployment, culminating in on-going support of the completedsoftware.

It can serve as a useful Process model in situations where requirements are fixed and work is to proceed to completion in a linear manner.

The waterfall model is applied in:

- Real projects rarely follow the sequential flow that the modelproposes.
- It is often difficult for the customer to state all requirementsexplicitly.
- The customer must havepatience.
- A working version of the program will not be available until late in the projecttime-span.

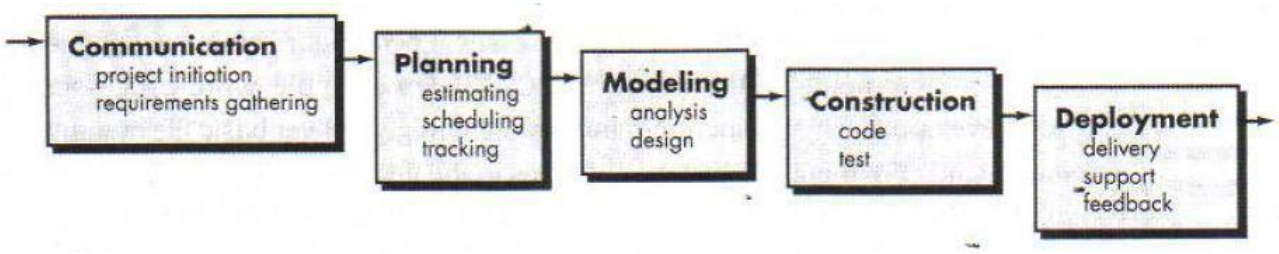


Fig: Waterfall Model

The PrototypingModel:

- Prototype model focuses on producing software productquickly.
- A prototype paradigm may be the best approach in manysituations.
- The developer may be ensured with the efficiency of thealgorithm.
- The prototyping model begins withcommunication.
- In this phase the software engineers and customer must define the overall objective for

the software and identify whatever requirements are known, outline the areas where further definition ismandatory.

- In quick plan phase, prototyping iteration is planned quickly & modelingoccurs.
- The quick design process on a representation and the prototype isconstructed.
- In final phase the prototype is developed and then evaluated by the customerloses.

Problem with the prototyping model:

- The customer sees what appears to be a working version of the software product. Hence software development management isnecessary.
- The developer often makes the implementation quickly which results in the usage of inefficientalgorithm.

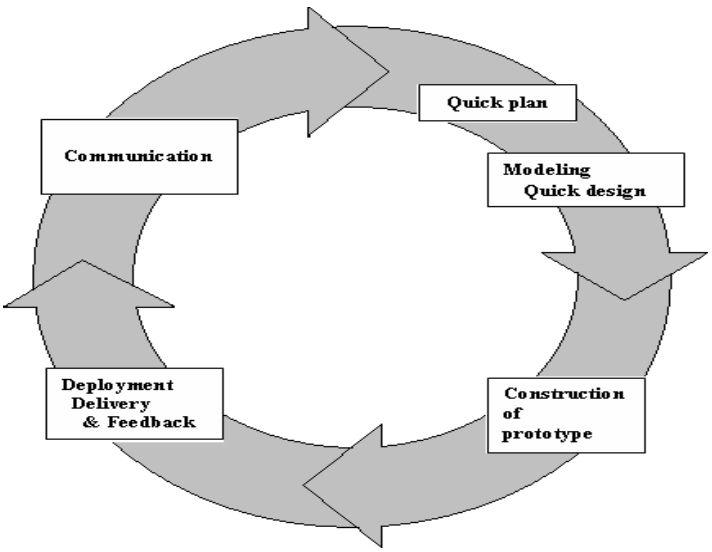


Fig. Prototype Model

Advantage:

- Iteration process facilitatingenhancements.
- Partial product can be viewed by thecustomer.
- Flexibility of theproduct.
- Customer satisfaction of theproduct.

Disadvantage:

- No optimalsolution.
- Time consuming if algorithm used isinefficient.

- Poor documentation resulting withdifficulty.

RAD Model :(Rapid ApplicationDevelopment)

- RAD is an incremental software process models that emphasis a short development cycle.
- In RAD model the rapid development is achieved by using a component based constructionapproach.
- The first phase is communication phase it works to understand the business problem & informationcharacteristics.
- Planning phase is essential because multiple software teams work in parallel on different softwarefunction.

- Modeling encompasses three majorphases.
    1. Componentreuse.
    2. Automatic codegeneration.
    3. Testing.

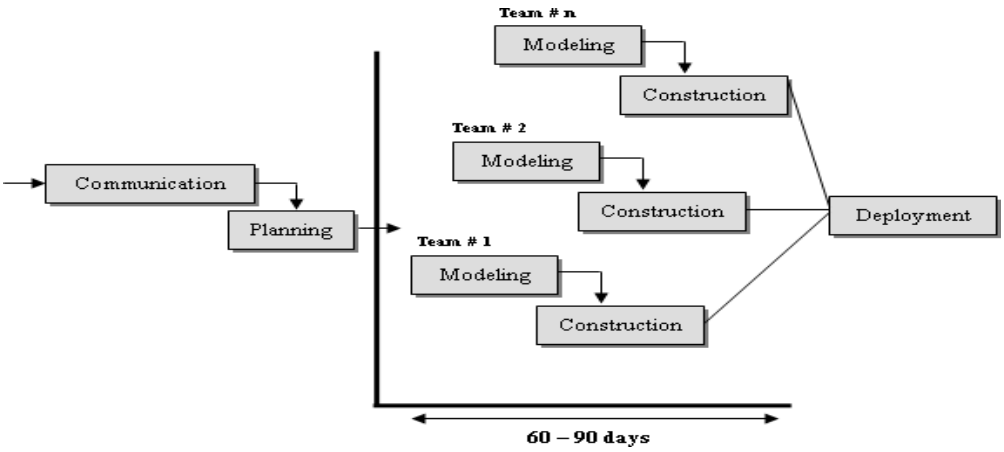


Fig: RAD model

Drawbacks of RAD:

- For large but scalable projects RAD requires sufficient human resources to create the right number of RADteam.
- If developer and customer are not committed to the rapid-fire activities necessary to complete the software in a much abbreviated time frame, RAD project willfail.
- If a system cannot be properly modularized, building the component necessary for RAD will beproblematic.
- If high performance is an issue & performance is to be achieved through tuning the

interface to system component, the RAD approach may notwork.

- RAD may not be appropriate when technical risks are high. Eg: when a new application makes necessary use of newtechnology.

EvolutionaryModel:

- Referred as the successive version model. In evolutionary model the software is firstbroken into a several model.
- The development team first develops the core module of thesoftware.
- The initial product skeleton is refined into increasing levels of capability by adding new functionality in the successiveversion.
- This modeling is very popular for projects because the system can easily be partitioned into standalone units in terms of the object.

Properties of evolutionary model:

- Continuing change indegradation.
- Increase incomplexity.
- Programevaluation.
- Invariantevaluation.
- Incremental growthlimit.

Continuing change in degradation:

Software system is continually changing which makes software become very less useful.

Increase in complexity:

Due to continual changes the software complexity increase, integrating various modules will bedifficult.

Program evaluation:

Program, process and measure of project and system attribute are statistically self- regulatory with determinable tends.

Invariant work rate:

Rate of activity in large software project is statistically invariant.

During the life cycle of large software system volume of modification in successive release is statistically invariant.

Disadvantage:

- Difficult to divide problem into severalunit.
- Used only for very largeprojects.

Advantages:

- Reduce errors
- User get chance to experiment with partially developed software much before the computer version of software, so the changes requested after thedelivery are minimized.

The Incremental Model

- There are many situations in which initial software requirements are reasonably well-defined but the overall scope of the development effort precludes a purely linearprocess.
- The **incremental model** combines elements of the waterfall model applied in an iterative fashion. The incremental model applies linear sequences in a staggered fashion ascalendar time progresses.Eachlinearsequenceproducesdeliverable—incrementsIofthesoftware.
- When an incremental model is used, the first incrementis often a **core product**. The incremental process model, like prototyping and other evolutionary approaches, isiterative in nature.
- Unlike prototyping, the incremental model focuses on the delivery of an operational product with eachincrement.

The figure 1.9 shows the linear sequences in a staged fashion as calendar time progresses of the incremental model.

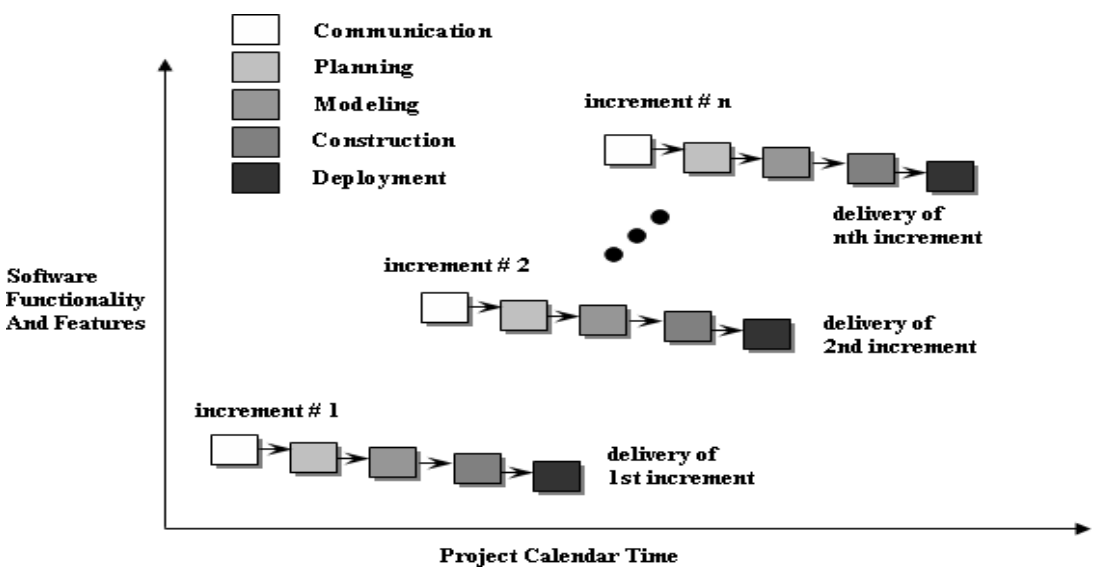


Figure: Incremental Models

Advantages:

- Requirements are identified in every phase.
- Cost is distributed with less human power & staffing.
- Errors are simultaneouslycorrected.
- Feedback requirement isclear.
- Testing iseasy.

Disadvantages:

- Requires proper planning to distribute thework.
- Total cost required for the development of the product ishigh.
- Interface model should bewell-defined.

The SpiralModel

The **spiral model** is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfallmodel. It provides the potential for rapid development of increasingly more complete versions of thesoftware.

The spiral model is a realistic approach to the development of large scale systems and software. The spiral development model is a risk-driven process model generator that is used to guide multi- stakeholder concurrent engineering of software intensive systems.

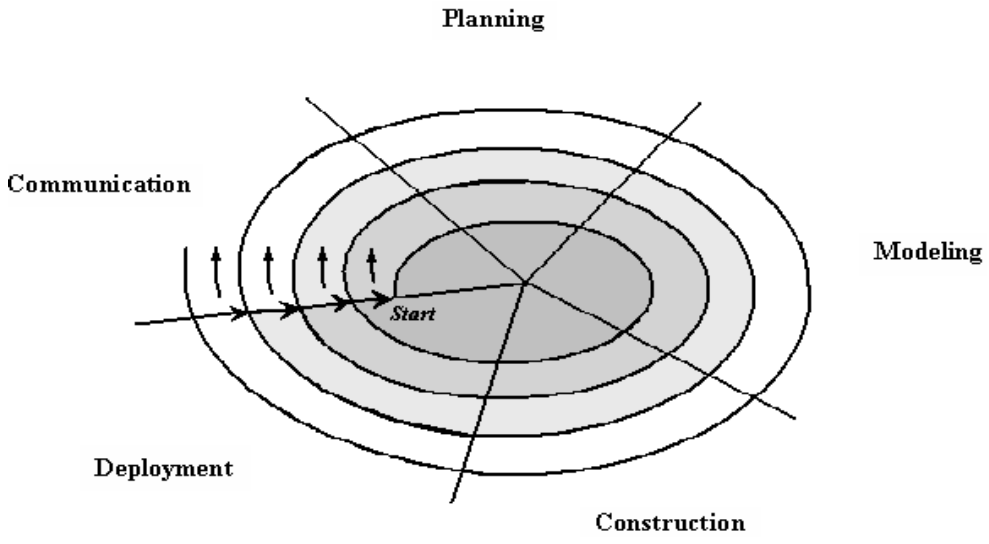


Figure: Spiral model

Task set:

In this model each of the regions with the set of work tasks is called as task set. The size of the task set will vary according to the project.

Task region:

Task region is a region where set of task is achieved.

Principle of spiral model:

- Elaborate software entity object and find out constraints andalternatives.
- Elaborate definition of the software entity for theproject.
- Spiral model terminate a project, if it is toorisky.

Customer communication:

- Task requires establishing effective communication between developer and customer.

Planning:

- Task requires defining resources, time& other relatedinformation.

Risk information:

- Task requires to access both technical and managementrisk.

Engineering:

- Task required building one or more representation of theapplication.

Construction &release:

- The task requires constructing the project & releasing the project to thecustomer.

Customer evaluation:

- Task required obtaining customer feedback based on evolution of the software representation created during the installationstage.

Advantage:

- User will be able to see the project developmentcycle.
- Risk analysis which resolves higher priorityerror.
- Project is very muchrefined.

- Reusability of thesoftware.

Disadvantages:

- It is only suitable for large sizeproject.
- Model is more complex touse.
- Management skill is necessary so as to analyze the riskfactor.

SoftwareApplications:

- Software may be applied in any situation for which a pre-specified set of procedural steps (i.e., an algorithm) has been defined.
- Information content and determinacy are important factors in determining the nature of a softwareapplication.
- Content refers to the meaning and form of incoming outgoinginformation.
- Information determinacy refers to the predictability of the order and timing ofinformation.
- An engineering analysis program accepts data that have a predefined order executes the analysis algorithm without interruption and produces resultant data in report or graphical format.
- A multiuser operating system on the other hand accepts inputs that have varied content and arbitrary timing executes algorithms that can be interrupted by external conditions and produces output that varies as a function of environment andtime.
- It is somewhat difficult to develop meaningful generic categories for softwareapplications.
- As software complexity grows neat compartmentalizationdisappears. The following software areas indicate the breadth of potentialapplications:
- SystemSoftware.
- Real-TimeSoftware.
- BusinessSoftware.
- Engineering and ScientificSoftware.
- EmbeddedSoftware.
- Personal ComputerSoftware.
- Artificial IntelligenceSoftware.

System Software:

System software is a collection of programs written to service other programs. Some system software (e.g., compiler s editors and file management utilities) processes complex but determinate information structures. Other systems application (e.g., operating system components drivers telecommunications processors) process largely indeterminate data. In either case the systems software area is characterized by heavy interaction with computer hardware heavy usage by multiple users; concurrent operation that requires scheduling resource sharing and sophisticated process management.

Real-Time Software:

Programs that monitor/analyze/ control real world events as they occur are called real-time software. Elements of real-time software include a data gathering component that collects and formats information from an external environment an analysis component that transforms information as required by the application a control / output component that responds to the external environment so that real-time response (typically ranging from 1 millisecond to 1 minute) can be maintained. It should be noted that the term —real-time‖ differs from —interactive‖ or timesharing‖. A real-time system must respond within strict timeconstraints.

Business Software:

Business information processing is the largest single software application area. Discrete

—systems‖ have evolved into management information system (MIS) software that accesses one or more large databases containing business information. Applications in this area restructure existing data in a way that facilitates business operation or management decision making. In addition to conventional data processing applications, business software applications also encompass interactive and client/servercomputing.

Engineering and Scientific Software:

Engineering and Scientific software has been characterized by —number crunching‖ algorithms. Application range from astronomy to volcanologist from

automotive stress analysis to space shuttle orbital dynamics and from molecular biology to automated manufacturing. New applications with the engineering/scientific area are moving away from conventionalnumerical

algorithms. Computer aided design system simulation and other interactive applications have begun to take on real-time and even system software characteristics.

EmbeddedSoftware:

Intelligent products have become commonplace in nearly every consumer and industrial market. Embedded software resides in read only memory and isused to control products and systems for the consumer and industrial markets. Embedded software can perform very limited and esoteric functions (e.g., digital functions in an automobile such as fuel control, dashboard displays, braking systems,etc.).

Personal Computer Software:

The personal computer software market has burgeoned over the past decade. Word processing, spreadsheets, computer graphics, multimedia entertainment, database management personal and business financial applications and external network or database access are only a few of hundreds ofapplication.

Artificial Intelligence Software:

Artificial Intelligence (AI) software makes use of non numerical algorithms to solve complex problems that are not amenable to computation or straight forward analysis. An active AIarea is expert systems also called knowledge-based systems. However other application areas for AI software are pattern recognition (image and voice) theorem proving and game playing. In recent years a new branch of AI software called artificial neural networks, has evolved. A neural network simulates the structure of brain processes (the functions of the biological neuron) and may ultimately lead to a new class of software that can recognize complex patterns and learn frompast experience.

Types of Feasibility Study

A feasibility analysis evaluates the project's potential for success; therefore, perceived objectivity is an essential factor in the credibility of the study for potential investors and lending institutions. There are five types of feasibility study—separate areas that a feasibility study examines, described below.

1. Technical Feasibility

   This assessment focuses on the technical resources available to the organization. It helps organizations determine whether the technical resources meet capacity and whether the technical team is capable of converting the ideas into working systems. Technical feasibility also involves the evaluation of the hardware, software, and other technical requirements of the proposed system. As an exaggerated example, an organization wouldn't want to try to put Star Trek's transporters in their building—currently, this project is not technically feasible.

2. Economic Feasibility

   This assessment typically involves a cost/ benefits analysis of the project, helping organizations determine the viability, cost, and benefits associated with a project before financial resources are allocated. It also serves as an independent project assessment and enhances project credibility—helping decision-makers determine the positive economic benefits to the organization that the proposed project will provide.

3. Legal Feasibility

   This assessment investigates whether any aspect of the proposed project conflicts with legal requirements like zoning laws, data protection acts or social media laws. Let's say an organization wants to construct a new office building in a specific location. A feasibility study might reveal the organization's ideal location isn't zoned for that type of business. That organization has just saved considerable time and effort by learning that their project was not feasible right from the beginning.

4. Operational Feasibility

   This assessment involves undertaking a study to analyze and determine whether—and how well—the organization's needs can be met by completing the project. Operational feasibility studies also examine how a project plan satisfies the requirements identified in the requirements analysis phase of system development.

5. Scheduling Feasibility

   This assessment is the most important for project success; after all, a project will fail if not completed on time. In scheduling feasibility, an organization estimates how much time the project will take to complete.

When these areas have all been examined, the feasibility analysis helps identify any constraints the proposed project may face, including:

- Internal Project Constraints: Technical, Technology, Budget, Resource, etc.
- Internal Corporate Constraints: Financial, Marketing, Export, etc.
- External Constraints: Logistics, Environment, Laws, and Regulations, etc.

Need to Conduct Feasibility Study

Feasibility Studies Show the Viability of Your Proposal

Every groundbreaking proposal or development has started with an idea, and while some have defied the odds set against them, those ideas rarely went to work without first being evaluated. By looking at the landscape that surrounds your project, including where your potential customers would come from and who you are competing with to gain then, you'll be able to gauge the likelihood of achieving your definition of success. It's important to note that feasibility studies can also help you determine the costs required to successfully launch a project. So if a proposal is not realistic initially, adjustments can be made to increase its viability.

Feasibility Studies Help Define Your Goals

Ideas are great, but they are only as great as their execution. A feasibility study will help you see what goals you need to put in place to be successful by providing benchmarks for a project's viability. For example, if your community would like to build an indoor/outdoor sports facility, you may not have a clear picture of the construction costs. A feasibility study helps you understand your facility's costs as well as its revenue earning potential. With this information in hand you can either attain the resources needed to complete your project or "right-size" your project based on available resources.

Feasibility Studies Help You Develop A Plan

Like ideas, goals are only useful when they are put to work. As you define your goals, with the help of your feasibility study, they will give you a better understanding of what steps you need to take. You can then take those steps and create a plan for facility development.

Feasibility Studies Help Execute That Plan

Arguably the greatest benefit of a feasibility study is that they give you specific information about what a project requires for it to be sustainable. By understanding development costs, the competitive landscape, where potential customers will come from, and revenue potential, you'll have a feel for what resources must be procured and what actions your team must take to achieve success. The components of the feasibility study will serve as a road map describing the most optimal path to creating a new complex.

Feasibility Studies Will Give You an Identity

When planning a new sports, recreation, or entertainment facility, you'll have a general idea of who you are targeting. However, to attain this valued audience, you must understand their needs as well as the competitive landscape. How can you stand apart from competitors? A feasibility study will help you understand their offerings. This is an important component of building your identity.

Unit – 3

- **What is Requirement Engineering?**
- **Types of Requirements**
- **Requirement Elicitation Techniques**
- **Traditional and Modern Methods**
- **Verification and Validation Process**
- **Principles of requirement specification**
- **Characteristics of good SRS:**
- **Completeness, correct, Unambiguity, consistent , modifiable, traceable, understandable, IEEE SRS**

Software Requirements

The software requirements are description of features and functionalities of the target system. Requirements convey the expectations of users from the software product. The requirements can be obvious or hidden, known or unknown, expected or unexpected from client's point of view.

RequirementEngineering

The process to gather the software requirements from client, analyze, and document them is known as requirement engineering.

The goal of requirement engineering is to develop and maintain sophisticated and descriptive 'System Requirements Specification' document.

Requirement Engineering Process

It is a four step process, which includes –

- FeasibilityStudy
- RequirementGathering
- Software RequirementSpecification
- Software

Requirement

Validation Let us see

the process briefly-

Feasibility study

When the client approaches the organization for getting the desired product developed, it comes up with a rough idea about what all functions the software must perform and which all features are expected from the software.

Referencing to this information, the analysts do a detailed study about

whether the desired system and its functionality are feasible to develop.

This feasibility study is focused towards goal of the organization. This study analyzes whether the software product can be practically materialized in termsof implementation, contribution of project to organization, cost constraints, and as per values and objectives of the organization. It explores technical aspects ofthe

projectandproductsuchasusability,maintainability,productivity,andintegration ability.

The output of this phase should be a feasibility study report that should contain
adequatecommentsandrecommendationsformanagementaboutwhetheror not the project should beundertaken.

RequirementGathering

If the feasibility report is positive towards undertaking the project, next phase starts with gathering requirements from the user. Analysts and engineers communicate with the client and end-users to know their ideas on what the software should provide and which features they want the software to include.

Software Requirement Specification (SRS)

SRSisadocumentcreatedbysystemanalystaftertherequirementsarecollected from variousstakeholders.

SRS defines how the intended software will interact with hardware, external interfaces, speed of operation, response time of system, portability of software across various platforms, maintainability, speed of recovery after crashing, Security, Quality, Limitations etc.

The requirements received from client are written in natural language. It is the responsibility of the system analyst to document the requirements in technical language so that they can be comprehended and used by the software development team.

SRS should come up with the following features:

- User Requirements are expressed in naturallanguage.
- Technicalrequirementsareexpressedinstructuredlanguage,whichisused inside theorganization.
- Design description should be written in Pseudocode.
- Format of Forms and GUI screenprints.
- Conditional and mathematical notations for DFDsetc.

Software Requirement Validation

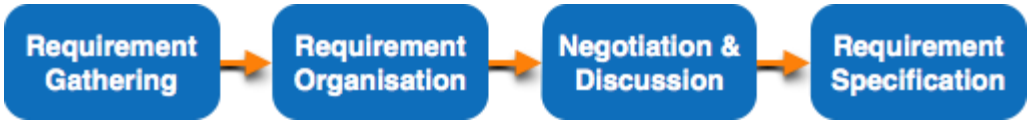
After requirement specifications are developed, the requirements mentioned in this document are validated. User might ask for illegal, impractical solution or expertsmayinterprettherequirementsinaccurately.Thisresultsinhugeincrease

in cost if not nipped in the bud. Requirements can be checked against following conditions -

- If they can be practicallyimplemented
- If they are valid and as per functionality and domain ofsoftware
- If there are anyambiguities
- If they arecomplete
- If they can bedemonstrated

Requirement Elicitation Process

Requirement elicitation process can be depicted using the folloiwng diagram:



- **Requirementsgathering-**Thedevelopersdiscusswiththeclientandend users and know their expectations from thesoftware.
- **Organizing Requirements -** The developers prioritize and arrange the requirements in order of importance, urgency andconvenience.
- **Negotiation & discussion -** If requirements are ambiguous or there

are someconflictsinrequirementsofvariousstakeholders,itisthennegotiated anddiscussedwiththestakeholders.Requirementsmaythenbeprioritized and reasonablycompromised.

The requirements come from various stakeholders. To remove the ambiguity and conflicts, they are discussed for clarity and correctness. Unrealistic requirements are compromised reasonably.

- **Documentation -** All formal and informal, functional and non-functional requirements are documented and made available for next phase processing.
-

Requirement Elicitation Techniques

Requirements Elicitation is the process to find out the requirements for an intendedsoftwaresystembycommunicatingwithclient,endusers,systemusers, and others who have a stake in the software systemdevelopment.

There are various ways to discover requirements. Some of them are explained below:

Interviews

Interviews are strong medium to collect requirements. Organization may conduct several types of interviews such as:

- Structured (closed) interviews, where every single information to gather is decided in advance, they follow pattern and matter of discussionfirmly.
- Non-structured (open) interviews, where information to gather is not decided in advance, more flexible and lessbiased.
- Oralinterviews
- Written interviews
- One-to-one interviews which are held between two persons across the table.
- Group interviews which are held between groups of participants. Theyhelp to uncover any missing requirement as numerous people areinvolved.

Surveys

Organizationmayconductsurveysamongvariousstakeholdersbyqueryingabout their expectation and requirements from the upcomingsystem.

Questionnaires

A document with pre-defined set of objective questions and respective options is handed over to all stakeholders to answer, which are collected and compiled.

A shortcoming of this technique is, if an option for some issue is not mentioned in the questionnaire, the issue might be left unattended.

Task analysis

Team of engineers and developers may analyze the operation for which the new system is required. If the client already has some software to perform certain operation, it is studied and requirements of proposed system are collected.

Domain Analysis

Every software falls into some domain category. The expert people in thedomain can be a great help to analyze general and specificrequirements.

Brainstorming

An informal debate is held among various stakeholders and all their inputs are recorded for further requirements analysis.

Prototyping

Prototyping is building user interface without adding detail functionality for user to interpret the features of intended software product. It helps giving better idea of requirements. If there is no software installed at client's end for developer's reference and the client is not aware of its own requirements, the developer creates a prototype based on initially mentioned requirements. The prototype is shown to the client and the feedback is noted. The client feedback serves as an input for requirement gathering.

Observation

Team of experts visit the client's organization or workplace. They observe the actualworkingoftheexistinginstalledsystems.Theyobservetheworkflowatthe client's end and how execution problems are dealt. The team itself draws

some conclusions which aid to form requirements expected from thesoftware.

Document analysis

Document analysis is one of the most helpful elicitation techniques in understanding the current process. Documents like user manuals, software vendor manuals, process documents about the current system can provide the inputs for the new system requirements.

Steps involved in document analysis are:

- Evaluating whether the existing system and business documents are appropriate to be studied.
- Analysing the documents to identify relevant business details.
- Reviewing and confirming identified details with subject matter experts.

There could be a lot of information that can be transferred to a new system requirements document. Evaluating the documentation can assist in making the As-Is process document, and conducting GAP analysis for scoping of the project in question.

Observation

This elicitation technique helps in collecting requirements by observing users or stakeholders. This can provide information about the exiting process, inputs and outputs. There are two kinds of observations — active and passive.

In active observation, the business analyst directly observes the users or stakeholders, whereas in passive observation the business analyst observes the subject matter experts.

This helps the business team understand the requirements when users are unable to explain requirements clearly.

Interview

An interview is a systematic approach to elicit information from a person or group of people. In this case, the business analyst acts as an interviewer. An interview provides an opportunity to explore and/or clarify requirements in more detail. Without knowing the expectations and goals of the stakeholders it is difficult to fulfil requirements.

Prototyping

Screen mockups can support the requirement gathering process, when introduced at the correct time. Mockups help stakeholders visualize the functionality of a system. This can be an advantage to business analysts and stakeholders since this allows them to identify gaps/problems early on.

Brainstorming

Brainstorming is an efficient way to define their requirements. Users can come up with very innovative ideas or requirements. This can help gather ideas and creative solutions from stakeholders in a short time.
Users or stakeholders can come up with ideas that they have seen or experienced elsewhere. These ideas can be reviewed and the relevant ones can then be included in the system requirements.

Workshop

Workshops comprise a group of users or stakeholders working together to identify requirements. A requirement workshop is a structured way to capture requirements. Workshops are used to scope, discover, define, and prioritize requirements for the proposed system.

They are the most effective way to deliver high-quality requirements quickly. They promote mutual understanding and strong communication between users or stakeholders and the project team.

JAD (Joint Application Development)

Joint Application Development (JAD) technique is an extended session to the workshop. In the JAD session stakeholders and project team works together to identify the requirements. These sessions allow the business team to gather and consolidate large amounts of information. Identification of stakeholders is the critical to the overall success of the JAD session. The JAD team includes business process owners, client representatives, ssers or stakeholders, business analysts, project managers, IT experts (developers, quality assurance, designers, and security).

Reverse engineering

This elicitation technique is generally used in migration projects. If an existing system has outdated documentation, it can be reverse engineered to understand what the system does. This is an elicitation technique that can extract implemented requirements from the system.
There are two types of reverse engineering techniques.
Black box reverse engineering: The system is studied without examining its internal structure (function and composition of software).
White box reverse engineering: The inner workings of the system are studied (analysing and understanding of software code).

Surveys/Questionnaires

Questionnaires are useful when there is a lot of information to be gathered from a larger group of stakeholders. This enables the business team to gather requirements

from stakeholders remotely. The design of the questionnaire is very important, since it can influence the answers that people provide.

In addition to the above-mentioned elicitation techniques, there are many more are on the market. It is very difficult to say that which elicitation technique is suitable for all projects. Not all elicitation techniques can be executed for every project.

When selecting an elicitation method, factors such as the nature of the project, organizational structure and type of stakeholders are taken into account by the business team before deciding which technique works best. Having said that, brainstorming, document analysis, interviews, prototyping and workshops are the most widely used requirement elicitation techniques.

Facilitated Application Specification Technique:

It's objective is to bridge the expectation gap – difference between what the developers think they are supposed to build and what customers think they are going to get.

A team oriented approach is developed for requirements gathering.

Each attendee is asked to make a list of objects that are-

1. Part of the environment that surrounds the system
2. Produced by the system
3. Used by the system

Each participant prepares his/her list, different lists are then combined, redundant entries are eliminated, team is divided into smaller sub-teams to develop mini-specifications and finally a draft of specifications is written down using all the inputs from the meeting.

Quality Function Deployment:

In this technique customer satisfaction is of prime concern, hence it emphasizes on the requirements which are valuable to the customer.

3 types of requirements are identified –

- Normal requirements – In this the objective and goals of the proposed software are discussed with the customer. Example – normal requirements for a result management system may be entry of marks, calculation of results etc
- Expected requirements – These requirements are so obvious that the customer need not explicitly state them. Example – protection from unauthorised access.
- Exciting requirements – It includes features that are beyond customer's expectations and prove to be very satisfying when present. Example – when an unauthorised access is detected, it should backup and shutdown all processes.

The major steps involved in this procedure are –

1. Identify all the stakeholders, eg. Users, developers, customers etc
2. List out all requirements from customer.
3. A value indicating degree of importance is assigned to each requirement.
4. In the end the final list of requirements is categorised as –
    - It is possible to achieve
    - It should be deferred and the reason for it
    - It is impossible to achieve and should be dropped off

Use Case Approach:

This technique combines text and pictures to provide a better understanding of the requirements.

The use cases describe the 'what', of a system and not 'how'. Hence they only give a functional view of the system.

The components of the use case deign includes three major things – Actor, Use cases, use case diagram.

1. Actor – It is the external agent that lies outside the system but interacts with it in some way. An actor maybe a person, machine etc. It is represented as a stick figure. Actors can be primary actors or secondary actors.
    - Primary actors – It requires assistance from the system to achieve a goal.
    - Secondary actor – It is an actor from which the system needs assistance.
2. Use cases – They describe the sequence of interactions between actors and the system. They capture who(actors) do what(interaction) with the system. A complete set of use cases specifies all possible ways to use the system.
3. Use case diagram – A use case diiagram graphically represents what happens when an actor interacts with a system. It captures the functional aspect of the system.
    - A stick figure is used to represent an actor.

- An oval is used to represent a use case.
- A line is used to represent a relationship between an actor and a use case.

For more information on use case diagram, refer to – Designing Use Cases for a Project

- Brainstorming: create a list of ideas using words and/or pictures.
- Interviews: formal or informal, in person or virtually.
- Document Analysis: review any recent documentation, schematics, lessons learned from prior projects that are similar to the current one, etc.
- Surveys or Questionnaires: SurveyMonkey or other automated (and easy to use) digital surveys can streamline the administration of surveys or questionnaires; you’ll need to be judicious about how you phrase your questions and the types of responses that will collect the right information, e.g., open-ended vs. closed-ended questions, etc.
- Workshops or Focus Groups: if you need input from SMEs or end-users in the same department or a cluster of targeted consumers that share similar end-user characteristics, then this may be the best choice.
- Prototyping: humans are heavily reliant on visual inputs, so creating a picture, model, or mock-up of the product is a great elicitation tool.
- **Observation:** if you have a prototype or mockup, you can observe the stakeholders using the product, and then ask clarifying questions.

By no means are you locked into using one type of tool or technique for a stakeholder or group of stakeholders during your elicitation session. You can read the guide to the 2020 Engineering Tool Stack here. If you’re interviewing an SME on a one-to-one basis, you can collaboratively brainstorm, review documents (with several questions you’ve generated after your initial document analysis), and send them a follow-up survey. You can do the same for workshops or focus groups.

Once you have your tools list, you can further refine and define your approach by generating a document that answers the following questions:

1. What information are you eliciting?
2. Where can you find the information (or from who)?

3. Which method is best for obtaining the information based on the stakeholder analysis and elicitation tools available?
4. When is the best time to conduct the elicitation sessions (also based on your answers to the prior questions)?

Software Requirements Characteristics

Gathering software requirements is the foundation of the entire software development project. Hence they must be clear, correct, and well-defined.

A complete Software Requirement Specifications must be:

- Clear
- Correct
- Consistent
- Coherent
- Comprehensible
- Modifiable
- Verifiable
- Prioritized
- Unambiguous
- Traceable
- Crediblesource

Software Requirements

We should try to understand what sort of requirements may arise in the requirement elicitation phase and what kinds of requirement are expected from the software system.

Broadly software requirements should be categorized in two categories:

Functional Requirements

Requirements, which are related to functional aspect of software fall into this category.

They define functions and functionality within and from the software system. EXAMPLES -

- Search option given to user to search from variousinvoices.
- User should be able to mail any report tomanagement.
- Users can be divided into groups and groups can be given separaterights.
- Should comply business rules and administrativefunctions.
- Software is developed keeping downward compatibilityintact.

Non-Functional Requirements

Requirements,whicharenotrelatedtofunctionalaspectofsoftware,fallintothis category. They are implicit or expected characteristics of software, which users make assumptionof.

Non-functional requirements include -

- Security
- Logging
- Storage
- Configuration
- Performance
- Cost
- Interoperability
- Flexibility
- Disasterrecovery
- Accessibility

Requirements are categorized logically as:

- **Must Have** : Software cannot be said operational withoutthem.
- **Should have** : Enhancing the functionality ofsoftware.
- **Could have** : Software can still properly function with theserequirements.
- **Wish list** : These requirements do not map to any objectives ofsoftware.

While developing software, 'Must have' must be implemented, 'Should have' is a matterofdebatewithstakeholdersandnegation,whereas'Couldhave'and'Wis h list' can be kept for softwareupdates.

User Interface requirements

User Interface (UI) is an important part of any software or hardware or hybrid system. A software is widely accepted if it is –

- easy tooperate

- quick in response
- effectively handling operationalerrors
- providing simple yet consistent userinterface

User acceptance majorly depends upon how user can use the software. UI is the only way for users to perceive the system. A well performing software system must also be equipped with attractive, clear, consistent, and responsive user interface. Otherwise the functionalities of software system can not be used in convenient way. A system is said to be good if it provides means to use it efficiently. User interface requirements are briefly mentioned below –

- Contentpresentation
- EasyNavigation
- Simpleinterface
- Responsive
- Consistent UIelements
- Feedbackmechanism
- Defaultsettings
- Purposefullayout
- Strategical use of color andtexture.
- Provide helpinformation
- User centricapproach
- Group based viewsettings.

Software System Analyst

System analyst in an IT organization is a person, who analyzes the requirement ofproposedsystemandensuresthatrequirementsareconceivedanddocumented properly and acuurately. Role of an analyst starts during Software AnalysisPhase

of SDLC. It is the responsibility of analyst to make sure that the developed software meets the requirements of the client.

System Analysts have the following responsibilities:

- Analyzing and understanding requirements of intendedsoftware
- Understanding how the project will contribute to the organizational objectives
- Identify sources ofrequirement

- Validation ofrequirement
- Develop and implement requirement managementplan
- Documentation of business, technical, process, and productrequirements
- Coordination with clients to prioritize requirements and removeambiguity
- Finalizing acceptance criteria with client and otherstakeholders

Software Metrics andMeasures

SoftwareMeasurescanbeunderstoodasaprocessofquantifyingandsymbolizing various attributes and aspects ofsoftware.

Software Metrics provide measures for various aspects of software process and software product.

Software measures are fundamental requirements of software engineering. They notonlyhelptocontrolthesoftwaredevelopmentprocessbutalsoaidtokeepthe quality of ultimate productexcellent.

According to Tom DeMarco, a (Software Engineer), "You cannot control what you cannotmeasure."Byhissaying,itisveryclearhowimportantsoftwaremeasures are.

Let us see some software metrics:

- **Size Metrics -** Lines of Code (LOC) (), mostly calculated in thousands of delivered source code lines, denoted asKLOC.
- Function Point Count is measure of the functionality provided by the software. Function Point count defines the size of functional aspect of the software.
- **Complexity Metrics -** McCabe's Cyclomatic complexity quantifies the upperboundofthenumberofindependentpathsinaprogram,whichis

perceived as complexity of the program or its modules. It is representedin terms of graph theory concepts by using control flowgraph.

- **Quality Metrics -** Defects, their types and causes, consequence, intensity of severity and their implications define the quality of theproduct.
- Thenumberofdefectsfoundindevelopmentprocessandnumberofdefects reported by the client after the product is installed or delivered at client- end, define quality of theproduct.

- **ProcessMetrics-**InvariousphasesofSDLC,themethodsandtoolsused, the company standards and the performance of development are software processmetrics.
- **Resource Metrics -** Effort, time, and various resources used, represents metrics for resourcemeasurement.

Requirements Validation

Requirements validation is the process of checking that requirements defined for development, define the system that the customer really wants. To check issues related to requirements, we perform requirements validation. We usually use requirements validation to check error at the initial phase of development as the error may increase excessive rework when detected later in the development process.
In the requirements validation process, we perform a different type of test to check the requirements mentioned in the Software Requirements Specification (SRS), these checks include:

- Completeness checks
- Consistency checks
- Validity checks
- Realism checks
- Ambiguity checks
- Verifiability

The output of requirements validation is the list of problems and agreed on actions of detected problems. The lists of problems indicate the problem detected during the process of requirement validation. The list of agreed action states the corrective action that should be taken to fix the detected problem.

There are several techniques which are used either individually or in conjunction with other techniques to check to check entire or part of the system:

1. **Test case generation:**
   Requirement mentioned in SRS document should be testable, the conducted tests reveal the error present in the requirement. It is generally believed that if the test is difficult or impossible to design than, this usually means that requirement will be difficult to implement and it should be reconsidered.
2. **Prototyping:**
   In this validation techniques the prototype of the system is presented before the

end-user or customer, they experiment with the presented model and check if it meets their need. This type of model is generally used to collect feedback about the requirement of the user.

3. **Requirements Reviews:**
   In this approach, the SRS is carefully reviewed by a group of people including people from both the contractor organisations and the client side, the reviewer systematically analyses the document to check error and ambiguity.
4. **Automated Consistency Analysis:**
   This approach is used for automatic detection of an error, such as nondeterminism, missing cases, a type error, and circular definitions, in requirements specifications.
   First, the requirement is structured in formal notation then CASE tool is used to check in-consistency of the system, The report of all inconsistencies is identified and corrective actions are taken.
5. **Walk-through:**
   A walkthrough does not have a formally defined procedure and does not require a differentiated role assignment.
   - Checking early whether the idea is feasible or not.
   - Obtaining the opinions and suggestion of other people.
   - Checking the approval of others and reaching an agreement.

Software verification and validation is an important model that allows the team of software developers and testers to ensure the accurate development of the product throughout the development life cycle. Both **software verification and validation** help create a software product that efficiently meets the specified requirements of the customer and offers them optimum services.Therefore, let us discuss software verification in detail to better understand its relevance in the **software development life cycle (SDLC)**.

What is Software Verification?

Verification is the process of checking or verifying the credentials, data or information to confirm their credibility and accuracy.In the field of software engineering, software verification is defined as the process of evaluating software product, to ensure that

the development phase is being carried out accurately, to build the desired software product.

It is performed during the ongoing phase of software development, to ensure the detection of defects and faults in the early stage of the development life cycle and to determine whether it satisfies the requirements of the customer.

Software verification offers answers to our query of **"Are we building the software product in a right manner?"** However, if the software passes during the verification process, it does not guarantee its validity. It is highly possible that a software product goes well through the verification process, but might fail to achieve the desired requirements.

Features of Software Verification:

In the realm of the software industry, software verification plays an integral role in building the product as per the requirements and needs of the customer. Other features of software verification that signify its importance are:

- Performed during the early stages of the software development process to determine whether the software meets the specified requirements.
- Verification denotes precision of the end or final product.
- It conducts **software review**, walk through, inspection, and evaluates documents, **plans**, requirements, and specifications.
- It demonstrates the consistency, completeness, and correctness of the software during each stage of the software development life cycle.
- Software verification can be termed as the first stage of the **software testing life cycle (STLC)**.

Types of Verification:

Software verifications are of two types, each of which, is focused on verifying various aspects of the software and ensuring its high quality. Together, these two verification types ensure that the software conforms to and satisfies specified requirements. The two types of software verification are:

1. **Static Verification**: Static verification involves inspection of the code before its execution, which ensures that the software meets its specified requirements and specifications.
    - It is an analysis based approach, usually carried out by just making an analysis of static aspects of the software system, such as the code conventions, software metrics calculation, anti-pattern detection, etc.
    - It does not involve the operation of the system or **component**.
    - Static verification includes both **manual** as well as **automated testing** techniques, such as consistency techniques and measurement techniques.
2. **Dynamic Verification**: It concerns with the working behavior of the software and is being carried out along with the execution of software. Also, know by 'Test phase' dynamic verification executes **test data** on the software, to assess the behavior of the software.
    - Unlike static verification, dynamic verification involves **execution** of the system or its components.
    - The team selects a group of test cases consisting of tests data, which are then used to find out the output test results.
    - There are three subtypes of dynamic verification: **functional testing**, structural testing, and **random testing**.

How to Perform Software Verification?

The process of software verification involves the assessment of the development phase and intermediary software product, based on the pre-defined specifications and guidelines.

It ensures that the methodology used for software development adheres to the specified specifications, set-up before the development phase, to build the correct software product.

Thus, this process requires cross-checking of these pre-defined specifications with the partially developed software product, to satisfy the process of carrying out the development of the product, in a right way.

Approaches:

As stated above, the process of software verification involves reviews, walkthrough, and inspection, which together helps to evaluate the accuracy of the product and ensure its quality. Therefore, defined below are these approaches of software verification:

The process of software verification can be performed through following approaches:

1. **Reviews:** It is an organized way of examining the documents such as design specifications, requirements specifications, code, etc. by one or more than one person, to explore defects in the software.

2. **Walkthroughs:** It is, usually an informal way of evaluating the software product, where different teams, whether associated or non-associated with the software development, goes through the software product and have discussion on possible errors and other defects, present in the software.

3. **Inspection:** It is one of the preferred and most common methods of the **static testing**. Unlike, walkthroughs, it's a formal way of testing the software, through examination of documents, carried out by the skilled moderator. This

method usually, checklist, rules, entry & exist criteria along with preparation and sharing of reports, in order to take corrective & necessary actions.

Advantages of Software Verification:

The advantages offered by software verification are numerous. It is among those processes, which makes the process of software development easy and allows the team to create an end product that conforms to the rules and regulations, as well as customer's requirements. Other advantages of this process are:

- It helps reduce the number of defects found during the later stages of development.
- Verifying the software in its initial development phase allows the team to get a better understanding of the product.
- It helps reduce the chances of system failures and crashes.
- Assists the team in developing a software product that conforms to the specified requirements and design specifications.

Difference Between Software Verification and Validation:

Software verification and validation are two of the most important processes used for ensuring the quality and accuracy of the product. However, these are usually confused by people and used interchangeably. Hence, to help you here is a comparison between these two software testing techniques.

Software Verification	***Software Validation***
1. Software verification is the process of evaluating the development phase to ensure the accuracy of the software.	1. **Software validation** evaluates the software during the development phase to ensure its compliance with business requirements.
2. It is performed before the execution of software validation.	2. It is performed once the process of software verification is completed.
3. Verification ensures the product is being developed as per the requirements	3. Validation ensures that the software meets the user's needs and is fit to be

and design specification.	used by them.
4. Evaluates plans, requirement specification, documents, **test cases**, etc.	4. Evaluates the software product to verify its readiness to be released.

What is Software Requirement Specifications?

A software requirements specification (SRS) is a description of an agreement between the customer and the supplier/contractor about the functional and non-functional requirements of the software system to be developed. This document will be used as a reference base for the development process.

Usually, the customers are not able to understand the development process and not clear on what they want from the system, at the same time the development team does not understand customers' requirements. This communication gap is usually filled by the Business Analyst by preparing an SRS.

Typically, an Software Requirement Specifications document consists of following sections (this is based on IEEE guide to software requirements specifications and this list may vary from organization to organization)-

1. **Introduction** – This section is written for the requirements document i.e. the purpose, scope, etc. of the requirements document is included.
   - Purpose
   - Scope
   - Definitions, Acronyms, and Abbreviations
   - References
   - Overview

2. **Overall Description** – This section contains an overview of all the requirements of the system.
   - Product Perspective
   - Product Functions
   - User Characteristics
   - General Constraints
   - Assumptions and Dependencies

3. **Detailed Requirements** – This is the most important part of the document; it contains details of product requirements that the development team and testing team need to know in order to code and test the system respectively.
   1. External Interface Requirements

- User Interfaces
- Hardware Interfaces
- Software Interfaces
- Communication Interfaces

2. Functional Requirements
    - Module 1
        - Functional Requirement 1.1
        - Functional Requirement 1.2

          ...
    - Module 2
        - Functional Requirement 2.1
        - Functional Requirement 2.2

          ...
3. Performance Requirements
4. Design Constraints
5. Attributes
6. Other Requirements

Advantages or Uses of SRS

The following are some important advantages or uses of an SRS:

- It is an agreement between the client and the supplier on what the software is supposed to do.
- It includes all the functional and non-functional requirements of the system.
- SRS works as a base for any further development and testing documents that need to be prepared for the development and testing processes. These documents include design documents, test plans, test scenarios, etc.
- SRS is used for the schedule and cost estimation process.
- SRS acts as a basis for any further enhancements in the product.
- SRS is used for validation of the final product i.e. once the final product is ready, the client, the development team, and the testing team can be assured that the product matches all the requirements in the document.

Desirable Characteristics of an SRS

This section discusses, in brief, some characteristics that describe a good SRS:

Correct – Requirements should be correct and should reflect exactly what the client wants. Every requirement should be such that it is required in the final product.

User review is used to ensure the correctness of requirements stated in the SRS. SRS is said to be correct if it covers all the requirements that are actually expected from the system.

Complete – SRS is said to be complete if everything the software is supposed to do is covered in the document. It should include all the functional and non-functional requirements, the correct numbering of the pages, any diagrams if required.

Unambiguous – All the requirements should have the same interpretation.

Verifiable – SRS is said to be verifiable if and only if each requirement is verifiable; there must be a way to determine whether every requirement is met in the final product.

Consistent – SRS is said to be consistent if there aren't any conflicts between the requirements.

Ranking for importance and stability – Each requirement should be ranked for its importance or stability.

Modifiability – An SRS is said to have modifiability quality if it is capable of adapting changes in the future as much as possible.

Traceability – Tracing of requirements to a design document, particular source code module or test cases should be possible.

Design Independence – SRS should include options of design alternatives for the final system; it should not have any implementation details.

Testability – An SRS is said to be testable if it is easier for the testing team to design test plans, test scenarios, and test cases using an SRS.

Understandable by the customer – An SRS should be written with easy and clear language so that the customer can understand the document.

SRS Principles

Software requirements specification may be viewed as a representation process requirements are represented in a manner that ultimately leads to successful software implementation. Balzer and Goldman propose eight principle of good specification.

1. Separate functionality from implementation.

2. Develop a model of the desired behaviour of a system that encompasses data and the functional responses of a system to various stimuli from the environment.

3. Establish the context in which software operated by specifying the manner in which other system components interact with software.

4. Define the environment in which the system operates and indicate how a highly intertwined collection of agents react to stimuli in the environments (change to objects) produced by those agents.

5. A system specification must be a cognitive model rather than a design or implementation model. The cognitive model describes a system as perceived by its user's community.

6. A specification must be operational.

7. The system specification must be tolerant of incompleteness and augmentable. A specification is always a model an abstraction of some real (or envisioned) situation that is normally quite complex. Hence it will be incomplete and will exist at many levels of detail.

8. A specification must be localized and loosely coupled so establish the content and structure of a specification in a way that will enable it be amenable to change.

Unit – 4

- **DFD, ERD, Structure Chart, decision tree, decision table, Data Dictionaries, pseudo code, input and output design**
- **Modules concept, types of modules**
- **Qualities of good design**
- **Coupling – types of coupling**
- **Cohesion – types of cohesion**

Software Design Basics

Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.

For assessing user requirements, an SRS (Software Requirement Specification) document is created whereas for coding and implementation, there is a need of more specific and detailed requirements in software terms. The output of this process can directly be used into implementation in programming languages.

Software design is the first step in SDLC (Software Design Life Cycle), which moves the concentration from problem domain to solution domain. It tries to specify how to fulfill the requirements mentioned in SRS.

Software DesignLevels

Software design yields three levels of results:

- **Architectural Design -** The architectural design is the highest abstract version of the system. It identifies the software as a system with many componentsinteractingwitheachother.Atthislevel,thedesignersgett he idea of proposed solutiondomain.
- **High-level Design -** The high-level design breaks the 'single entity-multiple component' concept of architectural design into less-abstracted view of sub-systems and modules and depicts their interaction with each other. High-level design focuses on how the system along with all of its components can be implemented in forms of modules. It recognizes modular structure of each sub-system and their relation and interaction among eachother.
- **Detailed Design-** Detailed design deals with the implementation part of what is seen as a system and its sub-systems in the previous two designs. It is more detailed towards modules and their implementations. It defines logical structure of each module and their interfaces to communicate with othermodules.

Modularization

Modularization is a technique to divide a software system into multiple

discrete and independent modules, which are expected to be capable of carrying out task(s)independently.Thesemodulesmayworkasbasicconstructsfortheentire software.Designerstendtodesignmodulessuchthattheycanbeexecutedand/or compiled separately andindependently.

Modular design unintentionally follows the rule of 'divide and conquer' problem- solving strategy, this is because there are many other benefits attached with the modular design of asoftware.

Advantage ofmodularization:

- Smaller components are easier tomaintain
- Program can be divided based on functionalaspects
- Desired level of abstraction can be brought in theprogram
- Components with high cohesion can be re-usedagain
- Concurrent execution can be madepossible
- Desired from securityaspect

Concurrency

Back in time, all software are meant to be executed sequentially. By sequential execution, we mean that the coded instruction will be executed one afteranother implying only one portion of program being activated at any given time. Say, a software has multiple modules, then only one of all the modules can be found active at any time ofexecution.

In software design, concurrency is implemented by splitting the software into multiple independent units of execution, like modules and executing them in parallel. In other words, concurrency provides capability to the software to execute more than one part of code in parallel to each other.

It is necessary for the programmers and designers to recognize those modules, which can be made parallel execution.

Example

The spell check feature in word processor is a module of software, which runs along side the word processor itself.

Coupling andCohesion

When a software program is modularized, its tasks are divided into several modules based on some characteristics. As we know, modules are set of instructions put together in order to achieve some tasks. They are though,

consideredasasingleentitybut,mayrefertoeachothertoworktogether.There are measures by which the quality of a design of modules and their interaction
amongthemcanbemeasured.Thesemeasuresarecalledcouplingandcohesion
.

Cohesion

Cohesion is a measure that defines the degree of intra-dependability within
elementsofamodule.Thegreaterthecohesion,thebetteristheprogramdesign.

There are seven types of cohesion, namely –

- **Co-incidental cohesion -** It is unplanned and random cohesion, which might be the result of breaking the program into smaller modules for the sake of modularization. Because it is unplanned, it may serve confusion to the programmers and is generallynot-accepted.
- **Logical cohesion -** When logically categorized elements are put together into a module, it is called logicalcohesion.
- **Emporal Cohesion -** When elements of module are organized such that
  theyareprocessedatasimilarpointoftime,itiscalledtemporalcohesion
  .
- **Procedural cohesion -** When elements of module are grouped together, which are executed sequentially in order to perform a task, it is called proceduralcohesion.
- **Communicational cohesion -** When elements of module are grouped together, which are executed sequentially and work on same data (information), it is called communicationalcohesion.
- **Sequential cohesion -** When elements of module are grouped because the output of one element serves as input to another and so on, it iscalled sequentialcohesion.
- **Functional cohesion -** It is considered to be the highest degree of cohesion, and it is highly expected. Elements of module in functional cohesion are grouped because they all contribute to a single well-defined function. It can also bereused.

Coupling

Couplingisameasurethatdefinesthelevelofinter-

dependabilityamongmodules of a program. It tells at what level the modules interfere and interact with each other. The lower the coupling, the better theprogram.

There are five levels of coupling, namely -

- **Content coupling -** When a module can directly access or modify or refer to the content of another module, it is called content levelcoupling.
- **Commoncoupling-** Whenmultiplemoduleshavereadandwriteaccessto some global data, it is called common or globalcoupling.
- **Control coupling-** Two modules are called control-coupled if one of them decides the function of the other module or changes its flow ofexecution.
- **Stamp coupling-** When multiple modules share common data structure and work on different part of it, it is called stampcoupling.
- **Data coupling-** Data coupling is when two modules interact with each other by means of passing data (as parameter). If a module passes data structure as parameter, then the receiving module should use all its components.

Ideally, no coupling is considered to be the best.

DesignVerification

The output of software design process is design documentation, pseudo codes, detailed logic diagrams, process diagrams, and detailed description of all functional or non-functional requirements.

The next phase, which is the implementation of software, depends on all outputs mentioned above.

It is then becomes necessary to verify the output before proceeding to the next phase. The early any mistake is detected, the better it is or it might not be detected until testing of the product. If the outputs of design phase are in formal notationform,thentheirassociatedtoolsforverificationshouldbeusedotherwise a thorough design review can be used for verification andvalidation.

By structured verification approach, reviewers can detect defects that might be caused by overlooking some conditions. A good design review is important for good software design, accuracy, and quality.

Characteristics of a good software design | Software Engineering

For good quality software to be produced, the software design must also be of good quality. Now, the matter of concern is how the quality of good software design is measured? This is done by observing certain factors in software design. These factors are:

1. Correctness
2. Understandability
3. Efficiency
4. Maintainability

Now, let us define each of them in detail,

1) Correctness

First of all, the design of any software is evaluated for its correctness. The evaluators check the software for every kind of input and action and observe the results that the software will produce according to the proposed design. If the results are correct for every input, the design is accepted and is considered that the software produced according to this design will function correctly.

2) Understandability

The software design should be understandable so that the developers do not find any difficulty to understand it. Good software design should be self- explanatory. This is because there are hundreds and thousands of developers that develop different modules of the software, and it would be very time consuming to explain each design to each developer. So, if the design is easy and self- explanatory, it would be easy for the developers to implement it and build the same software that is represented in the design.

3) Efficiency

The software design must be efficient. The efficiency of the software can be estimated from the design phase itself, because if the design is describing software that is not efficient and useful, then the developed software would also stand on the same level of efficiency. Hence, for efficient and good quality software to be developed, care must be taken in the designing phase itself.

4) Maintainability

The software design must be in such a way that modifications can be easily made in it. This is because every software needs time to time modifications and maintenance. So, the design of the software must also be able to bear such changes. It should not be the case that after making some modifications the other features of the software start misbehaving. Any change made in the software design must not affect the other available features, and if the features are getting affected, then they must be handled properly.

Software Analysisand Design Tools

Softwareanalysisanddesignincludesallactivities,whichhelpthetransformation of requirement specification into implementation. Requirement specifications specify all functional and non-functional expectations from the software. These requirement specifications come in the shape of human readable and understandable documents, to which a computer has nothing todo.

Software analysis and design is the intermediate stage, which helps human- readable requirements to be transformed into actual code.

Let us see few analysis and design tools used by software designers:

Data FlowDiagram

Data Flow Diagram (DFD) is a graphical representation of flow of data in an information system. It is capable of depicting incoming data flow, outgoing data flow, and stored data. The DFD does not mention anything about how data flows through the system.

ThereisaprominentdifferencebetweenDFDandFlowchart.Theflowchartdepicts flow of control in program modules. DFDs depict flow of data in the system at various levels. It does not contain any control or branchelements.

Types of DFD

Data Flow Diagrams are either Logical or Physical.

- **Logical DFD** - This type of DFD concentrates on the system process, and

flowofdatainthesystem.Forexampleinabankingsoftwaresystem,how data is moved between differententities.

- **Physical DFD** - This type of DFD shows how the data flow is actually implemented in the system. It is more specific and close to the implementation.

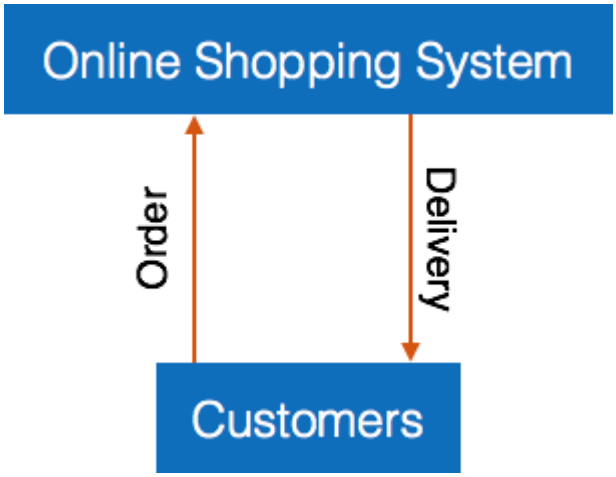
DFD Components

DFD can represent source, destination, storage, and flow of data using the following set of components -

- **Entities** - Entitiesaresourcesanddestinationsofinformationdata.Entities are represented by rectangles with their respectivenames.
- **Process**-ActivitiesandactiontakenonthedataarerepresentedbyCircle or Round-edgedrectangles.
- **Data Storage** - There are two variants of data storage - it can either be represented as a rectangle with absence of both smaller sides or as an open-sided rectangle with only one sidemissing.
- **Data Flow** - Movement of data is shown by pointed arrows. Data movement is shown from the base of arrow as its source towards head of the arrow asdestination.

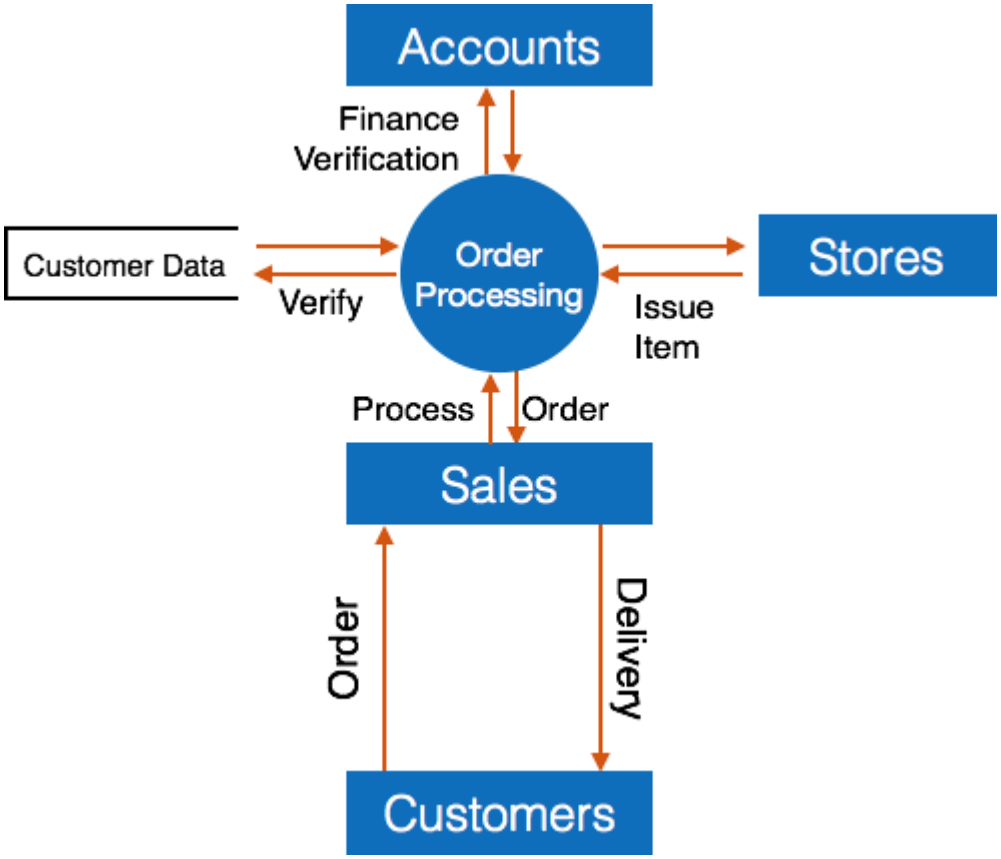
Levels of DFD

- **Level 0** - Highest abstraction level DFD is known as Level 0 DFD, which depicts the entire information system as one diagram concealing all the underlying details. Level 0 DFDs are also known



as context levelDFDs.

- **Level 1** - The Level 0 DFD is broken down into more specific, Level 1DFD. Level 1 DFD depicts basic modules in the system and flow of data among various modules. Level 1 DFD also mentions basic processes and sources ofinformation.



- **Level 2** - At this level, DFD shows how data flows inside the modules mentioned in Level1.

  Higher level DFDs can be transformed into more specific lower level DFDs with deeper level of understanding unless the desired level of specification is achieved.

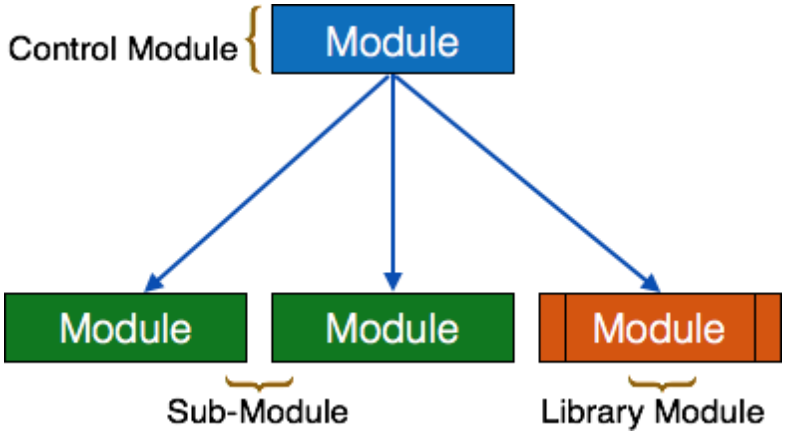
Structure Charts

Structure chart is a chart derived from Data Flow Diagram. It represents the system in more detail than DFD. It breaks down the entire system into lowest functional modules, describes functions and sub-functions of each module of the system to a greater detail than DFD.

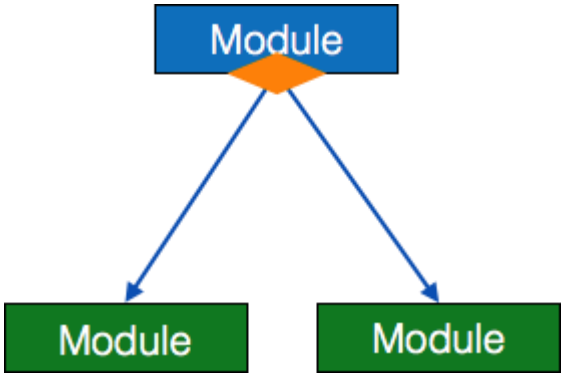
Structure chart represents hierarchical structure of modules. At each layer a specific task is performed.

Here are the symbols used in construction of structure charts -

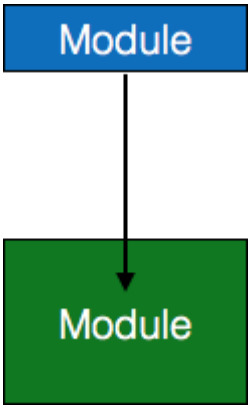
- **Module** - It represents process or subroutine or task. A control module branches to more than one sub-module. Library Modules are re-usableand invokable from anymodule.



- **Condition** - It is represented by small diamond at base of the module. It depicts that control module can select any of sub-routine based on some condition.

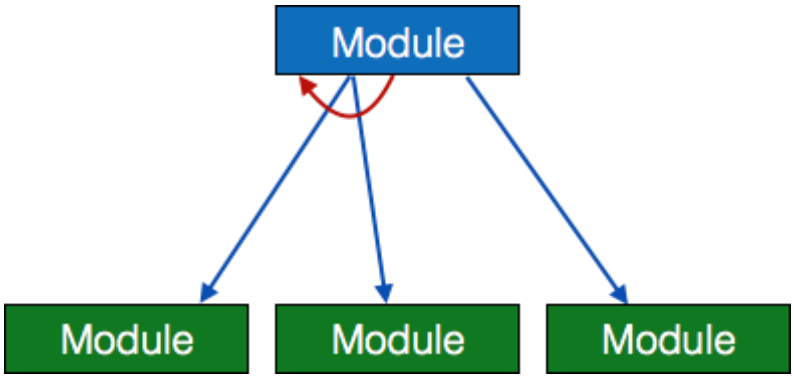


- **Jump** - An arrow is shown pointing inside the module to depict that the control will jump in the middle of thesub-module.

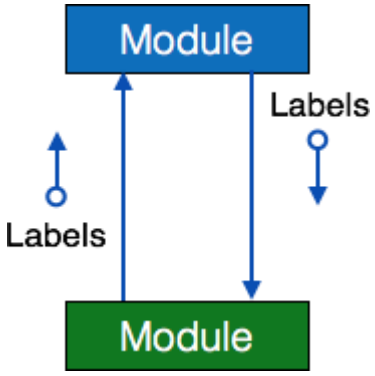


- **Loop** - A curved arrow represents loop in the module. All sub-

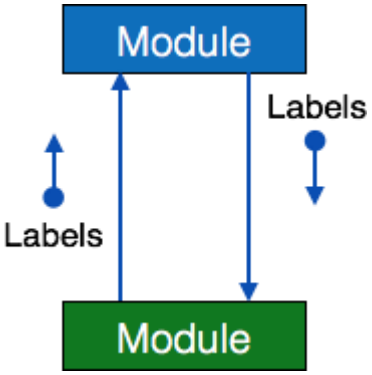
modules covered by loop repeat execution ofmodule.



- **Data flow** - A directed arrow with empty circle at the end represents data flow.



- **Control flow** - A directed arrow with filled circle at the end represents controlflow.



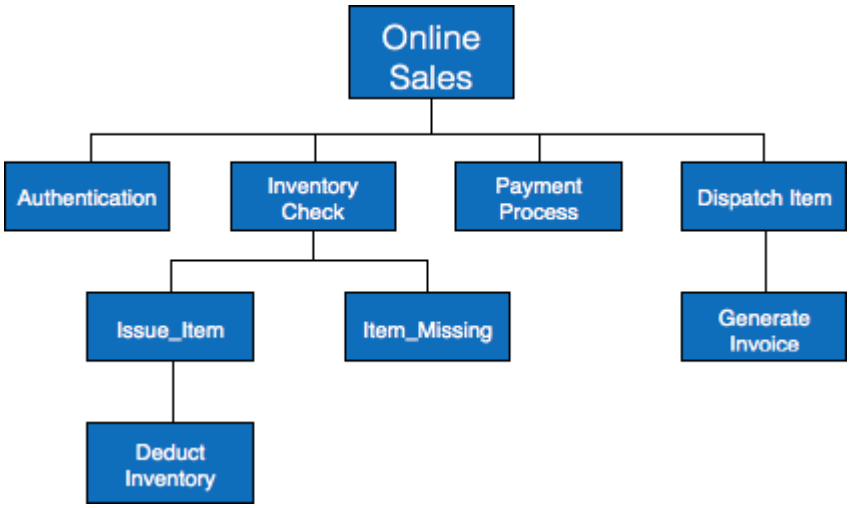
HIPODiagram

Hierarchical Input Process Output (HIPO) diagram is a combination of two organized methods to analyze the system and provide the means of documentation. HIPO model was developed by IBM in year 1970.

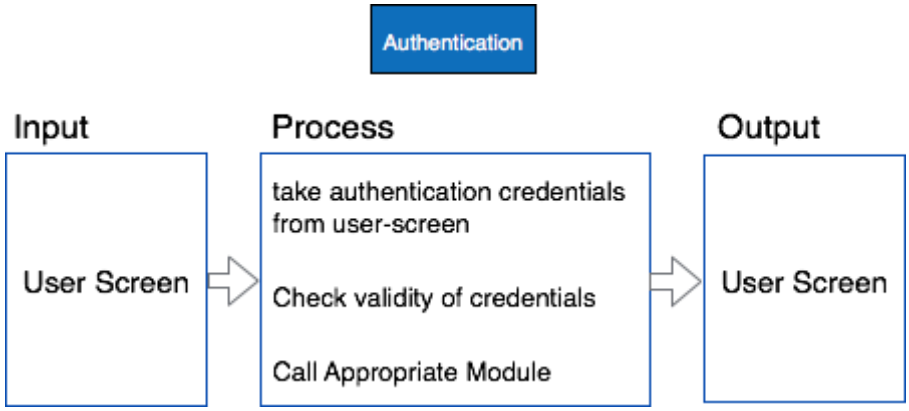
HIPOdiagramrepresentsthehierarchyofmodulesinthesoftwaresystem.Analyst usesHIPOdiagraminordertoobtainhigh-levelviewofsystemfunctions.It

decomposes functions into sub-functions in a hierarchical manner. It depicts the functions performed by system.

HIPO diagrams are good for documentation purpose. Their graphical representationmakesiteasierfordesignersandmanagerstogetthepictorialide a of the systemstructure.



In contrast to Input Process Output (IPO) diagram, which depicts the flow of control and data in a module, HIPO does not provide any information about data flow or control flow.



Example

BothpartsofHIPOdiagram,Hierarchicalpresentation,andIPOChartareusedfor structure designing of software program as well as documentation of thesame.

StructuredEnglish

Most programmers are unaware of the large picture of software so they

only rely on what their managers tell them to do. It is the responsibility of higher software management to provide accurate information to the programmers to develop accurate yet fast code.

Different methods, which use graphs or diagrams, at times might be interpreted in a different way by different people.

Hence, analysts and designers of the software come up with tools such as Structured English. It is nothing but the description of what is required to code and how to code it. Structured English helps the programmer to write error-free code. Here, both Structured English and Pseudo-Code tries to mitigate that understanding gap.

StructuredEnglishusesplainEnglishwordsinstructuredprogrammingparadigm. It is not the ultimate code but a kind of description what is required to code and how to code it. The following are some tokens of

```
IF-THEN-ELSE,

DO-WHILE-UNTIL
```

structuredprogramming:

Analyst uses the same variable and data name, which are stored in Data Dictionary, making it much simpler to write and understand the code.

Example

We take the same example of Customer Authentication in the online shopping environment. This procedure to authenticate customer can be written in Structured English as:

```
Enter Customer_Name

SEEK Customer_Namein Customer_Name_DBfile IF

Customer_Namefound THEN

   Call procedure USER_PASSWORD_AUTHENTICATE() ELSE

   PRINT error message

   Call procedure NEW_CUSTOMER_REQUEST()

ENDIF
```

The code written in Structured English is more like day-to-day spoken English. It can not be implemented directly as a code of software. Structured English is independent of programming language.

Pseudo-Code

Pseudocodeiswrittenmoreclosetoprogramminglanguage.Itmaybeconsidered as augmented programming language, full of comments, anddescriptions.

Pseudo code avoids variable declaration but they are written using some actual programming language's constructs, like C, Fortran, Pascal, etc.

Pseudo code contains more programming details than Structured English. It provides a method to perform the task, as if a computer is executing the code.

Example

Program to print Fibonacci up to n numbers.

```
void function Fibonacci
Get value of n;
Set value of a to 1;
Set value of b to 1;
Initialize I to 0 for
(i=0; i<n;i++)
{
   ifa greater than b
   {
      Increase b by a; Print b;
   }
   else if b greater than a
   {
      increase a by b;
      print a;
   }
}
```

Decision Tables

A Decision table represents conditions and the respective actions to be taken to address them, in a structured tabular format.

Itisapowerfultooltodebugandpreventerrors.Ithelpsgroupsimilarinformation into a single table and then by combining tables it delivers easy and convenient decision-making.

Creating Decision Table

To create the decision table, the developer must follow basic four steps:

- Identify all possible conditions to beaddressed
- Determine actions for all identifiedconditions
- Create Maximum possiblerules
- Define action for eachrule

Decision Tables should be verified by end-users and can lately be simplified by eliminating duplicate rules and actions.

Example

Let us take a simple example of day-to-day problem with our Internet connectivity.Webeginbyidentifyingallproblemsthatcanarisewhilestartingthe internet and their respective possiblesolutions.

Welistallpossibleproblemsundercolumnconditionsandtheprospectiveaction s under columnActions.

	Conditions/Actions	**Rules**							
Conditions	Shows Connected	N	N	N	N	Y	Y	Y	Y
	Ping is Working	N	N	Y	Y	N	N	Y	Y
	Opens Website	Y	N	Y	N	Y	N	Y	N
Actions	Check network cable	X							
	Check internet router	X				X	X	X	
	Restart Web Browser							X	
	Contact Service provider		X	X	X	X	X	X	
	Do no action								

Table : Decision Table – In-house Internet Troubleshooting

Decision Trees

Decision Tree Analysis is a general, predictive modelling tool that has applications spanning a number of different areas. In general, decision trees are constructed via an algorithmic approach that identifies ways to split a data set based on different conditions. It is one of the most widely used and practical methods for supervised learning. Decision Trees are a non-parametric supervised learning method used for both classification and regression tasks. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

The decision rules are generally in form of if-then-else statements. The deeper the tree, the more complex the rules and fitter the model.

Before we dive deep, let's get familiar with some of the terminologies:

- Instances: Refer to the vector of features or attributes that define the input space
- Attribute: A quantity describing an instance
- Concept: The function that maps input to output
- Target Concept: The function that we are trying to find, i.e., the actual answer
- Hypothesis Class: Set of all the possible functions
- Sample: A set of inputs paired with a label, which is the correct output (also known as the Training Set)
- Candidate Concept: A concept which we think is the target concept
- Testing Set: Similar to the training set and is used to test the candidate concept and determine its performance

Introduction

A decision tree is a tree-like graph with nodes representing the place where we pick an attribute and ask a question; edges represent the answers the to the question; and the leaves represent the actual output or class label. They are used in non-linear decision making with simple linear decision surface.

Decision trees classify the examples by sorting them down the tree from the root to some leaf node, with the leaf node providing the classification to the example. Each node in the tree acts as a test case for some attribute, and each edge descending from that node corresponds to one of the possible answers to the test case. This process is recursive in nature and is repeated for every subtree rooted at the new nodes.

Let's illustrate this with help of an example. Let's assume we want to play badminton on a particular day — say Saturday — how will you decide whether to play or not. Let's say you go out and check if it's hot or cold, check the speed of the wind and humidity, how the weather is, i.e. is it sunny, cloudy, or rainy. You take all these factors into account to decide if you want to play or not.

So, you calculate all these factors for the last ten days and form a lookup table like the one below.

Day	Weather	Temperature	Humidity	Wind	Play?
1	Sunny	Hot	High	Weak	No
2	Cloudy	Hot	High	Weak	Yes

Day	Weather	Temperature	Humidity	Wind	Play?
3	Sunny	Mild	Normal	Strong	Yes
4	Cloudy	Mild	High	Strong	Yes
5	Rainy	Mild	High	Strong	No
6	Rainy	Cool	Normal	Strong	No
7	Rainy	Mild	High	Weak	Yes
8	Sunny	Hot	High	Strong	No
9	Cloudy	Hot	Normal	Weak	Yes
10	Rainy	Mild	High	Strong	No

Table 1. Obeservations of the last ten days.

Now, you may use this table to decide whether to play or not. But, what if the weather pattern on Saturday does not match with any of rows in the table? This may be a problem. A decision tree would be a great way to represent data like this because it takes into account all the possible paths that can lead to the final decision by following a tree-like structure.

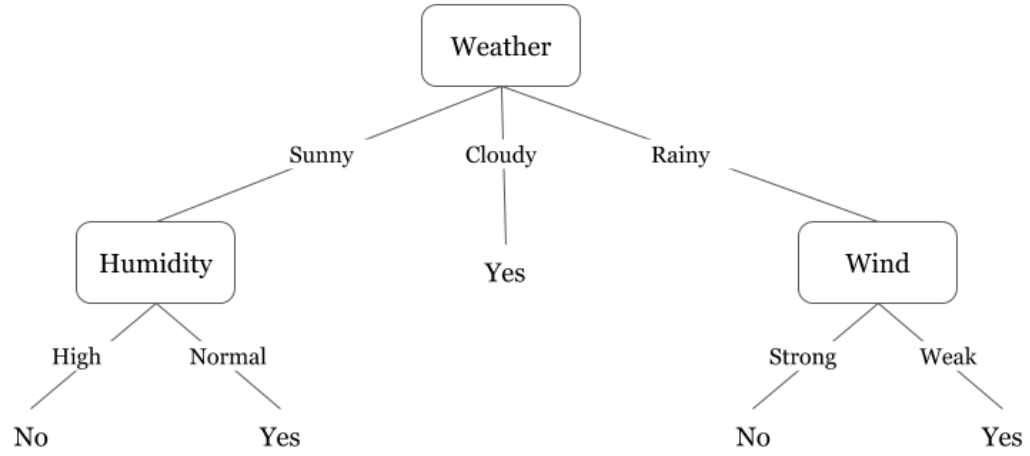


Fig 1. A decision tree for the concept Play Badminton

Fig 1. illustrates a learned decision tree. We can see that each node represents an attribute or feature and the branch from each node represents the outcome of that node. Finally, its the leaves of the tree where the final decision is made. If features are continuous, internal nodes can test the value of a feature against a threshold (see Fig. 2).

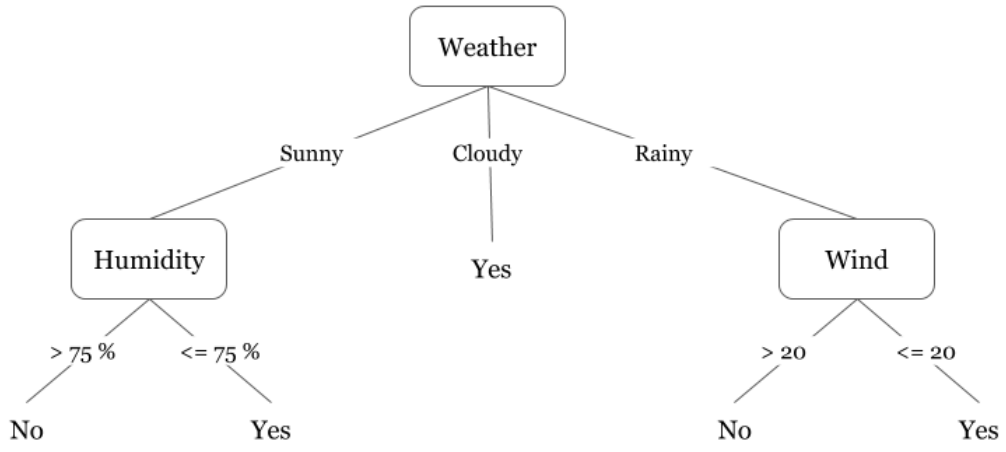


Fig 2. A decision tree for the concept Play Badminton (when attributes are continuous)

A general algorithm for a decision tree can be described as follows:

1. Pick the best attribute/feature. The best attribute is one which best splits or separates the data.

2. Ask the relevant question.
3. Follow the answer path.
4. Go to step 1 until you arrive to the answer.

The best split is one which separates two different labels into two sets.

Expressiveness of decision trees

Decision trees can represent any boolean function of the input attributes. Let's use decision trees to perform the function of three boolean gates AND, OR and XOR.

Boolean Function: AND

A	B	A AND B
F	F	F
F	T	F
T	F	F
T	T	T

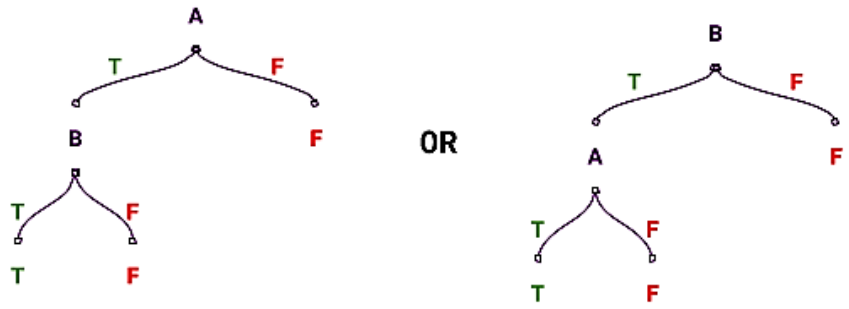


Fig 3. Decision tree for an AND operation.

In Fig 3., we can see that there are two candidate concepts for producing the decision tree that performs the AND operation. Similarly, we can also produce a decision tree that performs the boolean OR operation.

Boolean Function: OR

A	B	A OR B
F	F	F
F	T	T
T	F	T
T	T	T

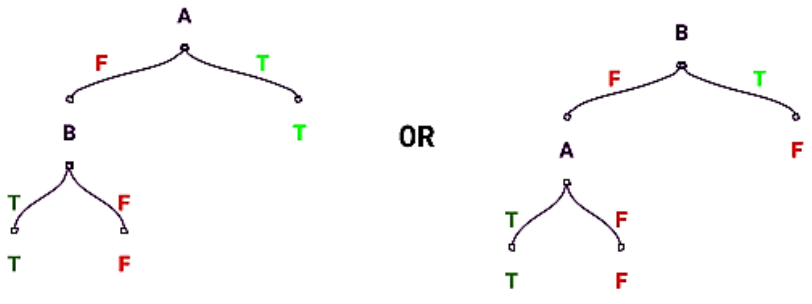


Fig 4. Decision tree for an OR operation

Boolean Function: XOR

A	B	A XOR B
F	F	F
F	T	T
T	F	T
T	T	F

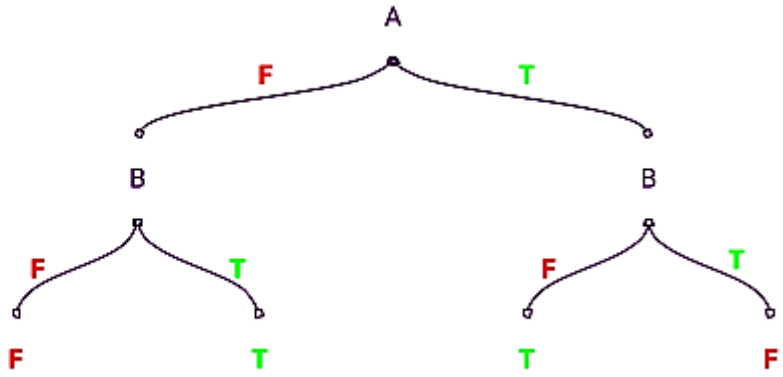


Fig 5. Decision tree for an XOR operation.

Let’s produce a decision tree performing XOR functionality using 3 attributes:

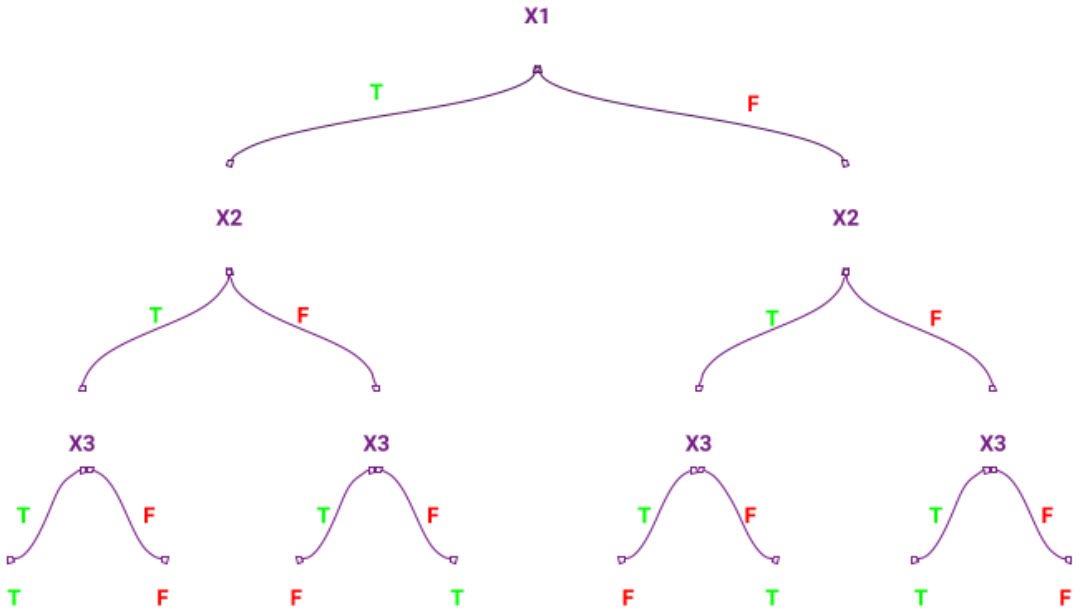


Fig 6. Decision tree for an XOR operation involving three operands

In the decision tree, shown above (Fig 6.), for three attributes there are 7 nodes in the tree, i.e., for $n = 3$, number of nodes = 2^3-1. Similarly, if we have n attributes, there are 2^n nodes (approx.) in the decision tree. So, the tree requires exponential number of nodes in the worst case.

We can represent boolean operations using decision trees. But, what other kind of functions can we represent and if we search over the various possible decision trees to find the right one, how many decision trees do we have to worry about. Let’s answer this question by finding out the possible number of decision trees we can generate given N different attributes (assuming the attributes are boolean). Since a truth table can be transformed into a decision tree, we will form a truth table of N attributes as input.

X1	X2	X3	....	XN	OUTPUT

X1	X2	X3	....	XN	OUTPUT
T	T	T	...	T	
T	T	T	...	F	
...	...	...	...	...	
...	...	...	...	...	
...	...	...	...	...	
F	F	F	...	F	

The above truth table has 2^n rows (i.e. the number of nodes in the decision tree), which represents the possible combinations of the input attributes, and since each node can a hold a binary value, the number of ways to fill the values in the decision tree is ${2^{2^n}}$. Thus, the space of decision trees, i.e, the hypothesis space of the decision tree is very expressive because there are a lot of different functions it can represent. But, it also means one needs to have a clever way to search the best tree among them.

Decision tree boundary

Decision trees divide the feature space into axis-parallel rectangles or hyperplanes. Let's demonstrate this with help of an example. Let's consider a simple AND operation on two variables (see Fig 3.). Assume X and Y to be the coordinates on the x and y axes, respectively, and plot the possible values of X and Y (as seen the table below). Fig 7. represents the formation of the decision boundary as each decision is taken. We can see that as each decision is made, the feature space gets divided into smaller rectangles and more data points get correctly classified.

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

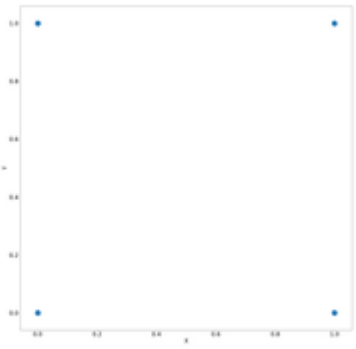


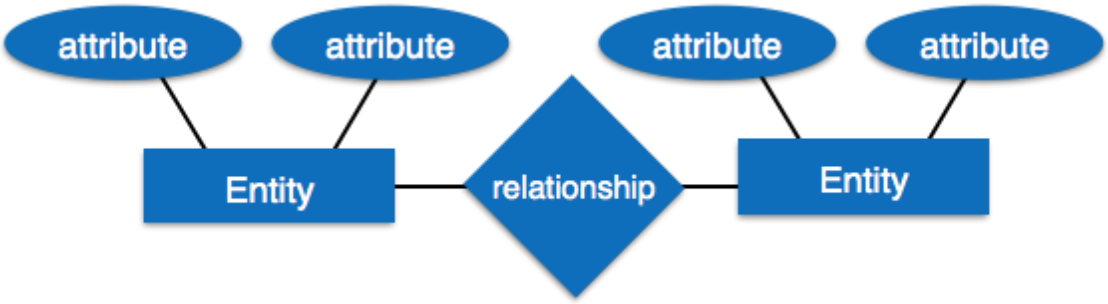
Fig 7.

Animation showing the formation of the

Entity-RelationshipModel

Entity-Relationshipmodelisatypeofdatabasemodelbasedonthenotionofreal worldentitiesandrelationshipamongthem.Wecanmaprealworldscenarioonto ER database model. ER Model creates a set of entities with their attributes, a set of constraints and relation amongthem.

ER Model is best used for the conceptual design of database. ER Model can be represented as follows :



- **Entity** - An entity in ER Model is a real world being, which has some propertiescalled***attributes***.Everyattributeisdefinedbyitscorrespondi ng set of values, called***domain***.

  For example, Consider a school database. Here, a student is an entity. Student has various attributes like name, id, age and class etc.

- **Relationship** - The logical association among entities is called ***relationship***. Relationships are mapped with entities in various ways. Mapping cardinalities define the number of associations between two entities.

Mapping cardinalities:

- one toone
- one tomany
- many toone
- many tomany

Data Dictionary

Data dictionary is the centralized collection of information about data. It stores meaningandoriginofdata,itsrelationshipwithotherdata,dataformatforusage, etc.Datadictionaryhasrigorousdefinitionsofallnamesinordertofacilitateuser and softwaredesigners.

Data dictionary is often referenced as meta-data (data about data) repository. It iscreatedalongwithDFD(DataFlowDiagram)modelofsoftwareprogramandis expected to be updated whenever DFD is changed orupdated.

Requirement of Data Dictionary

The data is referenced via data dictionary while designing and implementing software. Data dictionary removes any chances of ambiguity. It helps keeping

work of programmers and designers synchronized while using same object reference everywhere in the program.

Data dictionary provides a way of documentation for the complete database system in one place. Validation of DFD is carried out using data dictionary.

Contents

Data dictionary should contain information about the following:

- DataFlow
- DataStructure
- DataElements
- DataStores
- DataProcessing

Data Flow is described by means of DFDs as studied earlier and represented in algebraic form as described.

=	Composed of
{}	Repetition
()	Optional
+	And
[/]	Or

Example

Address = House No + (Street / Area) + City + State

Course ID = Course Number + Course Name + Course Level + Course Grades

Data Elements

Data elements consist of Name and descriptions of Data and Control Items, Internal or External data stores etc. with the following details:

- PrimaryName
- Secondary Name(Alias)
- Use-case (How and where touse)

- Content Description (Notation etc.)
- Supplementary Information (preset values, constraintsetc.)

Data Store

It stores the information from where the data enters into the system and exists out of the system. The Data Store may include -

- **Files**
  - Internal tosoftware.
  - External to software but on the samemachine.
  - External to software and system, located on differentmachine.
- **Tables**
  - Namingconvention
  - Indexingproperty

Data Processing

There are two types of Data Processing:

- **Logical:** As user seesit
- **Physical:** As software seesit

Software User InterfaceDesign

User interface is the front-end application view to which user interacts in orderto use the software. User can manipulate and control the software as well as hardware by means of user interface. Today, user interface is found at almost everyplacewheredigitaltechnologyexists,rightfromcomputers,mobilephones, cars, music players, airplanes, shipsetc.

Userinterfaceispartofsoftwareandisdesignedinsuchawaythatitisexpected to provide the user insight of the software. UI provides fundamental platform for human-computerinteraction.

UI can be graphical, text-based, audio-video based, depending upon the underlying hardware and software combination. UI can be hardware or software or a combination of both.

The software becomes more popular if its user interface is:

- Attractive
- Simple touse
- Responsive in shorttime
- Clear tounderstand
- Consistent on all

interfacing screens UI is broadly divided into twocategories:

- Command LineInterface
- Graphical UserInterface

Command Line Interface(CLI)

CLI has been a great tool of interaction with computers until the video display monitors came into existence. CLI is first choice of many technical users and programmers. It is the minimum interface a software can provide to its users.

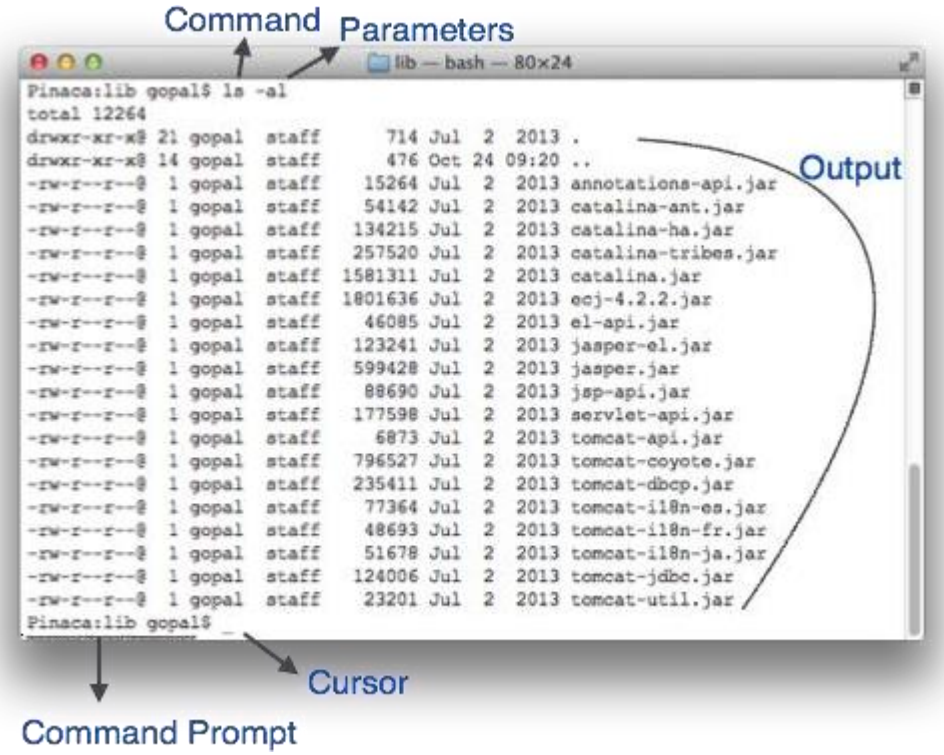
CLI provides a command prompt, the place where the user types the command and feeds to the system. The user needs to remember the syntax of command

anditsuse.EarlierCLIwerenotprogrammedtohandletheusererrorseffectively.

Acommandisatext-basedreferencetosetofinstructions,whichareexpectedto be executed by the system. There are methods like macros, scripts that make it easy for the user tooperate.

CLI uses less amount of computer resource as compared to GUI.

CLI Elements



A text-based command line interface can have the following elements:

- **Command Prompt** - It is text-based notifier that is mostly shows the contextinwhichtheuserisworking.Itisgeneratedbythesoftwaresyste m.
- **Cursor** - It is a small horizontal line or a vertical bar of the height of line, to represent position of character while typing. Cursor is mostly found in blinking state. It moves as the user writes or deletessomething.
- **Command** - A command is an executable instruction. It may have one or more parameters. Output on command execution is shown inline on the screen.Whenoutputisproduced,commandpromptisdisplayedonthen ext line.

Graphical UserInterface

GraphicalUserInterface(GUI)providestheusergraphicalmeanstointeractwith the system. GUI can be combination of both hardware and software. Using GUI, user interprets thesoftware.

Typically, GUI is more resource consuming than that of CLI. With advancing technology, the programmers and designers create complex GUI designs that work with more efficiency, accuracy, and speed.

GUI Elements

GUI provides a set of components to interact with software or hardware.

EverygraphicalcomponentprovidesawaytoworkwiththesystemAGUIsystem has following elements suchas:

Window - An area where contents of application are displayed. Contents in a windowcanbedisplayedintheformoficonsorlists,ifthewindowrepresentsfile structure. It is easier for a user to navigate in the file system in an exploring window. Windows can be minimized, resized or maximized to the size of screen. They can be moved anywhere on the screen. A window may contain another window of the same application, called childwindow.

- **Tabs** - If an application allows executing multiple instances of itself, they appearonthescreenasseparatewindows.**TabbedDocumentInterface** hascomeuptoopenmultipledocumentsinthesamewindow.Thisinterf

ace

also helps in viewing preference panel in application. All modern web- browsers use this feature.

- **Menu** - Menu is an array of standard commands, grouped together and placed at a visible place (usually top) inside the application window. The menu can be programmed to appear or hide on mouseclicks.
- **Icon**- Aniconissmallpicturerepresentinganassociatedapplication.When theseiconsareclickedordoubleclicked,theapplicationwindowisopened. Icondisplaysapplicationandprogramsinstalledonasystemintheformof smallpictures.
- **Cursor** - Interacting devices such as mouse, touch pad, digital pen are represented in GUI as cursors. On screen cursor follows the instructions fromhardwareinalmostreal-time.CursorsarealsonamedpointersinGUI systems. They are used to select menus, windows and other application features.

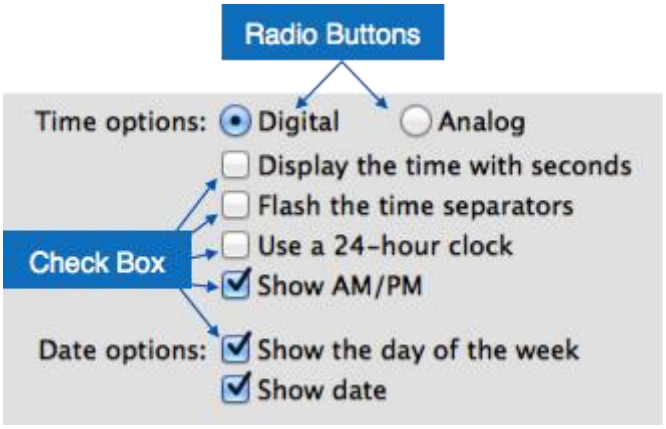
Application specific GUI components

A GUI of an application contains one or more of the listed GUI elements:

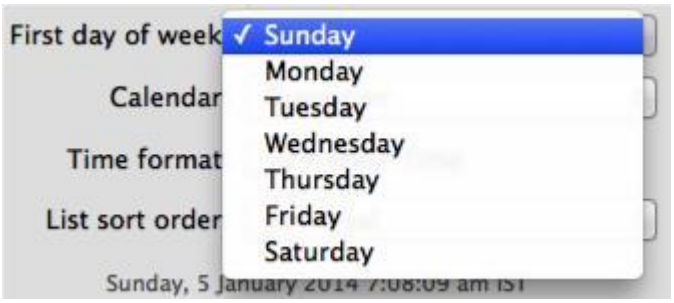
- **Application Window** - Most application windows uses the constructs supplied by operating systems but many use their own customer created windows to contain the contents ofapplication.
- **DialogueBox**-Itisachildwindowthatcontainsmessagefortheuserand request for some action to be taken. For Example: Application generate a dialogue to get confirmation from user to delete afile.

- **Text-Box** - Provides an area for user to type and enter text-baseddata.

- **Buttons** - They imitate real life buttons and are used to submit inputs to thesoftware.



- **Radio-button** - Displays available options for selection. Only one can be selected among alloffered.
- **Check-box** - Functions similar to list-box. When an option is selected, the boxismarkedaschecked.Multipleoptionsrepresentedbycheckboxesc an be selected.
- **List-box**- Provideslistofavailableitemsforselection.Morethanoneitem can



beselected.

Other impressive GUI components are:

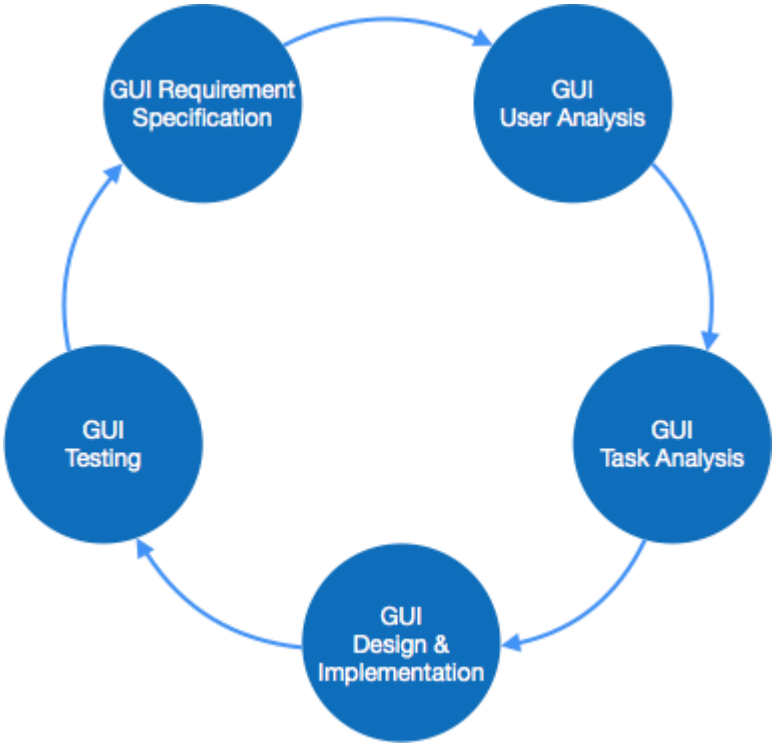
- Sliders
- Combo-box
- Data-grid
- Drop-downlist

User Interface DesignActivities

There are a number of activities performed for designing

user interface. The process of GUI design and implementation is alike SDLC. Any model can be used for GUI implementation among Waterfall, Iterative or Spiral Model.

A model used for GUI design and development should fulfill these GUI specific steps.



- **GUI Requirement Gathering** - The designers may like to have list of all functional and non-functional requirements of GUI. This can be taken from user and their existing softwaresolution.
- **UserAnalysis**-Thedesignerstudieswhoisgoingtousethesoftware GUI. The target audience matters as the design details change according to the knowledge and competency level of the user. If user is technical savvy, advanced and complex GUI can be incorporated. For a novice user, more information is included on how-to ofsoftware.
- **Task Analysis** - Designers have to analyze what task is to be done by the softwaresolution.HereinGUI,itdoesnotmatterhowitwillbedone.Task s can be represented in hierarchical manner taking one major task and dividing it further into smaller sub-tasks. Tasks provide goals for GUI presentation. Flow of information among sub-tasks determines the flow of GUI contents in thesoftware.

- **GUI Design and implementation** - Designers after having information about requirements, tasks and user environment, design the GUI and implementsintocodeandembedtheGUIwithworkingordummysoftware in the background. It is then self-tested by thedevelopers.

- **Testing** - GUI testing can be done in various ways. Organization can have in-houseinspection,directinvolvementofusersandreleaseofbetaversion are few of them. Testing may include usability, compatibility, user acceptanceetc.

GUI Implementation Tools

There are several tools available using which the designers can create entire GUI on a mouse click. Some tools can be embedded into the software environment (IDE).

GUI implementation tools provide powerful array of GUI controls. For software customization, designers can change the code accordingly.

There are different segments of GUI tools according to their different use and platform.

Example

MobileGUI,ComputerGUI,Touch-ScreenGUIetc.Hereisalistoffewtoolswhich come handy to buildGUI:

- FLUID
- AppInventor(Android)
- LucidChart
- Wavemaker
- VisualStudio

User Interface Goldenrules

Thefollowingrulesarementionedtobethegoldenrulesfor GUIdesign,described by Shneiderman and Plaisant in their book (Designing the UserInterface).

- **Strive for consistency** - Consistent sequences of actions should be required in similar situations. Identical terminology should be used in prompts, menus, and help screens. Consistent commands should be employedthroughout.

- **Enable frequent users to use short-cuts** - The user's desire to reduce the number of interactions increases with the frequency of

use. Abbreviations, function keys, hidden commands, and macro facilities are very helpful to an expertuser.

- **Offer informative feedback** - For every operator action, there should be some system feedback. For frequent and minor actions, the responsemust be modest, while for infrequent and major actions, the response must be moresubstantial.
- **Design dialog to yield closure** - Sequences of actions should be organized into groups with a beginning, middle, and end. The informative feedback at the completion of a group of actions gives the operators the satisfaction of accomplishment, a sense of relief, the signal to drop contingency plans and options from their minds, and this indicates thatthe way ahead is clear to prepare for the next group ofactions.
- **Offersimpleerrorhandling**-Asmuchaspossible,designthesystemso the user will not make a serious error. If an error is made, the system should be able to detect it and offer simple, comprehensible mechanisms for handling theerror.
- **Permiteasyreversalofactions**-Thisfeaturerelievesanxiety,sincethe userknowsthaterrorscanbeundone.Easyreversalofactionsencourage s exploration of unfamiliar options. The units of reversibility may be a single action, a data entry, or a complete group ofactions.

- **Support internal locus of control** - Experienced operators strongly desire the sense that they are in charge of the system and that thesystem respondstotheiractions.Designthesystemtomakeuserstheinitiatorso f actions rather than theresponders.
- **Reduce short-term memory load** - The limitation of human information processing in short-term memory requires the displays to be kept simple, multiple page displays to be consolidated, window-motion frequency be reduced,andsufficienttrainingtimebeallottedforcodes,mnemonics,a nd sequences ofactions.

Software Design Strategies

Software design is a process to conceptualize the software requirements into software implementation. Software design takes the user requirements as challenges and tries to find optimum solution. While the software is being conceptualized, a plan is chalked out to find the best possible design for implementing the intended solution.

There are multiple variants of software design. Let us study them briefly:

StructuredDesign

Structured design is a conceptualization of problem into several well-organized elements of solution. It is basically concerned with the solution design. Benefit of structured design is, it gives better understanding of how the problem is being solved.Structureddesignalsomakesitsimplerfordesignertoconcentrateonthe problem moreaccurately.

Structured design is mostly based on 'divide and conquer' strategy where a problem is broken into several small problems and each small problem is individually solved until the whole problem is solved.

The small pieces of problem are solved by means of solution modules.Structured designemphasisthatthesemodulesbewellorganizedinordertoachieveprecise solution.

These modules are arranged in hierarchy. They communicate with each other. A good structured design always follows some rules for communication among multiple modules, namely -

- **Cohesion** - grouping of all functionally relatedelements.
- **Coupling** - communication between differentmodules.

A good structured design has high cohesion and low coupling arrangements.

Function OrientedDesign

In function-oriented design, the system comprises of many smaller sub-

systems known as functions. These functions are capable of performing significant task in the system. The system is considered as top view of all functions.

Function oriented design inherits some properties of structured design where divide and conquer methodology is used.

This design mechanism divides the whole system into smaller functions, which provides means of abstraction by concealing the information and their operation. These functional modules can share information among themselves by means of information passing and using information available globally.

Another characteristic of functions is that when a program calls a function, the function changes the state of the program, which sometimes is not acceptableby other modules. Function oriented design works well where the system state does not matter and program/functions work on input rather than on astate.

Design Process

- Thewholesystemisseenashowdataflowsinthesystembymeansofdata flowdiagram.
- DFD depicts how functions change data and state of the entiresystem.
- The entire system is logically broken down into smaller units known as functions on the basis of their operation in thesystem.
- Each function is then described atlarge.

Object OrientedDesign

Object Oriented Design (OOD) works around the entities and their characteristics instead of functions involved in the software system. This design strategies focuses on entities and its characteristics. The whole concept of softwaresolution revolves around the engagedentities.

Let us see the important concepts of Object Oriented Design:

- **Objects -** All entities involved in the solution design are known as objects. For example, person, banks, company, and customers are treated as objects. Every entity has some attributes associated to it and has some methods to perform on theattributes.

- **Classes -** A class is a generalized description of an object. An object is an instanceofaclass.Classdefinesalltheattributes,whichanobjectcanhave and methods, which defines the functionality of theobject.

  In the solution design, attributes are stored as variables andfunctionalities are defined by means of methods orprocedures.

- **Encapsulation -** In OOD, the attributes (data variables) and methods (operation on the data) are bundled together is called encapsulation.
  Encapsulationnotonlybundlesimportantinformationofanobjecttogether, but also restricts access of the data and methods from the outside world. This is called informationhiding.

- **Inheritance -** OOD allows similar classes to stack up in hierarchical manner where the lower or sub-classes can import, implement and re-use allowed variables and methods from their immediate super classes. This property of OOD is known as inheritance. This makes it easier to define specific class and to create generalized classes from specificones.

- **Polymorphism -** OOD languages provide a mechanism where methods performing similar tasks but vary in arguments, can be assigned same name. This is called polymorphism, which allows a single interface performing tasks for different types. Depending upon how the function is invoked, respective portion of the code getsexecuted.

Design Process

Software design process can be perceived as series of well-defined steps.Though itvariesaccordingtodesignapproach(functionorientedorobjectoriented,yetIt may have the following stepsinvolved:

- A solution design is created from requirement or previous used system and/or system sequencediagram.
- Objects are identified and grouped into classes on behalf of similarity in attributecharacteristics.
- Class hierarchy and relation among them isdefined.
- Application framework isdefined.

Software Design Approaches

Here are two generic approaches for software designing:

Top Down Design

Weknowthatasystemiscomposedofmorethanonesub-systemsanditcontains a number of components. Further, these sub-systems and components mayhave their own set of sub-systems and components, and creates hierarchical structure in the system.

Top-down design takes the whole software system as one entity and then decomposesittoachievemorethanonesub-systemorcomponentbasedonsome characteristics. Each sub-system or component is then treated as a system and decomposed further. This process keeps on running until the lowest level of system in the top-down hierarchy isachieved.

Top-downdesignstartswithageneralizedmodelofsystemandkeepsondefining the more specific part of it. When all the components are composed the whole system comes intoexistence.

Top-downdesignismoresuitablewhenthesoftwaresolutionneedstobedesigned from scratch and specific details areunknown.

Bottom-up Design

The bottom up design model starts with most specific and basic components. It proceedswithcomposinghigherlevelofcomponentsbyusingbasicorlowerlevel components. It keeps creating higher level components until the desired system is not evolved as one single component. With each higher level, the amount of abstraction isincreased.

Bottom-up strategy is more suitable when a system needs to be created from some existing system, where the basic primitives can be used in the newer system.

Both,top-downandbottom-upapproachesarenotpracticalindividually.Instead, a good combination of both isused.

Unit – 5

Testing Techniques

- **Different testing techniques with examples**
- **Development and Execution of Test Cases: Debugging, Testing Tools & Environments**
- **Types of test cases and test plans**
- **Software Quality Concepts**
- **Quality Concepts**
- **What is Quality, Quality Control, Quality Assurance, Cost of Quality, Software Quality Assurance, Software Reviews**
- **Formal Technical Reviews**
- **The Review Meeting**
- **Review Reporting**
- **Record Keeping**
- **Review Guidelines**
- **Formal Approaches to SQA**
- **Statistical Quality Assurance**
- **Software Reliability**
- **SQA Plan**
- **SCM Process – Identification of objects, version control and change control.**

Software Testing Overview

SoftwareTestingisevaluationofthesoftwareagainstrequirementsgatheredfrom users and system specifications. Testing is conducted at the phase level in software development life cycle or at module level in program code. Software testing comprises of Validation andVerification.

SoftwareValidation

Validation is process of examining whether or not the software satisfies the user requirements. It is carried out at the end of the SDLC. If the software matches requirements for which it was made, it is validated.

- Validation ensures the product under development is as per the user requirements.
- Validation answers the question – "Are we developing the product which attempts all that user needs from this software?".
- Validation emphasizes on userrequirements.

SoftwareVerification

Verification is the process of confirming if the software is meeting the business requirements, and is developed adhering to the proper specifications and methodologies.

- Verification ensures the product being developed is according to design specifications.
- Verificationanswersthequestion– "Arewedevelopingthisproductbyfirmly following all design specifications?"
- Verifications concentrates on the design and system

specifications. Target of the test are-

- **Errors**- Theseareactualcodingmistakesmadebydevelopers.Inaddition, thereisadifferenceinoutputofsoftwareanddesiredoutput,isconsidered as anerror.

- **Fault** - When error exists fault occurs. A fault, also known as a bug, is a result of an error which can cause system tofail.
- **Failure** - failure is said to be the inability of the system to perform the desired task. Failure occurs when fault exists in thesystem.

Manual VsAutomatedTesting

Testing can either be done manually or using an automated testing tool:

- **Manual** - This testing is performed without taking help of automated testing tools. The software tester prepares test cases for different sections and levels of the code, executes the tests and reports the result to the manager.

  Manualtestingistimeandresourceconsuming.Thetesterneedstoconfirm whether or not right test cases are used. Major portion of testing involves manualtesting.
- **Automated** This testing is a testing procedure done with aid ofautomated testing tools. The limitations with manual testing can be overcome using automated testtools.

A test needs to check if a webpage can be opened in Internet Explorer. This can be easily done with manual testing. But to check if the web-server can take the load of 1 million users, it is quite impossible to test manually.

There are software and hardware tools which helps tester in conducting load testing, stress testing, regression testing.

Testing Approaches

Tests can be conducted based on two approaches –

1. Functionalitytesting
2. Implementationtesting

When functionality is being tested without taking the actual implementation in concern it is known as black-box testing. The other side is known as white-box testingwherenotonlyfunctionalityistestedbutthewayitisimplementedisalso analyzed.

Exhaustive tests are the best-desired method for a perfect testing. Every single possible value in the range of the input and output values is tested. It is not

possibletotesteachandeveryvalueinrealworldscenarioiftherangeofvalues is large.

Black-box testing

It is carried out to test functionality of the program and also called 'Behavioral' testing. The tester in this case, has a set of input values and respective desired results. On providing input, if the output matches with the desired results, the program is tested 'ok', and problematic

otherwise.

In this testing method, the design and structure of the code are not known tothe tester, and testing engineers and end users conduct this test on the software.

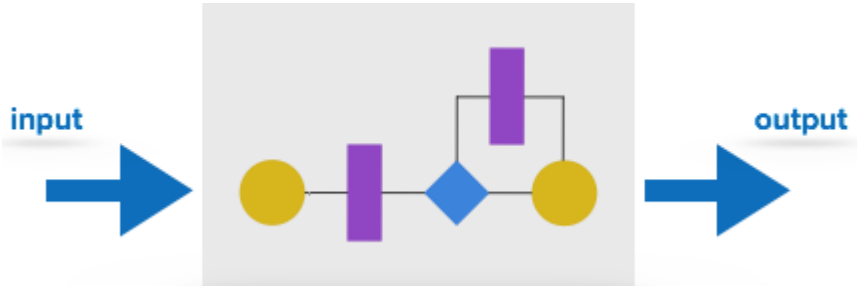
Black-box testing techniques:

- **Equivalence class** - The input is divided into similar classes. If one elementofaclasspassesthetest,itisassumedthatalltheclassispassed.
- **Boundary values** - The input is divided into higher and lower end values. If these values pass the test, it is assumed that all values in between may pass too.
- **Cause-effect graphing** - In both previous methods, only one input value at a time is tested. Cause (input) – Effect (output) is a testing technique where combinations of input values are tested in a systematicway.
- **Pair-wise Testing** - The behavior of software depends on multiple parameters. In pairwise testing, the multiple parameters are tested pair- wise for their differentvalues.
- **State-based testing** - The system changes state on provision of input. These systems are tested based on their states andinput.

White-box testing

It is conducted to test program and its implementation, in order to improve code efficiency or structure. It is also known as 'Structural'

testing.



In this testing method, the design and structure of the code are known to the tester. Programmers of the code conduct this test on the code.

The below are some White-box testing techniques:

- **Control-flow testing** - The purpose of the control-flow testing to set up test cases which covers all statements and branch conditions. The branch conditions are tested for both being true and false, so that all statements can becovered.
- **Data-flow testing** - This testing technique emphasis to cover all the data variables included in the program. It tests where the variables were declared and defined and where they were used orchanged.

TestingLevels

Testing itself may be defined at various levels of SDLC. The testing process runs parallel to software development. Before jumping on the next stage, a stage is tested, validated and verified.

Testing separately is done just to make sure that there are no hidden bugs or issues left in the software. Software is tested on various levels -

Unit Testing

While coding, the programmer performs some tests on that unit of program to know if it is error free. Testing is performed under white-box testing approach. Unit testing helps developers decide that individual units of the program are working as per requirement and are error free.

Integration Testing

Even if the units of software are working fine individually, there is a need to find outiftheunitsifintegratedtogetherwouldalsoworkwithouterrors.Forexample, argument passing and data updationetc.

System Testing

The software is compiled as product and then it is tested as a whole. This can be accomplished using one or more of the following tests:

- **Functionalitytesting**-Testsallfunctionalitiesofthesoftwareagainstthe requirement.
- **Performance testing** - This test proves how efficient the software is. It teststheeffectivenessandaveragetimetakenbythesoftwaretododesired task. Performance testing is done by means of load testing and stress testing where the software is put under high user and data load under various environmentconditions.
- **Security&Portability**-Thesetestsaredonewhenthesoftwareismeant to work on various platforms and accessed by number ofpersons.

Acceptance Testing

Whenthesoftwareisreadytohandovertothecustomerithastogothroughlast phase of testing where it is tested for user-interaction and response. This is important because even if the software matches all user requirements and ifuser does not like the way it appears or works, it may berejected.

- **Alpha testing** - The team of developer themselves perform alpha testing by using the system as if it is being used in work environment. They try to find out how user would react to some action in software and how the system should respond toinputs.
- **Beta testing** - After the software is tested internally, it is handed over to the users to use it under their production environment only for testing purpose. This is not as yet the delivered product. Developers expect that users at this stage will bring minute problems, which were skipped to attend.

Regression Testing

Whenever a software product is updated with new code, feature or functionality,
itistestedthoroughlytodetectifthereisanynegativeimpactoftheaddedcode.
This is known as regressiontesting.

TestingDocumentation

Testing documents are prepared at different stages -

Before Testing

Testing starts with test cases generation. Following documents are

needed for reference –

- **SRS document** - Functional Requirementsdocument
- **Test Policy document** - This describes how far testing should take place before releasing theproduct.
- **Test Strategy document** - This mentions detail aspects of test team, responsibility matrix and rights/responsibility of test manager and test engineer.
- **Traceability Matrix document** - This is SDLC document, which is related to requirement gathering process. As new requirements come, they are added to this matrix. These matrices help testers know the source of requirement. They can be traced forward andbackward.

While Being Tested

The following documents may be required while testing is started and is being done:

- **Test Case document** - This document contains list of tests required to be conducted.ItincludesUnittestplan,Integrationtestplan,Systemtestplan and Acceptance testplan.
- **Testdescription**-Thisdocumentisadetaileddescriptionofalltestcases and procedures to executethem.
- **Test case report** - This document contains test case report as a result of the test.
- **Test logs** - This document contains test logs for every test casereport.

After Testing

The following documents may be generated after testing :

- **Testsummary**-Thistestsummaryiscollectiveanalysisofalltestreports and logs. It summarizes and concludes if the software is ready to be launched. The software is released under version control system if it is ready tolaunch.

Testingvs.QualityControl&AssuranceandAudit

Testing vs. Quality Control & Assurance and Audit

We need to understand that software testing is different from software quality assurance, software quality control and software auditing.

- **Software quality assurance** - These are software development process monitoring means, by which it is assured that all the

measures are taken as per the standards of organization. This monitoring is done to make sure that proper software development methods werefollowed.

- **Software quality control** - This is a system to maintain the quality of software product. It may include functional and non-functional aspects of software product, which enhance the goodwill of the organization. This system makes sure that the customer is receiving quality product for their requirement and the product certified as 'fit foruse'.
- **Softwareaudit**-Thisisareviewofprocedureusedbytheorganizationto developthesoftware.Ateamofauditors,independentofdevelopmentteam examinesthesoftwareprocess,procedure,requirementsandotheraspects of SDLC. The purpose of software audit is to check that software and its development process, both conform standards, rules andregulations.

Software Quality Assurance (SQA)

What is Software Quality Assurance?

Software quality assurance (SQA) is a process which assures that all software engineering processes, methods, activities and work items are monitored and comply against the defined standards. These defined standards could be one or a combination of any like ISO 9000, CMMI model, ISO15504, etc.
SQA incorporates all software development processes starting from defining requirements to coding until release. Its prime goal is to ensure quality.

Software Quality Assurance Plan

Abbreviated as SQAP, the software quality assurance plan comprises of the procedures, techniques, and tools that are employed to make sure that a product or service aligns with the requirements defined in the SRS(software requirement specification).

The plan identifies the SQA responsibilities of a team, lists the areas that need to be reviewed and audited. It also identifies the SQA work products.

The SQA plan document consists of the below sections:

1. Purpose section
2. Reference section
3. Software configuration management section
4. Problem reporting and corrective action section
5. Tools, technologies and methodologies section
6. Code control section
7. Records: Collection, maintenance and retention section
8. Testing methodology

SQA Activities

Given below is the list of SQA activities:

#1) Creating an SQA Management Plan:

The foremost activity includes laying down a proper plan regarding how the SQA will be carried out in your project.

Along with what SQA approach you are going to follow, what engineering activities will be carried out, and it also includes ensuring that you have a right talent mix in your team.

#2) Setting the Checkpoints:

The SQA team sets up different checkpoints according to which it evaluates the quality of the project activities at each checkpoint/project stage. This ensures regular quality inspection and working as per the schedule.

#3) Apply software Engineering Techniques:

Applying some software engineering techniques aids a software designer in achieving high-quality specification. For gathering information, a designer may use techniques such as interviews and FAST (Functional Analysis System Technique).

Later, based on the information gathered, the software designer can prepare the project estimation using techniques like WBS (work breakdown structure), SLOC (source line of codes), and FP(functional point) estimation.

#4) Executing Formal Technical Reviews:

An FTR is done to evaluate the quality and design of the prototype.

In this process, a meeting is conducted with the technical staff to discuss regarding the actual quality requirements of the software and the design quality of the prototype. This activity helps in detecting errors in the early phase of SDLC and reduces rework effort in the later phases.

#5) Having a Multi- Testing Strategy:

By multi-testing strategy, we mean that one should not rely on any single testing approach, instead, multiple types of testing should be performed so that the software product can be tested well from all angles to ensure better quality.

#6) Enforcing Process Adherence:

This activity insists the need for process adherence during the software development process. The development process should also stick to the defined procedures.

This activity is a blend of two sub-activities which are explained below in detail:

(i) Product Evaluation:

This activity confirms that the software product is meeting the requirements that were discovered in the project management plan. It ensures that the set standards for the project are followed correctly.

(ii) Process Monitoring:

This activity verifies if the correct steps were taken during software development. This is done by matching the actually taken steps against the documented steps.

#7) Controlling Change:

In this activity, we use a mix of manual procedures and automated tools to have a mechanism for change control.

By validating the change requests, evaluating the nature of change and controlling the change effect, it is ensured that the software quality is maintained during the development and maintenance phases.

#8) Measure Change Impact:

If any defect is reported by the QA team, then the concerned team fixes the defect.

After this, the QA team should determine the impact of the change which is brought by this defect fix. They need to test not only if the change has fixed the defect, but also if the change is compatible with the whole project.

For this purpose, we use software quality metrics which allows managers and developers to observe the activities and proposed changes from the beginning till the end of SDLC and initiate corrective action wherever required.

#9) Performing SQA Audits:

The SQA audit inspects the entire actual SDLC process followed by comparing it against the established process.

It also checks whatever reported by the team in the status reports were actually performed or not. This activity also exposes any non-compliance issues.

#10) Maintaining Records and Reports:

It is crucial to keep the necessary documentation related to SQA and share the required SQA information with the stakeholders. The test results, audit results, review reports, change requests documentation, etc. should be kept for future reference.

#11) Manage Good Relations:

In fact, it is very important to maintain harmony between the QA and the development team.

We often hear that testers and developers often feel superior to each other. This should be avoided as it can affect the overall project quality.

Software Quality Assurance Standards

In general, SQA may demand conformance to one or more standards.

Some of the most popular standards are discussed below:

ISO 9000: This standard is based on seven quality management principles which help the organizations to ensure that their products or services are aligned with the customer needs'.

7 principles of ISO 9000 are depicted in the below image:

CMMI level:

COST OF QUALITY (COQ)

Quality Glossary Definition: Cost of quality

Cost of quality (COQ) is defined as a methodology that allows an organization to determine the extent to which its resources are used for activities that prevent poor quality, that appraise the quality of the organization's products or services, and that result from internal and external failures. Having such information allows an organization to determine the potential savings to be gained by implementing process improvements.

WHAT IS COST OF POOR QUALITY (COPQ)?

Cost of poor quality (COPQ) is defined as the costs associated with providing poor quality products or services. There are three categories:

1. Appraisal costs are costs incurred to determine the degree of conformance to quality requirements.
2. Internal failure costs are costs associated with defects found before the customer receives the product or service.
3. External failure costs are costs associated with defects found after the customer receives the product or service.

Quality-related activities that incur costs may be divided into prevention costs, appraisal costs, and internal and external failure costs.

Appraisal costs

Appraisal costs are associated with measuring and monitoring activities related to quality. These costs are associated with the suppliers' and customers' evaluation of purchased materials, processes, products, and services to ensure that they conform to specifications. They could include:

- Verification: Checking of incoming material, process setup, and products against agreed specifications
- Quality audits: Confirmation that the quality system is functioning correctly
- Supplier rating: Assessment and approval of suppliers of products and services

Internal failure costs

Internal failure costs are incurred to remedy defects discovered before the product or service is delivered to the customer. These costs occur when the results of work fail to reach design quality standards and are detected before they are transferred to the customer. They could include:

- Waste: Performance of unnecessary work or holding of stock as a result of errors, poor organization, or communication
- Scrap: Defective product or material that cannot be repaired, used, or sold
- Rework or rectification: Correction of defective material or errors
- Failure analysis: Activity required to establish the causes of internal product or service failure

External failure costs

External failure costs are incurred to remedy defects discovered by customers. These costs occur when products or services that fail to reach design quality standards are not detected until after transfer to the customer. They could include:

- Repairs and servicing: Of both returned products and those in the field
- Warranty claims: Failed products that are replaced or services that are re-performed under a guarantee
- Complaints: All work and costs associated with handling and servicing customers' complaints
- Returns: Handling and investigation of rejected or recalled products, including transport costs

PREVENTION COSTS

Prevention costs are incurred to prevent or avoid quality problems. These costs are associated with the design, implementation, and maintenance of the quality management system. They are planned and incurred before actual operation, and they could include:

- Product or service requirements: Establishment of specifications for incoming materials, processes, finished products, and services
- Quality planning: Creation of plans for quality, reliability, operations, production, and inspection
- Quality assurance: Creation and maintenance of the quality system
- Training: Development, preparation, and maintenance of programs

COST OF QUALITY AND ORGANIZATIONAL OBJECTIVES

The costs of doing a quality job, conducting quality improvements, and achieving goals must be carefully managed so that the long-term effect of quality on the organization is a desirable one.

These costs must be a true measure of the quality effort, and they are best determined from an analysis of the costs of quality. Such an analysis provides a method of assessing the effectiveness of the management of quality and a means of determining problem areas, opportunities, savings, and action priorities.

Cost of quality is also an important communication tool. Philip Crosby demonstrated what a powerful tool it could be to raise awareness of the importance of quality. He referred to the measure as the "price of nonconformance" and argued that organizations choose to pay for poor quality.

Many organizations will have true quality-related costs as high as 15-20% of sales revenue, some going as high as 40% of total operations. A general rule of thumb is that costs of poor quality in a thriving company will be about 10-15% of operations. Effective quality improvement programs can reduce this substantially, thus making a direct contribution to profits.

The quality cost system, once established, should become dynamic and have a positive impact on the achievement of the organization's mission, goals, and objectives.

Cost of Quality Example

COST OF QUALITY RESOURCES

Using Cost of Quality to Improve Business Results (PDF) Since centering improvement efforts on cost of quality, CRC Industries has reduced failure dollars as a percentage of sales and saved hundreds of thousands of dollars.

Cost of Quality: Why More Organizations Do Not Use It Effectively (World Conference on Quality and Improvement) Quality managers in organizations that do not track cost of quality cite as reasons a lack of management support for quality control, time and cost of COQ tracking, lack of knowledge of how to track data, and lack of basic cost data.

The Tip of the Iceberg (*Quality Progress*) A Six Sigma initiative focused on reducing the costs of poor quality enables management to reap increased customer satisfaction and bottom-line results.

Cost of Quality (COQ): Which Collection System Should Be Used? (World Conference on Quality and Improvement) This article identifies the various COQ systems available and the benefits and disadvantages of using each system.

Quality Control (QC)

Quality Control (QC) refers to quality related activities associated with the process of creating products and services. Quality control is used to verify that pre-determined quality standards are being met through quality inspections and reviews which detect poor quality, and identify, non-conformance with those standards

Quality Assurance (QA)

Quality Assurance is the process used to assure the consumer that a firm's products or services will be fit for purpose, by preventing quality issues rather than detecting them. The objective is to meet quality standards at each stage of production to ensure customer satisfaction with the final product or service. Examples of quality assurance include process checklists, project audits and methodology and standards development. Quality assurance is recognised by the international standard **ISO 9000**.

Formal Technical Review (FTR) in Software Engineering

Formal Technical Review (FTR) is a software quality control activity performed by software engineers.
Objectives of formal technical review (FTR):
Some of these are:

- Useful to uncover error in logic, function and implementation for any representation of the software.
- The purpose of FTR is to verify that the software meets specified requirements.
- To ensure that software is represented according to predefined standards.
- It helps to review the uniformity in software that is development in a uniform manner.
- To makes the project more manageable.

In addition, the purpose of FTR is to enable junior engineer to observer the analysis, design, coding and testing approach more closely. FTR also works to promote back up and continuity become familiar with parts of software they might not have seen otherwise.

Actually, FTR is a class of reviews that include walkthroughs, inspections, round robin reviews and other small group technical assessments of software. Each FTR is conducted as meeting and is considered successfully only if it is properly planned, controlled and attended.

The review meeting:

Each review meeting should be held considering the following constraints-

Involvement of people:

1. Between 3, 4 and 5 people should be involve in the review.
2. Advance preparation should occur but it should be very short that is at the most 2 hours of work for every person.
3. The short duration of the review meeting should be less than two hour. Gives these constraints, it should be clear that an FTR focuses on specific (and small) part of the overall software.

At the end of the review, all attendees of FTR must decide what to do.

1. Accept the product without any modification.
2. Reject the project due to serious error (Once corrected, another app need to be reviewed), or
3. Accept the product provisional (minor errors are encountered and are should be corrected, but no additional review will be required).

The decision was made, with all FTR attendees completing a sign-of indicating their participation in the review and their agreement with the findings of the review team.

Review reporting and record keeping :-

1. During the FTR, the reviewer actively records all issues that have been raised.
2. At the end of the meeting all these issues raised are consolidated and a review list is prepared.
3. Finally, a formal technical review summary report is prepared.

It answers three questions :-

1. What was reviewed ?
2. Who reviewed it ?
3. What were the findings and conclusions ?

Review guidelines :-

Guidelines for the conducting of formal technical reviews should be established in advance. These guidelines must be distributed to all reviewers, agreed upon, and then

followed. A review that is unregistered can often be worse than a review that does not minimum set of guidelines for FTR.

1. Review the product, not the manufacture (producer).
2. Take written notes (record purpose)
3. Limit the number of participants and insists upon advance preparation.
4. Develop a checklist for each product that is likely to be reviewed.
5. Allocate resources and time schedule for FTRs in order to maintain time schedule.
6. Conduct meaningful training for all reviewers in order to make reviews effective.
7. Reviews earlier reviews which serve as the base for the current review being conducted.
8. Set an agenda and maintain it.
9. Separate the problem areas, but do not attempt to solve every problem notes.
10. Limit debate and rebuttal.

What is Software Configuration Management?

In Software Engineering, **Software Configuration Management(SCM)** is a process to systematically manage, organize, and control the changes in the documents, codes, and other entities during the Software Development Life Cycle. The primary goal is to increase productivity with minimal mistakes. SCM is part of cross-disciplinary field of configuration management and it can accurately determine who made which revision.

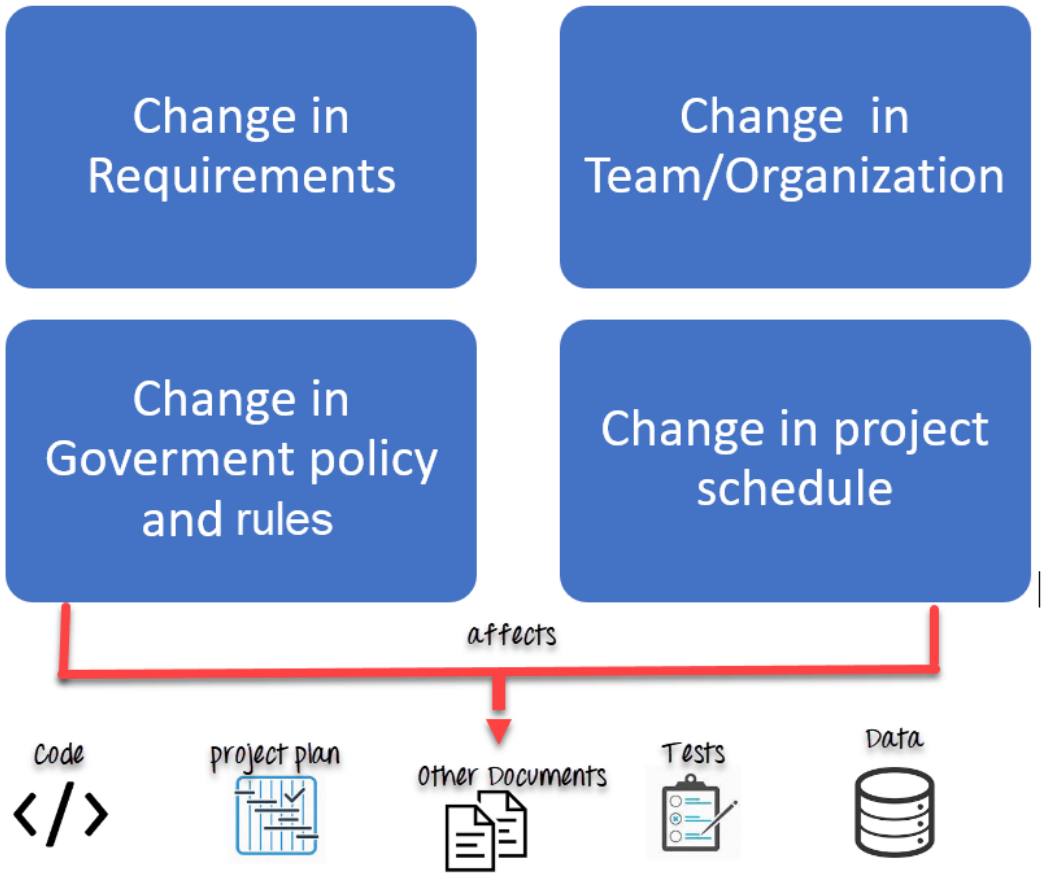
Software Configuration Item (SCI)

A SCI is a major software component of a system that is designated by the Buyer for configuration management to ensure integrity of the delivered product. It may exist at any level of the hierarchy, where interchangeability is required. Each SCI is to have (as appropriate) individual design reviews, individual qualification certification, individual acceptance reviews and separate user manuals

Why do we need Configuration management?

The primary reasons for Implementing Technical Software Configuration Management System are:

- There are multiple people working on software which is continually updating
- It may be a case where multiple version, branches, authors are involved in a software config project, and the team is geographically distributed and works concurrently
- Changes in user requirement, policy, budget, schedule need to be accommodated.
- Software should able to run on various machines and Operating Systems
- Helps to develop coordination among stakeholders
- SCM process is also beneficial to control the costs involved in making changes to a system



Any change in the software configuration Items will affect the final product. Therefore, changes to configuration items need to be controlled and managed.

Tasks in SCM process

- Configuration Identification
- Baselines
- Change Control
- Configuration Status Accounting
- Configuration Audits and Reviews

Configuration Identification:

Configuration identification is a method of determining the scope of the software system. With the help of this step, you can manage or control something even if you don't know what it is. It is a description that contains the CSCI type (Computer Software Configuration Item), a project identifier and version information.

Activities during this process:

- Identification of configuration Items like source code modules, test case, and requirements specification.

- Identification of each CSCI in the SCM repository, by using an object-oriented approach
- The process starts with basic objects which are grouped into aggregate objects. Details of what, why, when and by whom changes in the test are made
- Every object has its own features that identify its name that is explicit to all other objects
- List of resources required such as the document, the file, tools, etc.

Example:

Instead of naming a File login.phpits should be named login_v1.2.php where v1.2 stands for the version number of the file

Instead of naming folder "Code" it should be named "Code_D" where D represents code should be backed up daily.

Baseline:

A baseline is a formally accepted version of a software configuration item. It is designated and fixed at a specific time while conducting the SCM process. It can only be changed through formal change control procedures.

Activities during this process:

- Facilitate construction of various versions of an application
- Defining and determining mechanisms for managing various versions of these work products
- The functional baseline corresponds to the reviewed system requirements
- Widely used baselines include functional, developmental, and product baselines

In simple words, baseline means ready for release.

Change Control:

Change control is a procedural method which ensures quality and consistency when changes are made in the configuration object. In this step, the change request is submitted to software configuration manager.

Activities during this process:

- Control ad-hoc change to build stable software development environment. Changes are committed to the repository
- The request will be checked based on the technical merit, possible side effects and overall impact on other configuration objects.

- It manages changes and making configuration items available during the software lifecycle

Configuration Status Accounting:

Configuration status accounting tracks each release during the SCM process. This stage involves tracking what each version has and the changes that lead to this version.

Activities during this process:

- Keeps a record of all the changes made to the previous baseline to reach a new baseline
- Identify all items to define the software configuration
- Monitor status of change requests
- Complete listing of all changes since the last baseline
- Allows tracking of progress to next baseline
- Allows to check previous releases/versions to be extracted for testing

Configuration Audits and Reviews:

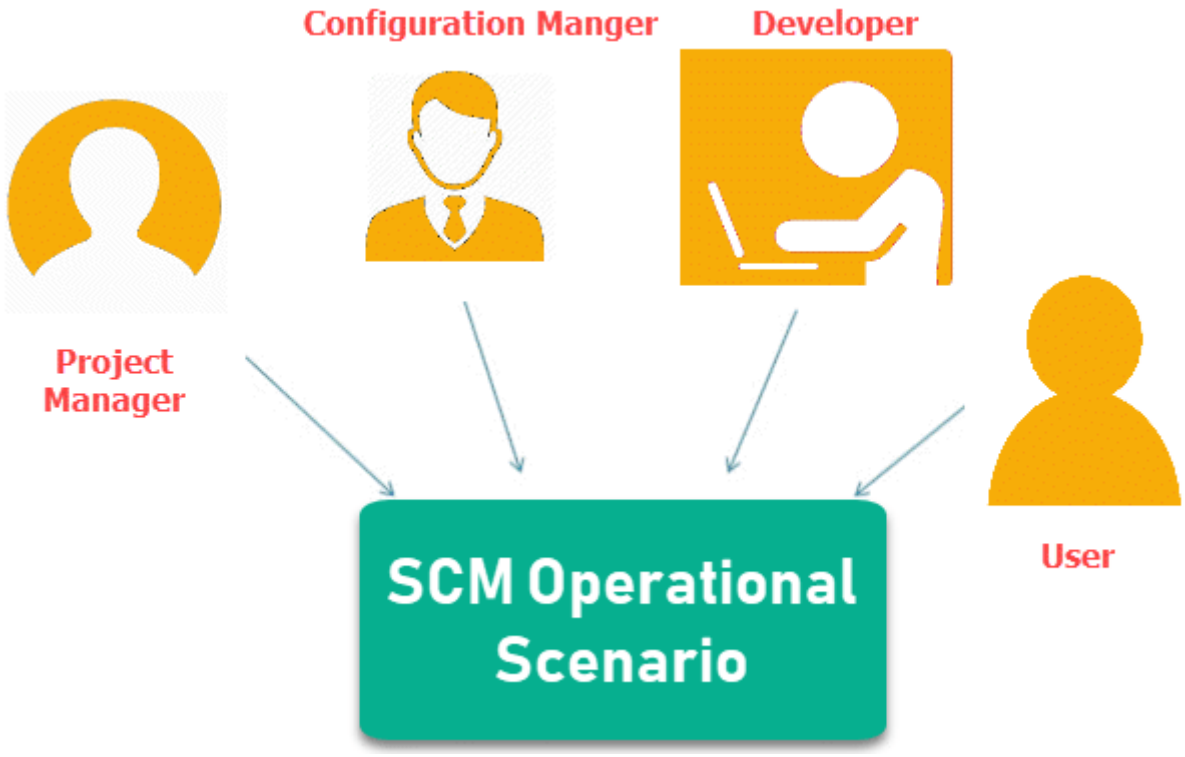
Software Configuration audits verify that all the software product satisfies the baseline needs. It ensures that what is built is what is delivered.

Activities during this process:

- Configuration auditing is conducted by auditors by checking that defined processes are being followed and ensuring that the SCM goals are satisfied.
- To verify compliance with configuration control standards. auditing and reporting the changes made
- SCM audits also ensure that traceability is maintained during the process.
- Ensures that changes made to a baseline comply with the configuration status reports
- Validation of completeness and consistency

Participant of SCM process:

Following are the key participants in SCM



1. Configuration Manager

- Configuration Manager is the head who is Responsible for identifying configuration items.
- CM ensures team follows the SCM process
- He/She needs to approve or reject change requests

2. Developer

- The developer needs to change the code as per standard development activities or change requests. He is responsible for maintaining configuration of code.
- The developer should check the changes and resolves conflicts

3. Auditor

- The auditor is responsible for SCM audits and reviews.
- Need to ensure the consistency and completeness of release.

4. Project Manager:

- Ensure that the product is developed within a certain time frame
- Monitors the progress of development and recognizes issues in the SCM process
- Generate reports about the status of the software system
- Make sure that processes and policies are followed for creating, changing, and testing

5. User

The end user should understand the key SCM terms to ensure he has the latest version of the software

Software Configuration Management Plan

The SCMP (Software Configuration management planning) process planning begins at the early coding phases of a project. The outcome of the planning phase is the SCM plan which might be stretched or revised during the project.

- The SCMP can follow a public standard like the IEEE 828 or organization specific standard
- It defines the types of documents to be management and a document naming. Example Test_v1
- SCMP defines the person who will be responsible for the entire SCM process and creation of baselines.
- Fix policies for version management & change control
- Define tools which can be used during the SCM process
- Configuration management database for recording configuration information.

Software Configuration Management Tools

Any Change management software should have the following 3 Key features:

Concurrency Management:

When two or more tasks are happening at the same time, it is known as concurrent operation. Concurrency in context to SCM means that the same file being edited by multiple persons at the same time.

If concurrency is not managed correctly with SCM tools, then it may create many pressing issues.

Version Control:

SCM uses archiving method or saves every change made to file. With the help of archiving or save feature, it is possible to roll back to the previous version in case of issues.

Synchronization:

Users can checkout more than one files or an entire copy of the repository. The user then works on the needed file and checks in the changes back to the repository.They can synchronize their local copy to stay updated with the changes made by other team members.

Unit – 6

- **What is Software Maintenance?**
- **Problems during Software Maintenance**
- **Categories of Software Maintenance: Corrective Maintenance,Adaptive Maintenance**
- **Perfective Maintenance and Preventive Maintenance**
- **Potential Solutions to Maintenance: Budget and efforts reallocation, complete replacement, maintenance of existing system**
- **Maintenance Process and Models: Maintenance Process, Fix Model, Iterative Enhancement Model, Reuse Oriented Model, Boehm Model and Taute's Models**

Software maintenance is widely accepted part of SDLC now a days. It stands for all the modifications and updations done after the delivery of software product. There are number of reasons, why modifications are required, some of them are briefly mentioned below:

- **Market Conditions** - Policies, which changes over the time, such as taxation and newly introduced constraints like, how to maintain bookkeeping, may trigger need formodification.
- **ClientRequirements**-Overthetime,customermayaskfornewfeatures or functions in thesoftware.
- **Host Modifications** - If any of the hardware and/or platform (such as operatingsystem)ofthetargethostchanges,softwarechangesareneeded to keepadaptability.
- **Organization Changes** - If there is any business level change at client end, such as reduction of organization strength, acquiring another company, organization venturing into new business, need to modify in the original software mayarise.

Types ofmaintenance

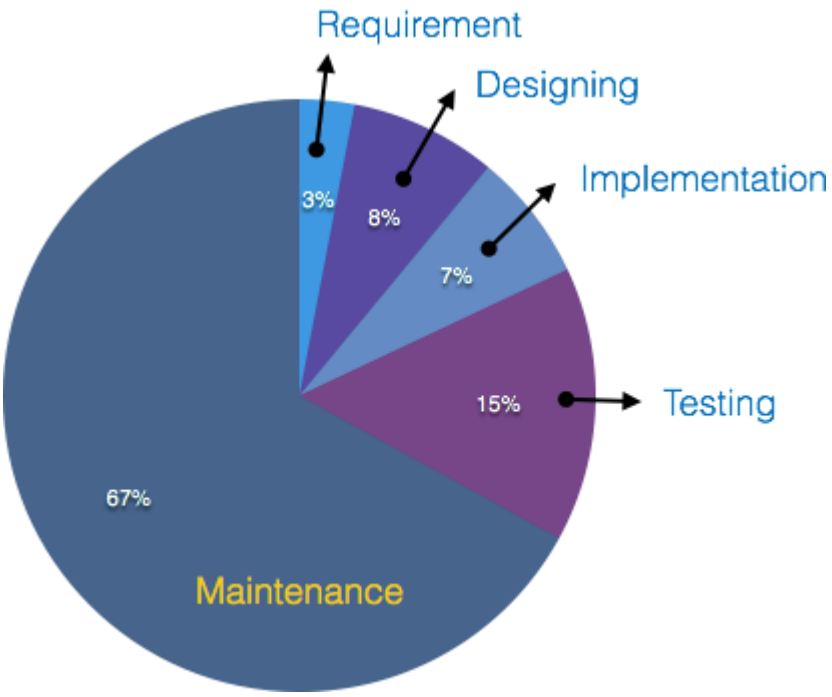
In a software lifetime, type of maintenance may vary based on its nature. It may be just a routine maintenance tasks as some bug discovered by some user or it may be a large event in itself based on maintenance size or nature. Followingare some types of maintenance based on theircharacteristics:

- **CorrectiveMaintenance**-Thisincludesmodificationsandupdationsdone in order to correct or fix problems, which are either discovered by user or concluded by user errorreports.
- **Adaptive Maintenance** - This includes modifications and updations applied to keep the software product up-to date and tuned to the ever changing world of technology and businessenvironment.

- **Perfective Maintenance** - This includes modifications and updates done in order to keep the software usable over long period of time. It includes newfeatures,newuserrequirementsforrefiningthesoftwareandimprove its reliability andperformance.
- **Preventive Maintenance** - This includes modifications and updations to preventfutureproblemsofthesoftware.Itaimstoattendproblems,which are not significant at this moment but may cause serious issues infuture.

Cost ofMaintenance

Reports suggest that the cost of maintenance is high. A study on estimating software maintenance found that the cost of maintenance is as high as 67% of the cost of entire software process cycle.



On an average, the cost of software maintenance is more than 50% of all SDLC phases. There are various factors, which trigger maintenance cost go high, such as:

Real-world factors affecting Maintenance Cost

- The standard age of any software is considered up to 10 to 15years.
- Older softwares, which were meant to work on slow machines with less memory and storage capacity cannot keep themselves challenging against newly coming enhanced softwares on

modernhardware.

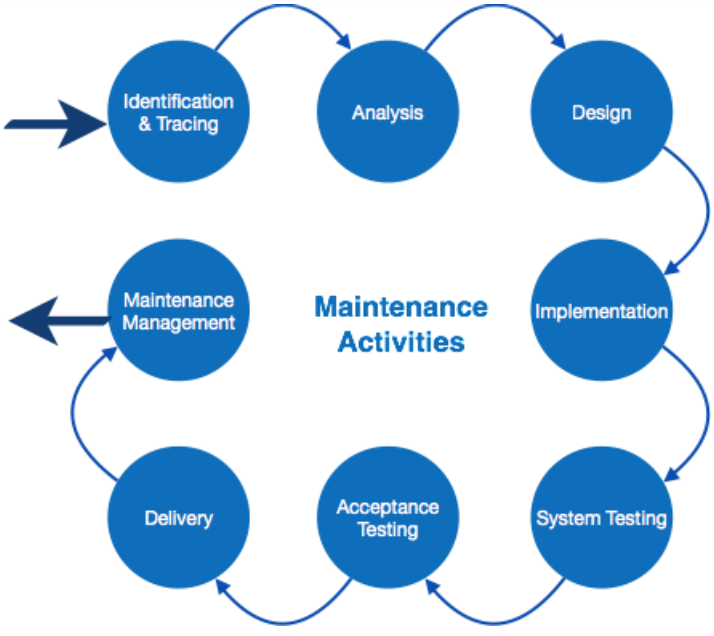
- As technology advances, it becomes costly to maintain oldsoftware.
- Most maintenance engineers are newbie and use trial and error method to rectifyproblem.
- Often, changes made can easily hurt the original structure of the software, making it hard for any subsequentchanges.
- Changes are often left undocumented which may cause more conflicts in future.

Software-end factors affecting Maintenance Cost

- Structure of SoftwareProgram
- ProgrammingLanguage
- Dependence on externalenvironment
- Staff reliability andavailability

Maintenance Activities

IEEE provides a framework for sequential maintenance process activities. It can be used in iterative manner and can be extended so that customized items and processes can be included.



These activities go hand-in-hand with each of the following phase:

- **Identification&Tracing** - Itinvolvesactivitiespertainingtoidentification ofrequirementofmodificationormaintenance.Itisgeneratedbyuseror

system may itself report via logs or error messages.Here, themaintenance type is classifiedalso.

- **Analysis** - The modification is analyzed for its impact on the system including safety and security implications. If probable impact is severe, alternative solution is looked for. A set of required modifications is then materialized into requirement specifications. The cost of modification/maintenance is analyzed and estimation isconcluded.
- **Design** - New modules, which need to be replaced or modified, are designed against requirement specifications set in the previous stage. Test cases are created for validation andverification.
- **Implementation**- Thenewmodulesarecodedwiththehelpofstructured design created in the design step.Every programmer is expected to dounit testing in parallel.
- **System Testing** - Integration testing is done among newly created modules. Integration testing is also carried out between new modules and the system. Finally the system is tested as a whole, following regressive testingprocedures.
- **Acceptance Testing** - After testing the system internally, it is tested for acceptance with the help of users. If at this state, user complaints some issues they are addressed or noted to address in nextiteration.
- **Delivery** - After acceptance test, the system is deployed all over the organization either by small update package or fresh installation of the system. The final testing takes place at client end after the software is delivered.

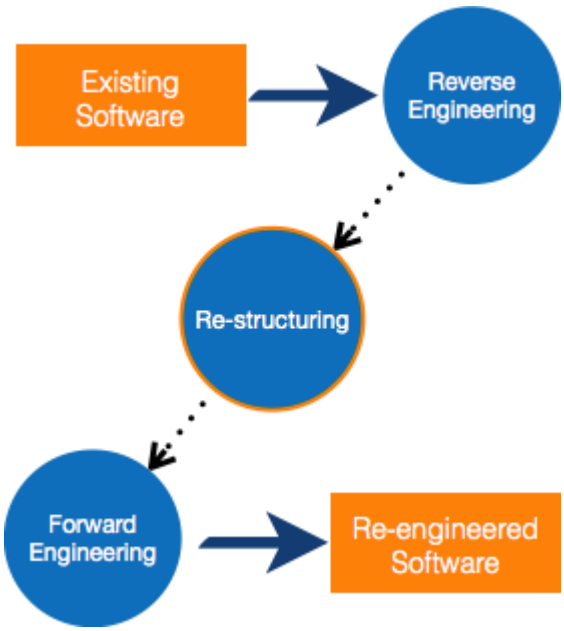
  Training facility is provided if required, in addition to the hard copy of user manual.
- **Maintenance management** - Configuration management is an essential partofsystemmaintenance.Itisaidedwithversioncontroltoolstocontrol versions, semi-version or patchmanagement.

When we need to update the software to keep it to the current market, without impactingitsfunctionality,itiscalled**softwarere-engineering**.Itisathorough process where the design of software is changed and programs arere-written.

Legacy software cannot keep tuning with the latest technology available in the market. As the hardware become obsolete, updating of software becomes a headache. Even if software grows old with time, its functionality does not.

For example, initially Unix was developed in assembly language. When language Ccameintoexistence,Unixwasre-engineeredinC,becauseworkinginassembly language wasdifficult.

Other than this, sometimes programmers notice that few parts of software need more maintenance than others and they also need re-



engineering.

Re-Engineering Process

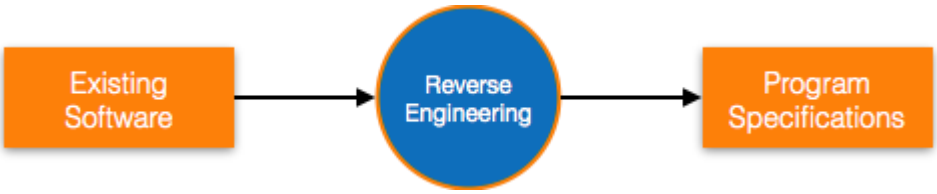
- **Decide** what to re-engineer. Is it whole software or a part ofit?
- **Perform** Reverse Engineering, in order to obtain specifications of existing software.
- **Restructure Program**if required. For example, changing function- orien
- **Re-structure data** asrequired.
- **Apply Forward engineering** concepts in order to get re-engineered software.

There are few important terms used in Software re-engineering

Reverse Engineering

It is a process to achieve system specification by thoroughly analyzing, understanding the existing system. This process can be seen as reverse SDLC model, i.e. we try to get higher abstraction level by analyzing lower abstraction levels.

An existing system is previously implemented design, about which we know nothing. Designers then do reverse engineering by looking at the code and try to getthedesign.Withdesigninhand,theytrytoconcludethespecifications.Thus,



going in reverse from code to systemspecification.

Program Restructuring

Itisaprocesstore-structureandre-constructtheexistingsoftware.Itisallabout re-arranging the source code, either in same programming language or from one programming language to a different one. Restructuring can have either source code-restructuring and data-restructuring orboth.

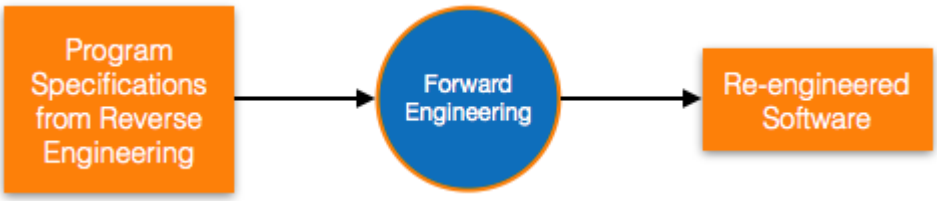
Re-structuring does not impact the functionality of the software but enhance reliability and maintainability. Program components, which cause errors very frequently can be changed, or updated with re-structuring.

The dependability of software on obsolete hardware platform can be removed via re-structuring.

Forward Engineering

Forward engineering is a process of obtaining desired software from the specificationsinhandwhichwerebroughtdownbymeansofreverseengineering . It assumes that there was some software engineering already done in thepast.

Forward engineering is same as software engineering process with only one difference – it is carried out always after reverse engineering.



Component reusability

A component is a part of software program code, which executes an independent task in the system. It can be a small module or sub-system itself.

Example

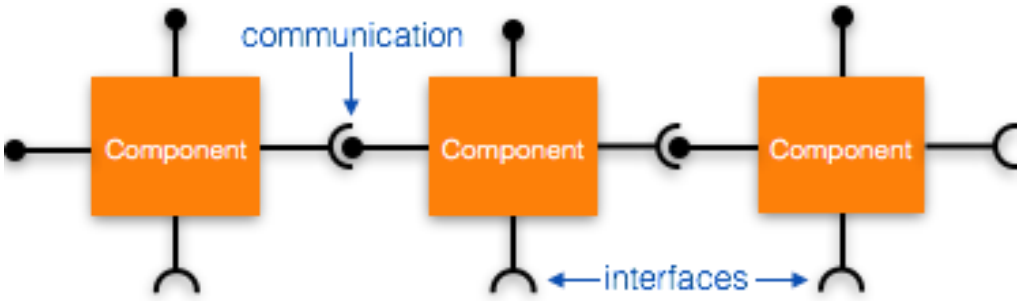
Theloginproceduresusedonthewebcanbeconsideredascomponents,printing system in software can be seen as a component of thesoftware.

Components have high cohesion of functionality and lower rate of coupling, i.e. they work independently and can perform tasks without depending on other modules.

InOOP,theobjectsaredesignedareveryspecifictotheirconcernandhavefewer chances to be used in some othersoftware.

In modular programming, the modules are coded to perform specific tasks which can be used across number of other software programs.

There is a whole new vertical, which is based on re-use of software component, and is known as Component Based Software Engineering (CBSE).



Re-use can be done at various levels

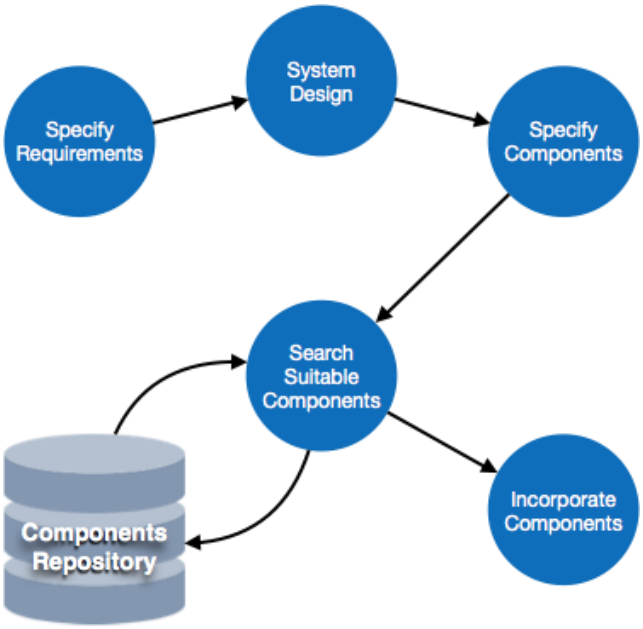
- **Application level** - Where an entire application is used as sub-system of newsoftware.
- **Component level** - Where sub-system of an application isused.
- **Modules level** - Where functional modules arere-used.

Software components provide interfaces, which can be used to establish communication among different components.

Reuse Process

Two kinds of method that can be adopted: either by keeping requirements same and adjusting components or by keeping components

same and modifying requirements.



- **Requirement Specification** - The functional and non-functional requirements are specified, which a software product must comply to,with the help of existing system, user input orboth.
- **Design** - This is also a standard SDLC process step, where requirements are defined in terms of software parlance. Basic architecture of system as a whole and its sub-systems arecreated.
- **Specify Components** - By studying the software design, the designers segregate the entire system into smaller components or sub-systems. One complete software design turns into a collection of a huge set of components workingtogether.
- **Search Suitable Components** - The software component repository is referred by designers to search for the matching component, on the basis of functionality and intended softwarerequirements..
- **IncorporateComponents**- Allmatchedcomponentsarepackedtogether to shape them as completesoftware.

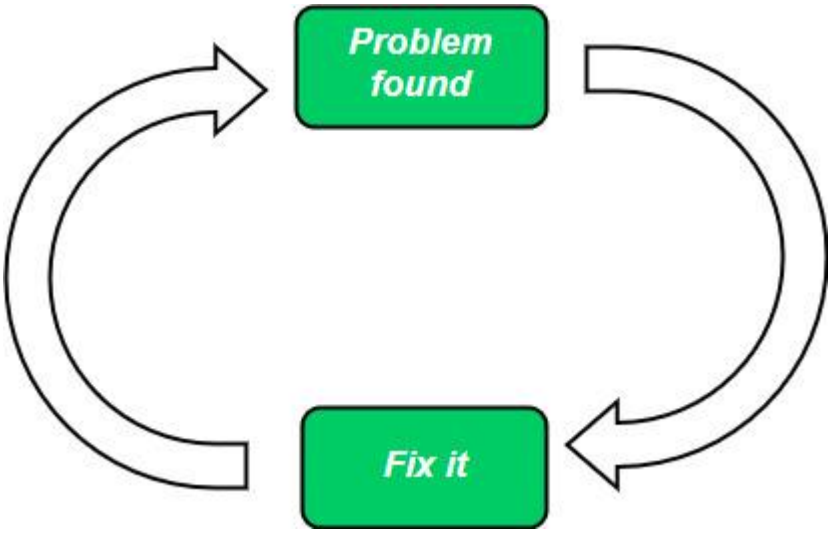
Software maintenance models

Software Engineering | Quick-fix Model

Software Maintenance is a process of modifying a software system after delivery to correct the faults, add new features and to remove obsolete functions. Maintenance process varies considerably depending on the type of the software being maintained. The most expensive part of the software life cycle is a software maintenance process. There are some models for the maintenance of the software system, Qquick-fix model is one of them.

Quick-fix Model :

- It is basically an adhoc approach to maintain software.
- It is a fire fighting approach waiting for the problem to accur and then trying to fix it as quick as possible.
- The main objective of this model is to identify the problem and then fix it as soon as possible.
- In this model, changes are made at code level as early as possible without accepting future problems.
- This model is an approach to modify the software code without little consideration of its impact on the overall structure of the software system.
- As a result of this model, the structure of the software degrade rapidly



Advantages:

1. The main advantage is that it performs its work at low cost and very quickly.
2. Sometimes, users don't wait for the long time. Rather, they require the modified software to be delivered to them in the least possible time. As a result, the software maintenance team needs to use a quick-fix model to avoid the time consuming process of Software maintenance life cycle.
3. This model is also advantageous in situations where the software system is to be maintained with certain deadlines and limited resources.

Disadvantages:

1. This model is not suitable for large project system.
2. This model is not suitable to fix errors for a longer period, as the structure of the software system degrade rapidly.

What is Software Maintenance?

After completing the hectic and time consuming process of developing and testing a software application, taking measures to ensure its maintenance is quite sensible and important. Nowadays, software maintenance is widely accepted as part of **Software Development Life Cycle (SDLC)**. It is the process of modifying and updating software application after its delivery to improve its performance, correct any new defects and adapt the product according to the modified environment. The purpose of **software maintenance** is to preserve the value of software over time, which can accomplished by:

- Expanding the customer base.
- Enhancing software's capabilities.
- Omitting obsolete capabilities.
- Employing newer technology.

Categories of Software Maintenance:

Basic software maintenance includes optimization, error correction, and enhancement of existing features, which combine together to make the software abreast with the latest changes and demands of the software industry. However, the type of maintenance can vary in a software based on its nature and requirement. In order to make the process of maintaining software more profiting and beneficial, Software Maintenance is divided into four main categories:

1. **Corrective Maintenance:** Corrective Maintenance is a reactive process that is focused on fixing failures in the system. It refers to the modification and enhancement done to the coding and design of a software to fix errors or defects detected by the user or concluded by error user report. This type of maintenance is initiated in the system to resolve any new or missed defects in the software. Corrective Maintenance is further divided into two types:

- Emergency Repairs.
- Scheduled Repairs.

2. **Adaptive Maintenance:** Adaptive Maintenance is initiated as a consequence of internal needs, like moving the software to a different hardware or software platform compiler, operating system or new processor and to match the external completion and requirements. The main goal of Adaptive Maintenance is to keep the software program up-to-dated and to meet the needs and demands of the user and the business.

3. **Perfective Maintenances:** Here enhancements, modifications and updates are done in order to keep the software usable for a long period of time. It aims at achieving reduced costs in using the system and increasing its maintainability. The process of perfective maintenance includes making the product faster, cleaner structured, improving its reliability and performance, adding new features, and more.

4. **Preventive Maintenance:** Most commonly known as Software Re-engineering, the purpose of this type of maintenance is to prevent future problems in the software by making it more understandable, enhancing its features and improving its existing qualities, which will facilitate future maintenance work. The objective of Preventive Maintenance is to attend problems, which may seem insignificant but can cause serious issues in the future.

Challenges in Software Maintenance:

Maintaining software is though considered essential these days, it is not a simple procedure and entails extreme efforts. The process requires knowledgeable experts who are well versed in latest software engineering trends and can perform suitable programming and testing. Furthermore, the programmers can face several challenges while executing software maintenance which can make the process time consuming and costly. Some of the challenges encountered while performing software maintenance are:

- Finding the person or developer who constructed the program can be difficult and time consuming.

- Changes are made by an individual who is unable to understand the program clearly.

- The systems are not maintained by the original authors, which can result in confusion and misinterpretation of changes executed in the program.

- Information gap between user and the developer can also become a huge challenge in software maintenance.

- The biggest challenge in software maintenance is when systems are not designed for changes.

Process of Software Maintenance:

Software Maintenance is an important phase of Software Development Life Cycle (SDLC), and it is implemented in the system through a proper software maintenance process, known as **Software Maintenance Life Cycle (SMLC)**. This life cycle consists of seven different phases, each of which can be used in iterative manner and can be extended so that customized items and processes can be included. These seven phases of Software Maintenance process are:

1. **Identification Phase:**

   In this phase, the requests for modifications in the software are identified and analysed. Each of the requested modification is then assessed to determine and classify the type of maintenance activity it requires. This is either generated by the system itself, via logs or error messages, or by the user.

2. **Analysis Phase:**

   The feasibility and scope of each validated modification request are determined and a plan is prepared to incorporate the changes in the software. The input attribute comprises validated modification request, initial estimate of resources, project documentation, and repository information. The cost of modification and maintenance is also estimated.

3. **Design Phase:**

   The new modules that need to be replaced or modified are designed as per the requirements specified in the earlier stages. Test cases are developed for the new design including the safety and security issues. These test cases are created for the **validation and verification** of the system.

4. **Implementation Phase:**

   In the implementation phase, the actual modification in the software code are made, new features that support the specifications of the present software are added, and the modified software is installed. The new modules are coded with the assistance of structured design created in the design phase.

5. **System Testing Phase:**

   Regression testing is performed on the modified system to ensure that no defect, error or bug is left undetected. Furthermore, it validates that no new faults are introduced in the software as a result of maintenance activity. **Integration testing** is also carried out between new modules and the system.

6. **Acceptance Testing Phase:**

   Acceptance testing is performed on the fully integrated system by the user or by the third party specified by the end user. The main objective of this testing is to verify that all the features of the software are according to the requirements stated in the modification request.

7. **Delivery Phase:**

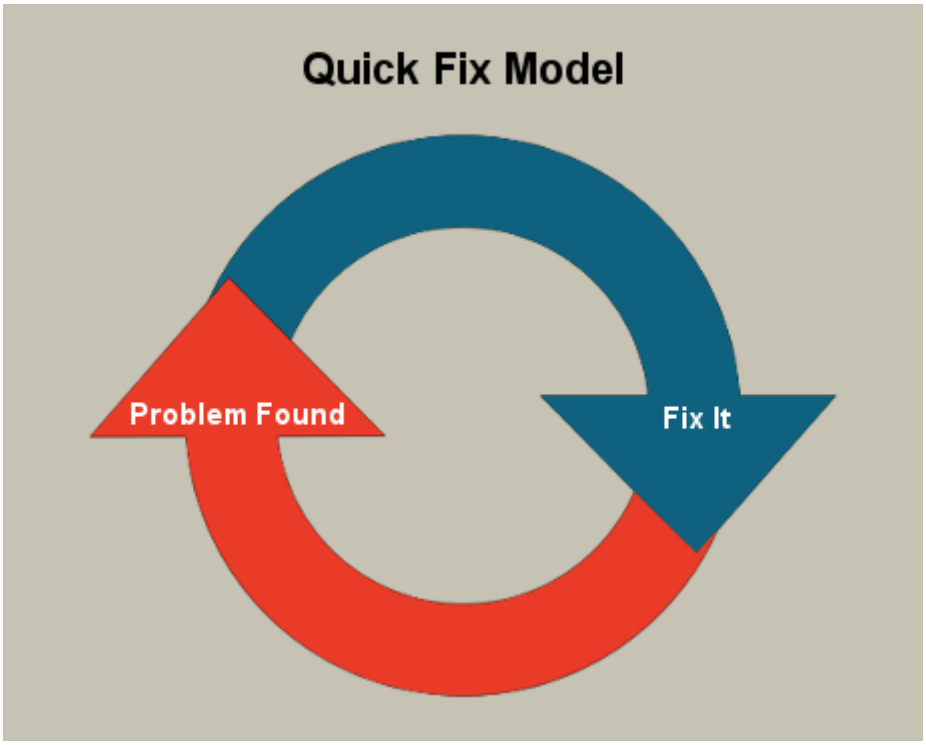
   Once the acceptance testing is successfully accomplished, the modified system is delivered to the users. In addition to this, the user is provided proper consisting of manuals and help files that describe the operation of the software along with its hardware specifications. The final testing of the system is done by the client after the system is delivered.

Software Maintenance Models:

To overcome internal as well as external problems of the software, Software maintenance models are proposed. These models use different approaches and techniques to simplify the process of maintenance as well as to make is cost effective. Software maintenance models that are of most importance are:

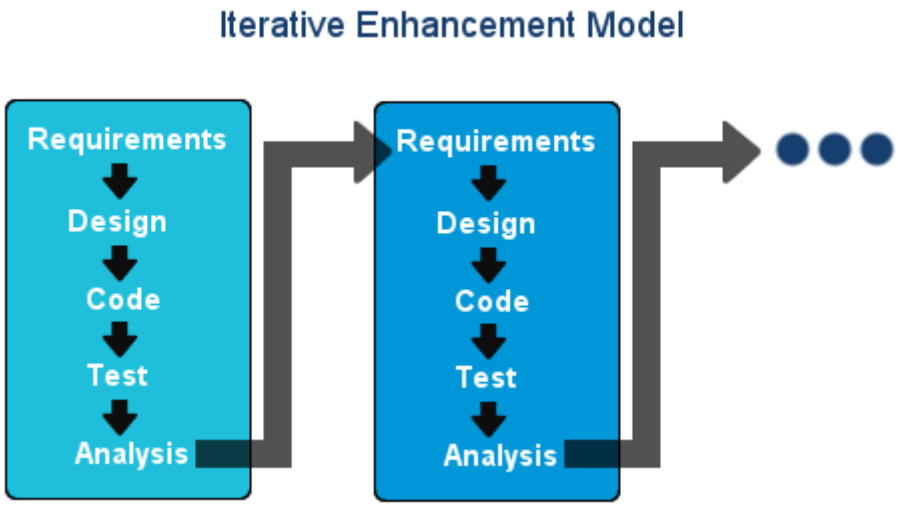
Quick-Fix Model:

This is an ad hoc approach used for maintaining the software system. The objective of this model is to identify the problem and then fix it as quickly as possible. The advantage is that it performs its work quickly and at a low cost. This model is an approach to modify the software code with little consideration for its impact on the overall structure of the software system.



Iterative Enhancement Model:

Iterative enhancement model considers the changes made to the system are iterative in nature. This model incorporates changes in the software based on the analysis of the existing system. It assumes complete documentation of the software is available in the beginning. Moreover, it attempts to control complexity and tries to maintain good design.



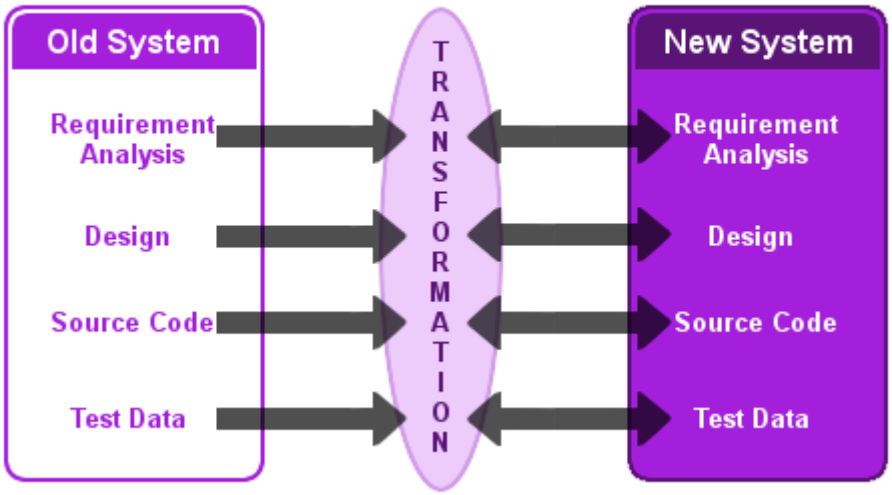
Iterative Enhancement Model is divided into three stages:

1. Analysis of software system.
2. Classification of requested modifications.
3. Implementation of requested modifications.

The Re-use Oriented Model:

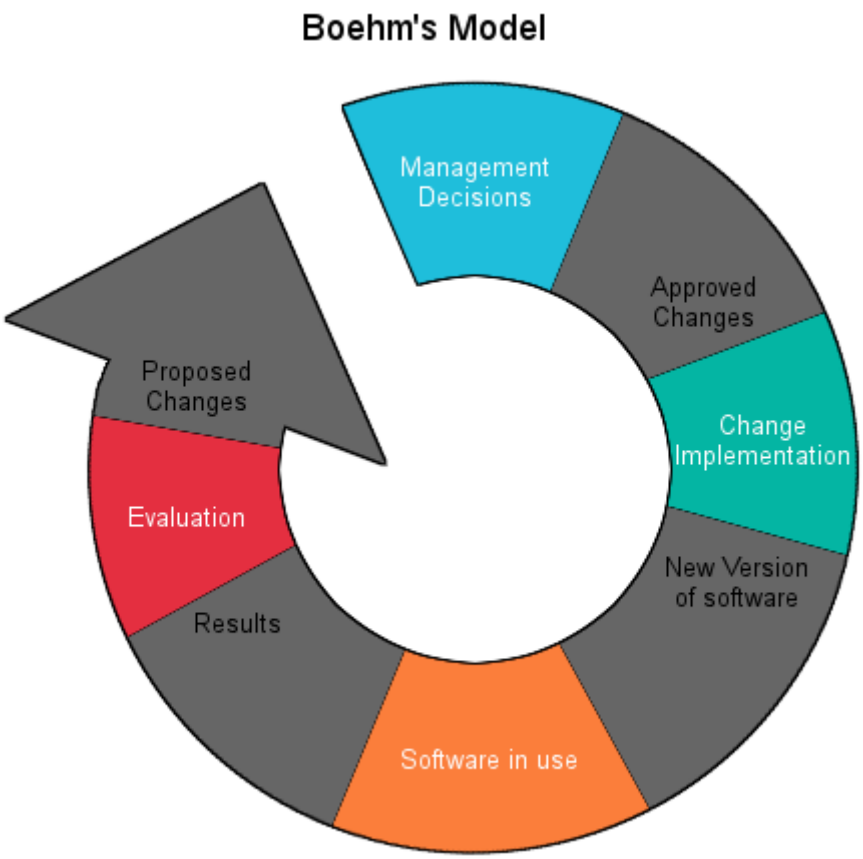
The parts of the old/existing system that are appropriate for reuse are identified and understood, in Reuse Oriented Model. These parts are then go through modification and enhancement, which are done on the basis of the specified new requirements. The final step of this model is the integration of modified parts into the new system.

Reuse Oriented Model



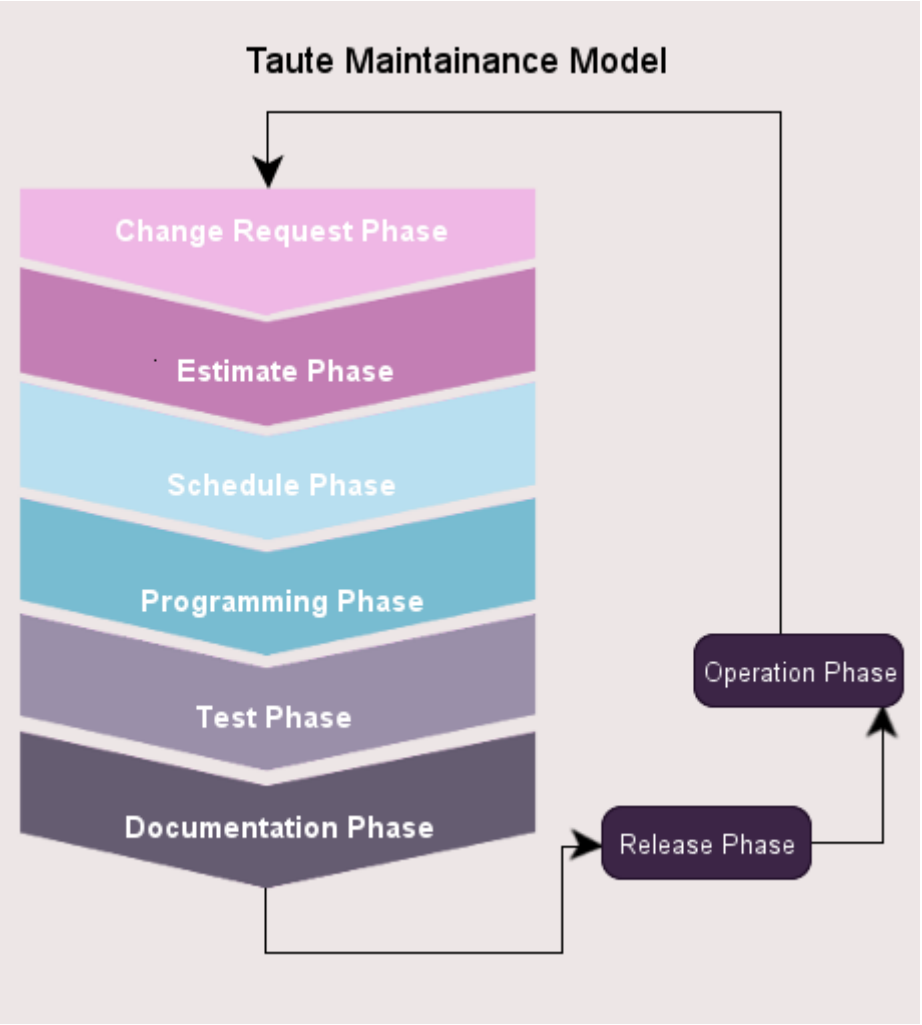
Boehm's Model:

Boehm's Model performs maintenance process based on the economic models and principles. It represents the maintenance process in a closed loop cycle, wherein changes are suggested and approved first and then are executed.



Taute Maintenance Model:

Named after the person who proposed the model, Taute's model is a typical maintenance model that consists of eight phases in cycle fashion. The process of maintenance begins by requesting the change and ends with its operation. The phases of **Taute's Maintenance Model** are:

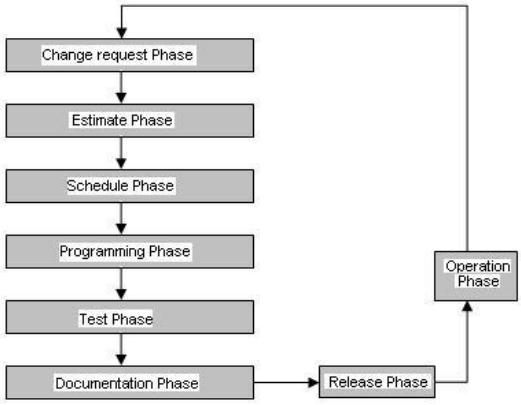


1. **Change request Phase.**
2. **Estimate Phase.**
3. **Schedule Phase.**
4. **Programming Phase.**
5. **Test Phase.**
6. **Documentation Phase.**
7. **Release Phase.**

8. **Operation Phase.**

Taute Software Maintenance Model

The model was developed by B.J. Taute in 1983 and is very easy to understand and implement. It is a typical maintenance model and has eight phases in cycle fashion.



Taute maintenance model

(i) Change requires phase: Maintenance team gets a request in a prescribed format from the client to make a change. This change may fall in any category of maintenance activities. We identify the type of requires (i.e., corrective, adaptive, preventive or preventive) and assign a unique identification number of request.

(ii) Estimate phase: This phase is devoted to estimate the time and effort required to make the change. It is difficult to make exact estimates. but our objective is to have at least reasonable estimate of time and efforts. Impact analysis on existing system is also required to minimize the ripple effect.

(iii) Schedule phase: We may like to identify change request for the next scheduled release and may also prepare the documents that are required for planning.

(iv) Programming phase: In this phase, source code is modified to implement the requested change. All relevant documents like design document, manuals, etc. are updated accordingly. Final output is the test version of the source code.

(v) Test phase: We would like to ensure that modification is correctly implemented.

Hence, we test the code. We may use already available test cases and may also design new test cases. The term used for such testing is known as regression testing.

(vi) Documentation phase: After regression testing, system and user documents are prepared and updated before releasing the system. This helps us to maintain co-relation between code and documents.

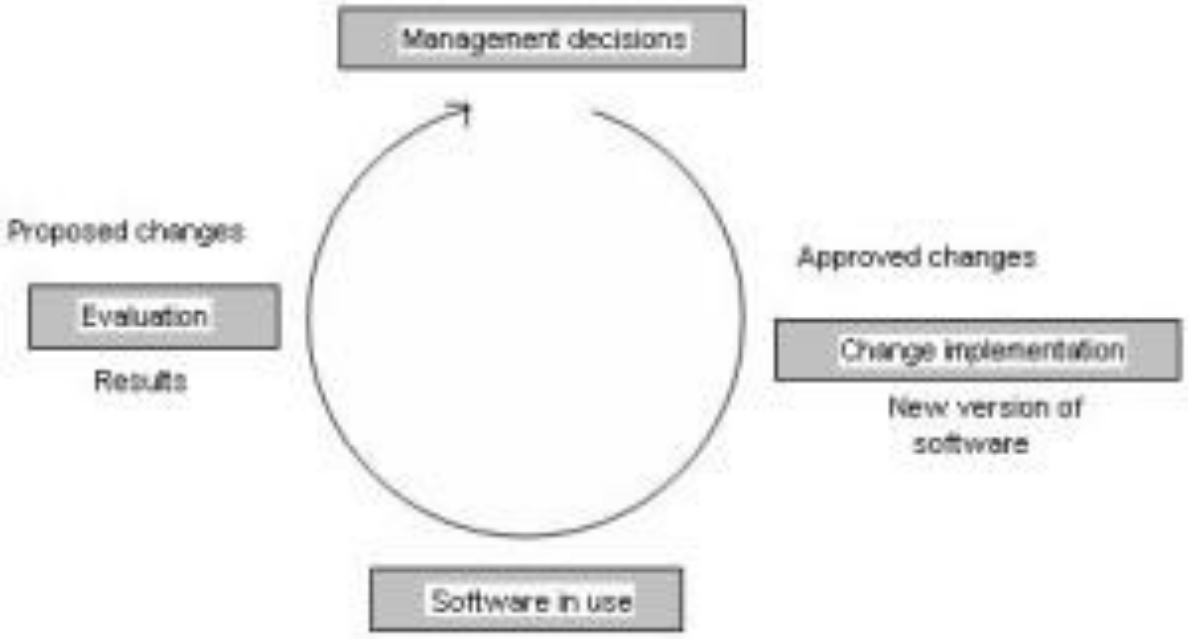
(vii) Release phase: The new software product along with updated documents are delivered to the customer. Acceptance testing is carried out by the users of the system.

(viii) Operation phase: After acceptance testing, software is placed under normal operation. During usage, when another problem is identified or new functionality requirement is felt or even the enhancement of existing capability is desired, again a 'change request' process is initiated. If we do so, we may go back to change request phase and repeat all phases to implement the change.

Boehm's Software Maintenance Model

Source

Boehm represents the maintenance process as a closed loop cycle. He theorizes that it is the stage where management decisions are made that drives the process. In the stage, asset of approved changes is determined by applying particular strategies and cost-benefit evaluations to a set of proposed changes. The approved changes are accompanied by their own budgets, which will largely determine the extent and type of resources expanded.



Boehm sees the maintenance manager's task as one of balancing the pursuit f the objectives of maintenance against the constraints imposed by the environment in which maintenance work is carried out. Thus the maintenance process is driven by the maintenance manager's decisions, which are based on the balancing of objectives against the constraints.

Previous Years University Question Papers:

AMOOR- III (2014 Course) CBCS : WINTER - 2016

Subject : Software Engineering

Day : Thursday
Date : 10/11/2016

32277

Time : 02.00 PM TO 05.00 PM
Max Marks : 100 Total Pages : 1

N.B.

1) Answer any **FOUR** questions from Section – I and any **TWO** questions from Section - II.
2) Figures to the right indicate **FULL** marks.
3) Answers to both the sections should be written in **SAME** answer book.

SECTION – I

Q.1 What is Software Engineering? Explain basic concepts of Software Engineering with importance of Software Engineering in Software Development. **(15)**

Q.2 Explain Software Project Management in brief. What are applications of PERT and GANTT charts? **(15)**

Q.3 What are the stages in Software Development Life Cycle? Explain in brief Feasibility study and its benefits. **(15)**

Q.4 Explain Requirement Engineering concepts with types of Requirements in Software Development Process. **(15)**

Q.5 What are characteristics of (SRS) Software Requirement Specification? Explain in brief why SRS is required? **(15)**

Q.6 Write detail note on Function Oriented Modeling and Object Oriented Modeling with respect to software development. **(15)**

Q.7 Write short notes on the following: **(15)**

a) Testing Techniques
b) Quality Concepts
c) Software Maintenance

SECTION – II

Q.8 Explain Formal Technical Reviews for Software Quality Assurance Plan for any business application Software. Assume appropriate real business documents in Review Meeting and Review guidelines for business Software development. **(20)**

Q.9 Write detail note on Maintenance Process and Models. Explain in brief Reuse Oriented Model. **(20)**

Q.10 Draw the Entity Relationship Diagram and Context Level Data Flow Diagram for your College Library Management System. (Assume appropriate processes in the Library Management System) **(20)**

* * *

1

AMOOR- III (2014 Course) : WINTER - 2015

Subject : Data Structures

Day : Thursday
Date : 26/11/2015

26317

Time : 02.00 PM TO 05.00 PM
Max Marks : 100 Total Pages : 1

N.B.:

1) Attempt any **FOUR** questions from Section -I.
2) Attempt any **TWO** questions from Section – II.
3) Figures to the right indicate **FULL** marks.
4) Answers to both the sections should be written in **SAME** answer book.

SECTION-I

Q.1 Explain the applications of stack with example. **(15)**

Q.2 Explain in detail any three sorting techniques. **(15)**

Q.3 Explain Advantages and disadvantages of Linked list. **(15)**

Q.4 What is Data structure? Explain types of data structures. **(15)**

Q.5 What is Queues? Explain types of queues. **(15)**

Q.6 Write a program to allocate memory dynamically for string and store their addresses in array of pointers to string. **(15)**

Q.7 Explain the terms: **(15)**

i) Inorder Traversal
ii) Preorder Traversal
iii) Postorder Traversal

SECTION-II

Q.8 Write a program to implement depth first search algorithm. **(20)**

Q.9 Write a program to sort 20, 35, 40, 100, 3, 10, 15 using insertion sort. **(20)**

Q.10 Write program to find specific element from the array using binary search. **(20)**

AMOOR- III (2014 Course) CBCS : SUMMER - 2016

Subject : Software Engineering

Day : Saturday
Date : 16/04/2016

29232

Time : 02.00 PM TO 05.00 PM
Max Marks : 100 Total Pages : 1

N. B. :

1) Attempt **ANY FOUR** questions from Section –**I**. Each question carries **15**marks.
2) Attempt **ANY TWO** questions from Section –**II**. Each question carries **20** marks.
3) Answers to both the sections should be written in the **SAME** answer book.

SECTION - I

Q. 1 What are software engineering concepts? Explain principles and importance of software engineering. **(15)**

Q. 2 What is software project management? Explain software configuration in brief. **(15)**

Q. 3 Explain software development life cycle in brief. What are stages in SDLC? **(15)**

Q. 4 What is need of feasibility study? Explain in brief types of feasibility. **(15)**

Q. 5 What is requirement engineering? Explain in brief types of requirements. **(15)**

Q. 6 What are characteristics of SRS? Explain function oriented modeling in brief. **(15)**

Q. 7 Explain ERD concepts with example. What are benefits of flow charts? **(15)**

SECTION – II

Q. 8 What are concepts of Testing? Explain different testing techniques with example. **(20)**

Q. 9 What are quality concepts with respect to software? Explain in brief software quality assurance and software reviews. **(20)**

Q.10 Write short notes on **ANY TWO** of the following: **(20)**

a) Software maintenance
b) Object oriented design
c) Waterfall model

* * * * *

AMOOR- III (2014 Course) CBCS : SUMMER - 2017

Subject : Software Engineering

Day : Thursday
Date : 13/04/2017

35632

Time : 02.00 PM TO 05.00 PM
Max Marks : 100 Total Pages : 1

N. B. :

1) Attempt **ANY FOUR** questions from Section –**I**. Each question carries **15**marks.
2) Attempt **ANY TWO** questions from Section –**II**. Each question carries **20** marks.
3) Answers to both the sections should be written in the **SAME** answer book.

SECTION - I

Q. 1 Explain different software development models in brief. What are principles of software engineering? **(15)**

Q. 2 What is software project management? Explain application of PERT and GANTT charts. **(15)**

Q. 3 Explain cost benefit analysis process in short. What are types of feasibility study? **(15)**

Q. 4 What are the types of requirements? Explain traditional methods and modern methods in brief. **(15)**

Q. 5 What are the concepts of object oriented modeling? Explain UML and class diagrams in short. **(15)**

Q. 6 What are testing techniques? Explain in short test plans and test cases. **(15)**

Q. 7 Explain in detail software quality assurance measures. **(15)**

SECTION – II

Q. 8 What are maintenance process and models? Explain need of maintenance in brief. **(20)**

Q. 9 What are quality concepts? Explain formal technical reviews in brief. **(20)**

Q.10 What are characteristics of SRS? Explain in brief IEEE-SRS. **(20)**

* * * * *

B.C.A. SEM–III (2014 COURSE) CBCS : SUMMER - 2018

SUBJECT : SOFTWARE ENGINEERING

Day : Monday
Date : 30/04/2018

S-2018-1703

Time : 02.00 PM TO 05.00 PM
Max. Marks: 100

N.B.:

1) Attempt **ANY FOUR** questions from Section – I and **ANY TWO** questions from section – II.
2) Answers to both the sections should be written in the **SEPARATE** answer books.
3) Figures to the right indicate **FULL** marks.

SECTION – I

Q.1 Explain software engineering principles and need in detail. **[15]**

Q.2 What are the various stages in SDLC? Explain in short Software Development Life Cycle Requirement phase. **[15]**

Q.3 Explain software project management and software security measures in brief. **[15]**

Q.4 What are the software quality measures? How we can develop quality software explain in brief? **[15]**

Q.5 What are E-R diagram concepts? Explain with example E-R diagrams for any business software application. **[15]**

Q.6 Explain in detail various software development models in brief. Write stages in waterfall model. **[15]**

Q.7 What is the role of team leader in software development? Explain functions of programmer in software development in brief. **[15]**

SECTION – II

Q.8 Explain in detail various testing methods with advantages and disadvantages for the same. **[20]**

Q.9 Draw data flow diagrams of various levels and E-R diagram for college management system to be developed. **[20]**

Q.10 Write short notes on **ANY TWO** of the following: **[20]**

a) Object oriented design concepts
b) Software testing tools
c) Role of software engineer

B.C.A. SEM–III (2014 Course) CBCS : SUMMER - 2019

SUBJECT: SOFTWARE ENGINEERING

Day : **Saturday** Time : 02.00 PM TO 05.00 PM

Date : **20/04/2019** **S-2019-2068** Max. Marks :100

N.B.:

1) Attempt any **FOUR** questions from Section –I and any **TWO** questions from Section- II.
2) Figures to the right indicate **FULL** marks.
3) Answers to both the sections should be written in SAME answer book.

SECTION – I

Q.1 What is software maintenance? Explain corrective, adaptive and preventive maintenance. **(15)**

Q.2 What is software process model? Explain with neat diagram Spiral model and advantages of it. **(15)**

Q.3 Explain the concept of software quality, quality control and quality assurance in brief. **(15)**

Q.4 Explain various project management activities in brief. **(15)**

Q.5 What is software design process? Explain in detail. **(15)**

Q.6 Explain cost benefit analysis with example. **(15)**

Q.7 Write short notes on any **THREE** of the following: **(15)**

a) Taute's Model.
b) PERT.
c) Feasibility study.
d) Black Box testing.

SECTION – II

Q.8 Draw DFD and ERD for Hostel Management System by considering your own assumptions. **(20)**

Q.9 ABC Transport company has decided to develop software for their company. You are called for studying the operations of the firm. Suppose you have done the complete study of the firm with its functions. **(20)**

Now suggest an appropriate software development model for the same. Also justify the advantages and disadvantages of the software model selected by you.

Q.10 Draw following diagram for Hostel Management System:

a) Use Case Diagram. **(10)**
b) Class Diagram. **(10)**

* * * *

Previous Years Internal Question Papers:

1st Internal Examination (2019)

Course: BCA | Semester: III

Subject: Software Engineering | Course Code: 302

Max. Marks: 40 | Max. Time: 2 Hours

Instructions (if any):- Use of calculator for subjects like Financial Mgt. Operation etc. allowed if required. (Scientific calculator is not allowed).

Use of unfair means will lead to cancellation of paper followed by disciplinary action.

Question No. 1 is compulsory. Attempt any two questions from Q2 to Q5.

Attempt any two question from section 2.

Section 1

Answer in 400 words. Each question carry 06 marks.

Q. 1 Explain Win-Win Spiral model? What are the advantages of this model?

Q. 2 What is Software configuration management? Explain the process.

Q. 3 What is software engineering? What are the principles of software engineering?

Q.4 What is feasibility study? Why it is required? Explain different types of feasibility study.

Q.5 Write Short Note on any two. Answer in 300 words. Each carry 03 marks.

a) SQA.

b) Brainstorming

c) Software engineering vs programming

Section 2

Answer in 800 words. Attempt any 2 questions. Each question carry 11 marks

Q6. What is SRS document? What is the structure of SRS document? Explain characteristics of good SRS document.

Q7. Explain RAD and spiral models with the help of diagram. Also differentiate between both.

Q8. What is requirements engineering? Explain different requirements elicitation techniques.

BharatiVidyapeeth Deemed University,
Institute of Management and Research (BVIMR), New Delhi
1st Internal Examination (August, 2016)

Course: BCA **Semester: III**

Subject: Software Engineering **Course Code: 302**

Max. Marks: 40 **Max. Time: 2 Hours**

Instructions: 1. Give examples wherever required.
2. Make use of diagrams wherever required.

Q. 1 Attempt any five questions. Answer in 50 words (Recall) [5 x 2]

a) What is the difference between software engineering and software programming?

b) Differentiate between program and software.

c What do you understand by Audit?

d) Name any two requirements gathering methods.

e) Define FTR.

f) What do you understand by quality control?

g) What are various estimates developed by software development team?

h) What do you understand by SQA?

Q. 2 attempt any two question. Answer in 200 words (Theoretical Concept) [2 x 5]

a) Define 4 P's of software engineering.

b) What are the pros and cons of prototyping model?

c) Explain Incremental model with the help of diagram.

Q.3 Attempt any two questions. Answer in 200 words (Practical/Application oriented) [2 x 5]

a) What is the role of quality assurance team?

b) What do you understand by ISO Standards?

b) What do you understand by real time and system software?

Q.4 Attempt any one. Answer in 600 words (Analytical Question / Case Study / Essay Type Question to test analytical and Comprehensive Skills) [10 x 1]

a) Differentiate between spiral model and RAD model with the help of diagram and Pros and cons of both the models also.

b) What do you understand by quality control and quality assurance? Explain in detail the different types of cost of quality.

1st Internal Examination (2019)

Course: BCA Semester: III
Subject: Software Engineering Course Code: 302
Max. Marks: 40 Max. Time: 2 Hours

Instructions (if any):- Use of calculator for subjects like Financial Mgt. Operation etc. allowed if required. (Scientific calculator is not allowed).
Use of unfair means will lead to cancellation of paper followed by disciplinary action.
Question No. 1 is compulsory. Attempt any two questions from Q2 to Q5.
Attempt any two question from section 2.

Section 1

Answer in 400 words. Each question carry 06 marks.

Q. 1 Explain various software maintenance models
Q. 2 What do you understand by Structure chart and data dictionary? Explain with the help of diagram.
Q.3 Explain system testing in detail.
Q. 4 Design Entity Relationship Diagram of hospital management system.
Q.5 Write Short Note on any two. Answer in 300 words. Each carry 03 marks.
a) Formal Technical Reviews
b) Types of Maintenance
c) White box testing

Section 2

Answer in 800 words. Attempt any 2 questions. Each question carry 11 marks

Q6. What do you understand by SQA and SCM? Also explain Change control in detail.
Q7. Write and explain the organization of SRS document. Also explain the characteristics of good SRS document
Q8. Design a 0 level, 1 level and 2 level DFD of college management system

BharatiVidyapeeth Deemed University,
Institute of Management and Research (BVIMR), New Delhi
Ist Internal Examination (February 2015)

Subject: **Software Engineering** Course Code:

Max. Marks: 50 Max. Time: 2 Hours

Instructions: 1. Diagrams & Examples should be given wherever possible
2. Use only blue and black pens.

Q. 1 Attempt any five questions. Answer in 50 words **(Recall)** [5 x 2]

a) What do you understand by software engeering?
b) what is the difference between s/w engineering and S/w Programming?
c) Why software projects fail?
d) What do you understand by SQA?
e) What do you understand by SCM?
f) What do ou understand by baseline?
g) What do you understand by SCI?

Q. 2 Attempt any two questions. Answer in 200 words **(Theoretical Concept)** [2 x 5]

a) Explain SCM process in brief?
b) Define the concept of cost of quality?
c) What is quality, quality control and quality assurance?

Q.3 Attempt any two questions. Answer in 200 words **(Practical/Application Oriented)** [2 x 5]

a) What is the role of quality assurance team?
b) What are the characteristics of software?
c) What do you understand by real time and system software?

Q.4 Attempt any one. Answer in 600 words **(Analytical Question / Case Study / Essay type question to test analytical and Comprehensive Skill)** [20 x 1]

a) Define any two software process models with the help of diagram?

b) What is the difference between spiral model and win-win spiral model?

BharatiVidyapeeth(Deemed To be University), Institute of Management and Research (BVIMR), New Delhi
1st Internal Examination (September-2018)

Course : BCA | Semester : III
Subject : Software Engineering | Course Code: 302
Max. Marks: 40 | Max. Time: 2 Hours

Instructions (if any) :- (accounting, mathematics regarding use of Calculator, if required)

Q. 1 Attempt any five questions. Answer in 50 words [5 x 2]

a) What are the functions of prototype?
b) What is a software? How is it different from a program?
c) What are the steps of requirement engineering?
d) What is a software development model?
e) What are the functions of the facilitator in requirement elicitation?
f) What do you understand by Audit and FTR?
g) Define SCM process in brief.
h) What do you understand by SCI?

Q. 2 Attempt any two questions. Answer in 200 words (**Theoretical Concept**) [2 x5]

a)Define the term "Software engineering. Explain its Principals.

b) What is a RAD model? Why it is used?

c) Define the term metrics. What are the different types of metrics used in software engineering?

Q.3 Attempt any two questions. Answer in 200 words (**Application oriented**) [2 x 5]

a) What isSoftware project management? Explain its steps and SPMP document.
b) What are software requirements? What are the different types of requirements?
c) What is Waterfall model? Gives it advantages and disadvantages.
d) What is requirement elicitation? Explain with the help FAST technique.

Q.4 Attempt any one. Answer in 600 words (**Analytical Question / Case Study / Essay Type Question to test analytical and Comprehensive Skills**)

[10 x 1]

a) What is a software development model? Differentiate between incremental and spiral model.
b) Write a short note on (any 2):-

1. GANTT and PERT chart
2. Cost Benefit analysis
3. Component based development model
4. Cost of Quality

1st Internal Examination (2019)

Course: BCA Semester: 3rd
Subject: Software Engineering Course Code: 302
Max. Marks: 40 Max. Time: 2 Hours

Instructions (if any):- Use of Calculators for subjects like Financial Mgt. Operation ,etc allowed if required(Scientific Calculator is not allowed)
Use of unfair means will lead to cancellation of paper followed by disciplinary action.
Question No. 1 is compulsory. Attempt any two questions from Q2 to Q5.
Attempt any two question from section 2.

Section 1

(Theoretical Concept and Practical/Application oriented)

Answer in 400 words. Each question carry 06 marks.

Q1. What is the need of Feasibility Study . What are the types of Feasibility study.
Q2. What is Software Engineering? State the importance and Principles of Software Engineering.
Q3. Explain the cost benefit analysis in reference to the Software Engineering
Q4. What is requirement Engineering. Explain different requirement elicitation techniques.
Q5. Write Short Note on any two. Answer in 300 words. Each carry 03 marks.
a) Software Engineering vs Software Programming
b) Brainstorming sessions vs interviews
c) Waterfall model vs prototype model

Section 2

(Analytical Question/ case study/Essay Type Question to test analytical and Comprehensive Skills)

Answer in 800 words. Attempt any 2 questions.Each question carry 11 marks

Q6.State and explain any five members involved in the software engineering process
Q7.What do you understand by SDLC models. State and explain all the SDLC models with suitable diagrams.
Q8. What do you understand by the term SRS. Explain the characteristics of SRS in detail referring suitable examples.

BharatiVidyapeeth (Deemed to be University)
Institute of Management and Research (BVIMR), New Delhi
2nd Internal Examination (October 2018)

Course : BCA **Semester : III**
Subject : Software Engineering **Course Code: 302**
Max. Marks: 40 **Max. Time: 2 Hours**

Instructions (if any):- (accounting, mathematics regarding use of Calculator, if required). Give Examples & Diagrammatic Representations wherever as possible

Question No. 1 is compulsory. Attempt any two questions from Q2 to Q5.

Attempt any two question from section 2.

Each Question in Section 1 carries 6 marks & Each Question in Section 2 carries 11 marks

Section 1

Answer in 400 words. Each question carry 06 marks.

Q. 1. What is Software maintenance? Explain the maintenance process.
Q. 2. What is software design? What are the different types of cohesion and coupling?
Q.3. Define SRS. What are the characteristics of IEEE SRS.
Q. 4. Explain the concepts of software quality assurance and cost of quality.
Q.5. Write Short Note on any two. Answer in 300 words. Each carry 03 marks.

a) Debugging and acceptance testing
b) UML and Use case diagrams.
c) DFD and ERD

Section 2

Answer in 800 words. Attempt any 2 questions. Each question carry 11 marks

Q6. What is software testing? What are the different types of testing methods?
Q7. What is software quality and its attributes? Explain Boehm's and Taute's model for S/w maintenance?
Q8. Design DFD and ERD for Delhi Metro system.

2nd Internal Examination (2019)

Course: BCA Semester: III
Subject: Software Engineering Course Code: 302
Max. Marks: 40 Max. Time: 2 Hours

Instructions (if any):- Use of unfair means will lead to cancellation of paper followed by disciplinary action.
Question No. 1 is compulsory. Attempt any two questions from Q2 to Q5.
Attempt any two question from section 2.

Section 1

Answer in 400 words. Each question carry 06 marks.

Q. 1 What do you understand by Quality control, quality assurance and cost of quality?
Q. 2 What do you understand by decision tree and decision table? Explain with the help of diagram.
Q.3 Explain white box testing in detail.
Q. 4 What do you understand by baseline, SCI and Change control
Q.5 Write Short Note on any two. Answer in 300 words. Each carry 03 marks.
a) Formal Technical Reviews
b) Types of Maintenance
c) System testing

Section 2

Answer in 800 words. Attempt any 2 questions. Each question carry 11 marks

Q6. Write and explain the organization of SRS document in detail.
Q7. What do you understand by software maintenance and maintenance process? Explain various software maintenance models
Q8. Design a 0 level, 1 level and 2 level DFD of Hospital management system

2nd Internal Examination (March 2020)

Backlog 2018 Course

Course: BCA Semester: III
Subject: Software Engineering Course Code: 302
Max. Marks: 40 Max. Time: 2 Hours

Instructions (if any):- Use of calculator for subjects like Financial Mgt. Operation etc. allowed if required. (Scientific calculator is not allowed).
Use of unfair means will lead to cancellation of paper followed by disciplinary action.
Question No. 1 is compulsory. Attempt any two questions from Q2 to Q5.
Attempt any two question from section 2.

Section 1

Answer in 400 words. Each question carry 06 marks.

Q. 1 Explain various software maintenance models
Q. 2 What do you understand by Structure chart and data dictionary? Explain with the help of diagram.
Q.3 Explain system testing in detail.
Q. 4 Design Entity Relationship Diagram of hospital management system.
Q.5 Write Short Note on any two. Answer in 300 words. Each carry 03 marks.
a) Formal Technical Reviews
b) Types of Maintenance
c) White box testing

Section 2

Answer in 800 words. Attempt any 2 questions. Each question carry 11 marks

Q6. What do you understand by SQA and SCM? Also explain Change control in detail.
Q7. Write and explain the organization of SRS document. Also explain the characteristics of good SRS document
Q8. Design a 0 level, 1 level and 2 level DFD of college management system

2nd Internal Examination (March 2020)

Backlog 2014 Course

Course: BCA | Semester: III
Subject: Software Engineering | Course Code: 302
Max. Marks: 40 | Max. Time: 2 Hours

Instructions (if any):- Use of calculator for subjects like Financial Mgt. Operation etc. allowed if required. (Scientific calculator is not allowed).
Use of unfair means will lead to cancellation of paper followed by disciplinary action.
Question No. 1 is compulsory. Attempt any two questions from Q2 to Q5.
Attempt any two question from section 2.

Section 1

Answer in 400 words. Each question carry 06 marks.

Q. 1 Explain any two software maintenance models.
Q. 2 What do you understand by decision tree and data dictionary? Explain with the help of diagram.
Q.3 Explain integration testing in detail.
Q. 4 Design 0 level and 1 level DFD of hospital management system.
Q.5 Write Short Note on any two. Answer in 300 words. Each carry 03 marks.
a) Cost of quality
b) Types of Maintenance
c) Black box testing

Section 2

Answer in 800 words. Attempt any 2 questions. Each question carry 11 marks

Q6. What do you understand by testing? Explain various techniques of white box and black box testing.
Q7. Write and explain the organization of SRS document. Also explain the characteristics of good SRS document
Q8. Design a ERD of college management system.

BharatiVidyapeethDeemed University,
Institute of Management and Research (BVIMR), New Delhi
2nd Internal Examination (October, 2016)

Course: BCA Semester: III

Subject: Software EngineeringCourse Code: 302

Max. Marks: 40 Max. Time: 2 Hours

Q. 1 Attempt any five questions. Answer in 50 words [5 x 2]

a) What do you understand by SCI?

b) What are characteristics of good SRS document?

c) What are various problems in maintenance?

d) Define various types of relationships in ERD with help of example.

e) What are various sources no change in software during SDLC?

f) What is data dictionary?

g) Explain activity network diagram with the help of example.

h) What is the role of CCA?

Q. 2 attempt any two question. Answer in 200 words [2 x 5]

a) What do you understand by SRS? Explain various components of SRS.

b) What do you understand by maintenance? Explain different types of maintenance?

c) What do you understand by GANTT charts? Explain with the help of diagram.

Q.3 Attempt any two questions. Answer in 200 words [2 x 5]

a) Define coupling and cohesion in detail? Also explain various types of coupling and cohesion.

b) Explain various requirements elicitation techniques?

a) Explain Taute's model of maintenance?

Q.4 Attempt any one. Answer in 600 words [10 x 1]

a) Design a context level and first level DFD for hospital system.

b) Design a ERD for college management system.

Checklist for Coursepacks

- ☐ Title page should be standardized bearing title of subject, course, course code, semester, year of batch (see sample attached)
  - o Name of the instructors teaching the course
  - o Name of course leader
- ☐ Forwarding by HOD bearing his/her signature for approval by Director
- ☐ Logo of BVIMR, name of the institution, address
- ☐ Warning "strictly for internal use" must be printed on the front title page.
- ☐ Table of content bearing
  - o Serial no.
  - o Contents
  - o Page no.
- ☐ Copy of latest syllabus of course as specified by university
- ☐ Lesson plan bearing
  - o Introduction to course
  - o Course objectives
  - o Learning outcomes
- ☐ List of topics/ modules with content
- ☐ Evaluation criteria
  - o CES evaluation description
  - o Recommended text books & reference books
  - o Internet resources
  - o Swayan courses
- ☐ Session plan bearing
  - o Session number
  - o Topic
  - o Readings/case required
  - o Pedagogy followed
  - o Learning outcome
- ☐ Contact details of instructors along with profile
- ☐ Main body of course pack having reading material, exercises, case studies, pages for notes
- ☐ University question papers (preferably last five years including latest university paper)
- ☐ Internal question papers (internal-I-05 papers), (Internal-II-05 papers with latest last year papers)

Note: Include question paper of same subject of old syllabus if required to cover up five years papers.

Declaration by Faculty

I **Daljeet Singh Bawa, Designation Assistant Professor**Teaching **Software Engineering** subject in **BCA- III Morning and Afternoon**course,have incorporated all the necessary pages section/quotations papers mentioned in the check list above.

Daljeet Singh Bawa