

SOFTWARE ENGINEERING M.Sc(CA)-204

SELF LEARNING MATERIAL



**DIRECTORATE
OF DISTANCE EDUCATION**

SWAMI VIVEKANAND SUBHARTI UNIVERSITY

MEERUT – 250 005,

UTTAR PRADESH (INDIA)

SLM Module Developed By :

Author:

Reviewed by :

Assessed by:

Study Material Assessment Committee, as per the SVSU ordinance No. VI (2)

Copyright © **Gayatri Sales**

DISCLAIMER

No part of this publication which is material protected by this copyright notice may be reproduced or transmitted or utilized or stored in any form or by any means now known or hereinafter invented, electronic, digital or mechanical, including photocopying, scanning, recording or by any information storage or retrieval system, without prior permission from the publisher.

Information contained in this book has been published by Directorate of Distance Education and has been obtained by its authors from sources be lived to be reliable and are correct to the best of their knowledge. However, the publisher and its author shall in no event be liable for any errors, omissions or damages arising out of use of this information and specially disclaim and implied warranties or merchantability or fitness for any particular use.

Published by: Gayatri Sales

Typeset at: Micron Computers

Printed at: Gayatri Sales, Meerut.

SOFTWARE ENGINEERING

M.Sc(CA)-204

Unit - I

Software Engineering : Definition and paradigms, A generic view of software engineering.

Unit - II

Requirements Analysis : Statement of system scope, isolation of top level processes and entities and their allocation to physical elements, refinement and review. Analyzing a problem, creating a software specification document, review for correctness, consistency, and completeness.

Unit - III

Designing Software Solutions : Refining the software Specification; Application of fundamental design concept for data, architectural and procedural designs using software blue print methodology and object oriented design paradigm; creating design document : .

Unit - IV

Software Implementation: Relationship between design and implementation: Implementation issues and programming support environment; Coding the procedural design, Good coding style & review of correctness and readability.

Unit - V

Software Maintenance: Maintenance as part of software evaluation, reasons for maintenance, types of maintenance (Perceptive, adoptive, corrective), designing for maintainability, techniques for maintenance. Comprehensive examples using available software platforms/case tools, Configuration Management.

UNIT – I

Software Engineering

The product that software professionals build and then support over the long term.

Software encompasses: (1) instructions (computer programs) that when executed provide desired features, function, and performance; (2) data structures that enable the programs to adequately store and manipulate information and (3) documentation that describes the operation and use of the programs.

Software products

• Generic products

- Stand-alone systems that are marketed and sold to any customer who wishes to buy them.
- Examples – PC software such as editing, graphics programs, project management tools; CAD software; software for specific markets such as appointments systems for dentists.

• Customized products

- Software that is commissioned by a specific customer to meet their own needs.
- Examples – embedded control systems, air traffic control software, traffic monitoring systems.

Why Software is Important?

- The economies of ALL developed nations are dependent on software.
- More and more systems are software controlled (transportation, medical, telecommunications, military, industrial, entertainment,)
- Software engineering is concerned with theories, methods and tools for professional software development.
- Expenditure on software represents a significant fraction of GNP in all developed countries.

Software costs

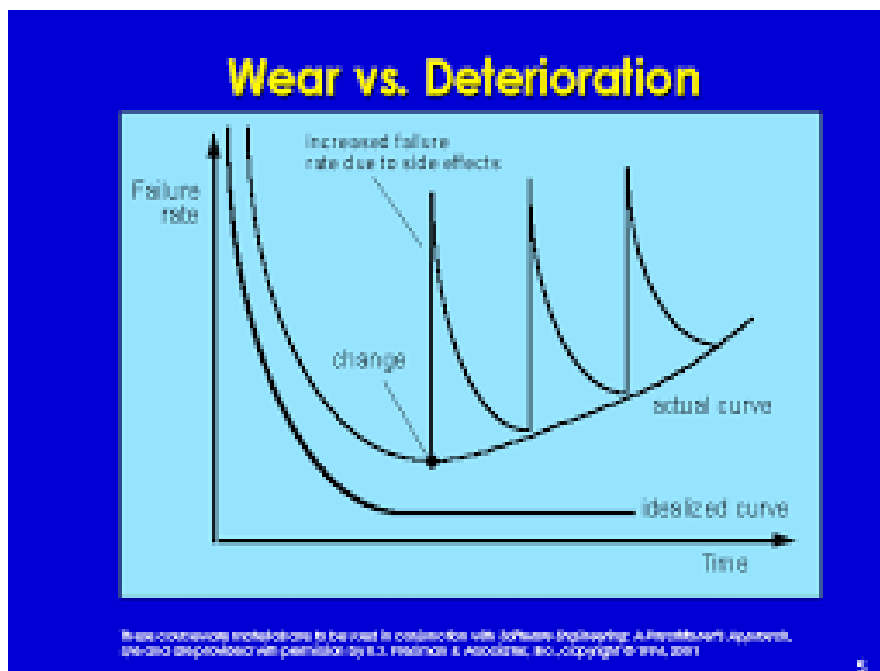
- Software costs often dominate computer system costs. The costs of software on a PC are often greater than the hardware cost.

- Software costs more to maintain than it does to develop. For systems with a long life, maintenance costs may be several times development costs.
- Software engineering is concerned with cost-effective software development.

Features of Software?

- Its characteristics that make it different from other things human being build. Features of such logical system:
- Software is developed or engineered, it is not manufactured in the classical sense which has quality problem.
- Software doesn't "wear out." but it deteriorates (due to change). Hardware has bathtub curve of failure rate (high failure rate in the beginning, then drop to steady state, then cumulative effects of dust, vibration, abuse occurs).
- Although the industry is moving toward component-based construction (e.g. standard screws and off-the-shelf integrated circuits), most software continues to be custom-built. Modern reusable components encapsulate data and processing into software parts to be reused by different programs. E.g. graphical user interface, window, pull-down menus in library etc.

Wear vs. Deterioration



Software Applications

- 1. System software: such as compilers, editors, file management utilities
- 2. Application software: stand-alone programs for specific needs.
- 3. Engineering/scientific software: Characterized by “number crunching” algorithms. such as automotive stress analysis, molecular biology, orbital dynamics etc
- 4. Embedded software resides within a product or system. (key pad control of a microwave oven, digital function of dashboard display in a car)
- 5. Product-line software focus on a limited marketplace to address mass consumer market. (word processing, graphics, database management)
- 6. WebApps (Web applications) network centric software. As web 2.0 emerges, more sophisticated computing environments is supported integrated with remote database and business applications.
- 7. AI software uses non-numerical algorithm to solve complex problem. Robotics, expert system, pattern recognition game playing

Software—New Categories

- Open world computing—pervasive, ubiquitous, distributed computing due to wireless networking. How to allow mobile devices, personal computer, enterprise system to communicate across vast network.
- Netsourcing—the Web as a computing engine. How to architect simple and sophisticated applications to target end-users worldwide.
- Open source—“free” source code open to the computing community (a blessing, but also a potential curse!)
- Also ... (see Chapter 31)
 - Data mining • Grid computing
 - Cognitive machines
 - Software for nanotechnologies

Software Engineering Definition

The seminal definition: [Software engineering is] the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.

The IEEE definition:

Software Engineering: (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 (1).

Importance of Software Engineering

- More and more, individuals and society rely on advanced software systems. We need to be able to produce reliable and trustworthy systems economically and quickly.
- It is usually cheaper, in the long run, to use software engineering methods and techniques for software systems rather than just write the programs as if it was a personal programming project. For most types of system, the majority of costs are the costs of changing the software after it has gone into use.

FAQ about software engineering

Question	Answer
What is software?	Computer programs, data structures and associated documentation. Software products may be developed for a particular customer or may be developed for a general market.
What are the attributes of good software?	Good software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable.
What is software engineering?	Software engineering is an engineering discipline that is concerned with all aspects of software production.
What is the difference between software engineering and computer science?	Computer science focuses on theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.
What is the difference between software engineering and system engineering?	System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this more general process.

Essential attributes of good software

Product characteristic	Description
Maintainability	Software should be written in such a way so that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.
Dependability and security	Software dependability includes a range of characteristics including reliability, security and safety. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system.
Efficiency	Software should not make wasteful use of system

	resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilisation, etc.
Acceptability	Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable and compatible with other systems that they use.

Software Engineering

Any engineering approach must rest on organizational commitment to quality which fosters a continuous process improvement culture.



Process layer as the foundation defines a framework with activities for effective delivery of software engineering technology. Establish the context where products (model, data, report, and forms) are produced, milestone are established, quality is ensured and change is managed.

Method provides technical how-to's for building software. It encompasses many tasks including communication, requirement analysis, design modeling, program construction, testing and support.

Tools provide automated or semi-automated support for the process and methods.

- A process is a collection of activities, actions and tasks that are performed when some work product is to be created. It is not a rigid prescription for how to build computer software. Rather, it is an adaptable approach that enables the people doing the work to pick and choose the appropriate set of work actions and tasks.
- Purpose of process is to deliver software in a timely manner and with sufficient quality to satisfy those who have sponsored its creation and those who will use it.

Software Process

- Communication: communicate with customer to understand objectives and gather requirements
- Planning: creates a “map” defines the work by describing the tasks, risks and resources, work products and work schedule.
- Modeling: Create a “sketch”, what it looks like architecturally, how the constituent parts fit together and other characteristics.
- Construction: code generation and the testing.
- Deployment: Delivered to the customer who evaluates the products and provides feedback based on the evaluation.
- These five framework activities can be used to all software development regardless of the application domain, size of the project, complexity of the efforts etc, though the details will be different in each case.
- For many software projects, these framework activities are applied iteratively as a project progresses. Each iteration produces a software increment that provides a subset of overall software features and functionality.

Five Activities of a Generic Process framework

Umbrella Activities

Complement the five process framework activities and help team manage and control progress, quality, change, and risk.

- Software project tracking and control: assess progress against the plan and take actions to maintain the schedule.
- Risk management: assesses risks that may affect the outcome and quality.
- Software quality assurance: defines and conduct activities to ensure quality.
- Technical reviews: assesses work products to uncover and remove errors before going to the next activity.
- Measurement: define and collects process, project, and product measures to ensure stakeholder’s needs are met.
- Software configuration management: manage the effects of change throughout the software process. • Reusability management: defines criteria for work product reuse and establishes mechanism to achieve reusable components.
- Work product preparation and production: create work products such as models, documents, logs, forms and lists.

The process should be agile and adaptable to problems. Process adopted for one project might be significantly different than a process adopted from another project. (to the problem, the project, the team, organizational culture). Among the differences are:

- the overall flow of activities, actions, and tasks and the interdependencies among them
- the degree to which actions and tasks are defined within each framework activity

- the degree to which work products are identified and required
- the manner which quality assurance activities are applied
- the manner in which project tracking and control activities are applied
- the overall degree of detail and rigor with which the process is described
- the degree to which the customer and other stakeholders are involved with the project
- the level of autonomy given to the software team
- the degree to which team organization and roles are prescribed

Adapting a Process Model

- The prescriptive process models stress detailed definition, identification, and application of process activities and tasks. Intent is to improve system quality, make projects more manageable, make delivery dates and costs more predictable, and guide teams of software engineers as they perform the work required to build a system.
- Unfortunately, there have been times when these objectives were not achieved. If prescriptive models are applied dogmatically and without adaptation, they can increase the level of bureaucracy.
- Agile process models emphasize project “agility” and follow a set of principles that lead to a more informal approach to software process. It emphasizes maneuverability and adaptability. It is particularly useful when Web applications are engineered.

Prescriptive and Agile Process Models

- How does the practice of software engineering fit in the process activities mentioned above? Namely, communication, planning, modeling, construction and deployment.
- George Polya outlines the essence of problem solving, suggests:
 - 1.Understand the problem (communication and analysis).
 - 2.Plan a solution (modeling and software design).
 - 3.Carry out the plan (code generation).
 - 4.Examine the result for accuracy (testing and quality assurance).

The Essence of Practice

- Who has a stake in the solution to the problem? That is, who are the stakeholders?
- What are the unknowns? What data, functions, and features are required to properly solve the problem?
- Can the problem be compartmentalized? Is it possible to represent smaller problems that may be easier to understand?
- Can the problem be represented graphically? Can an analysis model be created?

Understand the Problem

- Have you seen similar problems before? Are there patterns that are recognizable in a potential solution? Is there existing software that implements the data, functions, and features that are required?
- Has a similar problem been solved? If so, are elements of the solution reusable?
- Can subproblems be defined? If so, are solutions readily apparent for the subproblems?
- Can you represent a solution in a manner that leads to effective implementation? Can a design model be created?

Plan the Solution

- Does the solutions conform to the plan? Is source code traceable to the design model?
- Is each component part of the solution provably correct? Has the design and code been reviewed, or better, have correctness proofs been applied to algorithm?

Carry Out the Plan

- Is it possible to test each component part of the solution? Has a reasonable testing strategy been implemented?
- Does the solution produce results that conform to the data, functions, and features that are required? Has the software been validated against all stakeholder requirements?

Examine the Result

Help you establish mind-set for solid software engineering practice (David Hooker 96).

- 1: The Reason It All Exists: provide values to users
- 2: KISS (Keep It Simple, Stupid! As simple as possible)
- 3: Maintain the Vision (otherwise, incompatible design)
- 4: What You Produce, Others Will Consume (code with concern for those that must maintain and extend the system)
- 5: Be Open to the Future (never design yourself into a corner as specification and hardware changes)
- 6: Plan Ahead for Reuse
- 7: Think! Place clear complete thought before action produces better results.

Hooker's General Principles for Software Engineering Practice: important underlying law

Software Myths

Erroneous beliefs about software and the process that is used to build it.

- Affect managers, customers (and other non-technical stakeholders) and practitioners
- Are believable because they often have elements of truth, but ...

- Invariably lead to bad decisions, therefore ...
- Insist on reality as you navigate your way through software engineering

Software Myths Examples

- Myth 1: Once we write the program and get it to work, our job is done.
 - Reality: the sooner you begin writing code, the longer it will take you to get done. 60% to 80% of all efforts are spent after software is delivered to the customer for the first time.
 - Myth 2: Until I get the program running, I have no way of assessing its quality.
 - Reality: technical review are a quality filter that can be used to find certain classes of software defects from the inception of a project.
 - Myth 3: software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down.
 - Reality: it is not about creating documents. It is about creating a quality product. Better quality leads to a reduced rework. Reduced work results in faster delivery times.
 - Many people recognize the fallacy of the myths. Regrettably, habitual attitudes and methods foster poor management and technical practices, even when reality dictates a better approach.
- SafeHome:
 - Every software project is precipitated by some business need—
 - the need to correct a defect in an existing application;
 - the need to adapt a 'legacy system' to a changing business environment;
 - the need to extend the functions and features of an existing application, or
 - the need to create a new product, service, or system.

How It all Starts

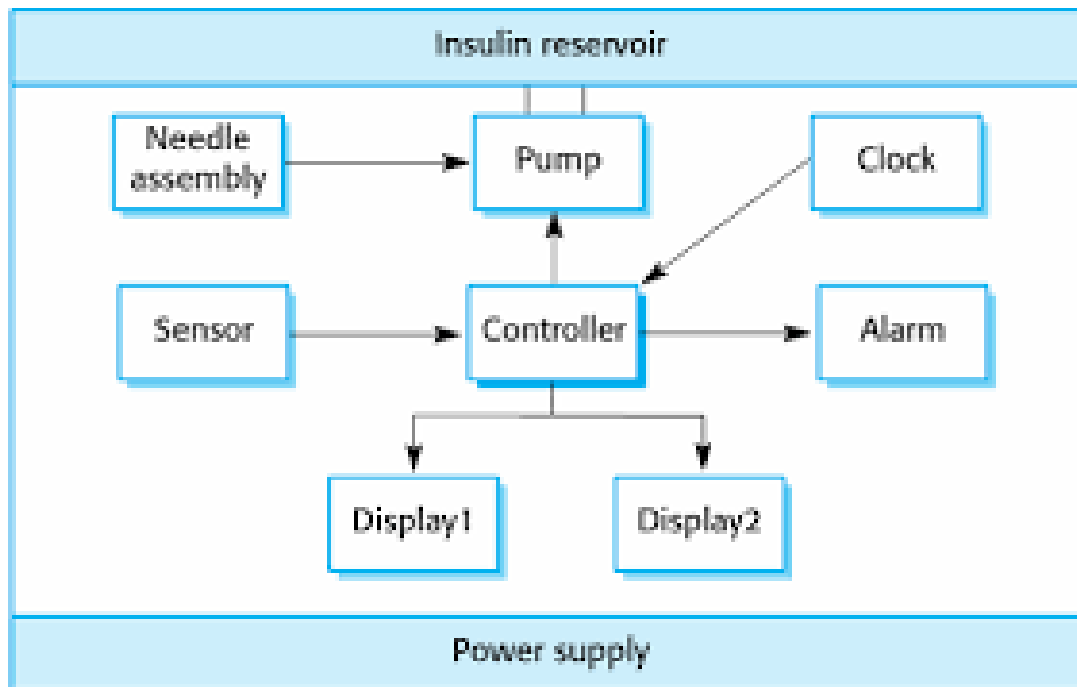
Case studies

- A personal insulin pump
- An embedded system in an insulin pump used by diabetics to maintain blood glucose control.
- A mental health case patient management system
- A system used to maintain records of people receiving care for mental health problems.
- A wilderness weather station
- A data collection system that collects data about weather conditions in remote areas.

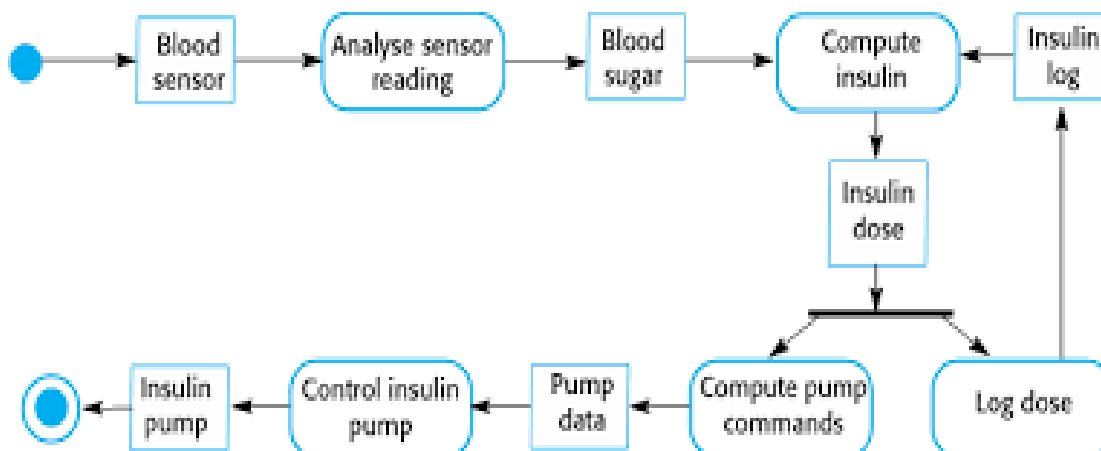
Insulin pump control system

- Collects data from a blood sugar sensor and calculates the amount of insulin required to be injected.
- Calculation based on the rate of change of blood sugar levels.
- Sends signals to a micro-pump to deliver the correct dose of insulin.
- Safety-critical system as low blood sugars can lead to brain malfunctioning, coma and death; high-blood sugar levels have long-term consequences such as eye and kidney damage.

Insulin pump hardware architecture



Activity model of the insulin pump



Essential high-level requirements

- The system shall be available to deliver insulin when required.
- The system shall perform reliably and deliver the correct amount of insulin to counteract the current level of blood sugar.
- The system must therefore be designed and implemented to ensure that the system always meets these requirements.

A patient information system for mental health care

- A patient information system to support mental health care is a medical information system that maintains information about patients suffering from mental health problems and the treatments that they have received.
- Most mental health patients do not require dedicated hospital treatment but need to attend specialist clinics regularly where they can meet a doctor who has detailed knowledge of their problems.
- To make it easier for patients to attend, these clinics are not just run in hospitals. They may also be held in local medical practices or community centres.

MHC-PMS

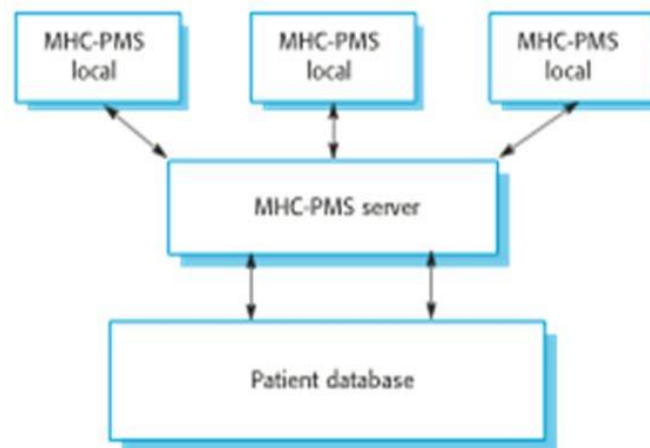
- The MHC-PMS (Mental Health Care-Patient Management System) is an information system that is intended for use in clinics.
- It makes use of a centralized database of patient information but has also been designed to run on a PC, so that it may be accessed and used from sites that do not have secure network connectivity.
- When the local systems have secure network access, they use patient information in the database but they can download and use local copies of patient records when they are disconnected.

MHC-PMS goals

- To generate management information that allows health service managers to assess performance against local and government targets.
- To provide medical staff with timely information to support the treatment of patients.

The organization of the MHC-PMS

The organization of the MHC-PMS



MHC-PMS key features

- Individual care management
- Clinicians can create records for patients, edit the information in the system, view patient history, etc. The system supports data summaries so that doctors can quickly learn about the key problems and treatments that have been prescribed.
- Patient monitoring
- The system monitors the records of patients that are involved in treatment and issues warnings if possible problems are detected.
- Administrative reporting
- The system generates monthly management reports showing the number of patients treated at each clinic, the number of patients who have entered and left the care system, number of patients sectioned, the drugs prescribed and their costs, etc.

MHC-PMS concerns

- Privacy
- It is essential that patient information is confidential and is never disclosed to anyone

apart from authorised medical staff and the patient themselves.

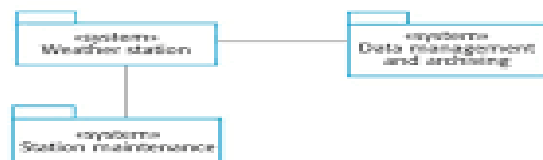
- Safety
- Some mental illnesses cause patients to become suicidal or a danger to other people. Wherever possible, the system should warn medical staff about potentially suicidal or dangerous patients.
- The system must be available when needed otherwise safety may be compromised and it may be impossible to prescribe the correct medication to patients.

Wilderness weather station

- The government of a country with large areas of wilderness decides to deploy several hundred weather stations in remote areas.
- Weather stations collect data from a set of instruments that measure temperature and pressure, sunshine, rainfall, wind speed and wind direction.
- The weather station includes a number of instruments that measure weather parameters such as the wind speed and direction, the ground and air temperatures, the barometric pressure and the rainfall over a 24-hour period. Each of these instruments is controlled by a software system that takes parameter readings periodically and manages the data collected from the instruments.

The weather station's environment

Figure 1.7 The weather station's environment



Weather information system

- The weather station system
- This is responsible for collecting weather data, carrying out some initial data processing and transmitting it to the data management system.
 - The data management and archiving system
- This system collects the data from all of the wilderness weather stations, carries out data processing and analysis and archives the data.
- The station maintenance system
 - This system can communicate by satellite with all wilderness weather stations to monitor the health of these systems and provide reports of

Additional software functionality

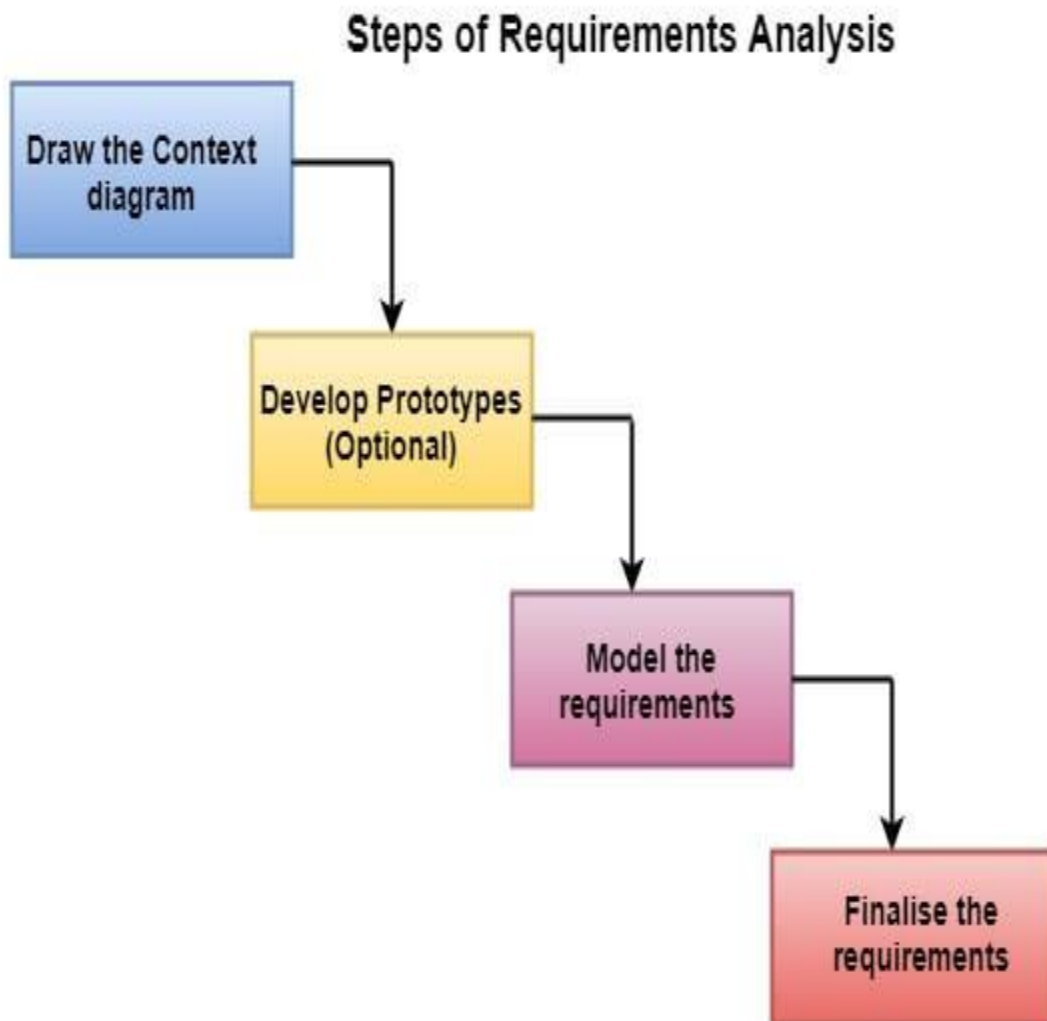
- Monitor the instruments, power and communication hardware and report faults to the management system.
- Manage the system power, ensuring that batteries are charged whenever the environmental conditions permit but also that generators are shut down in potentially damaging weather conditions, such as high wind.
- Support dynamic reconfiguration where parts of the software are replaced with new versions and where backup instruments are switched into the system in the event of system failure.

UNIT - II

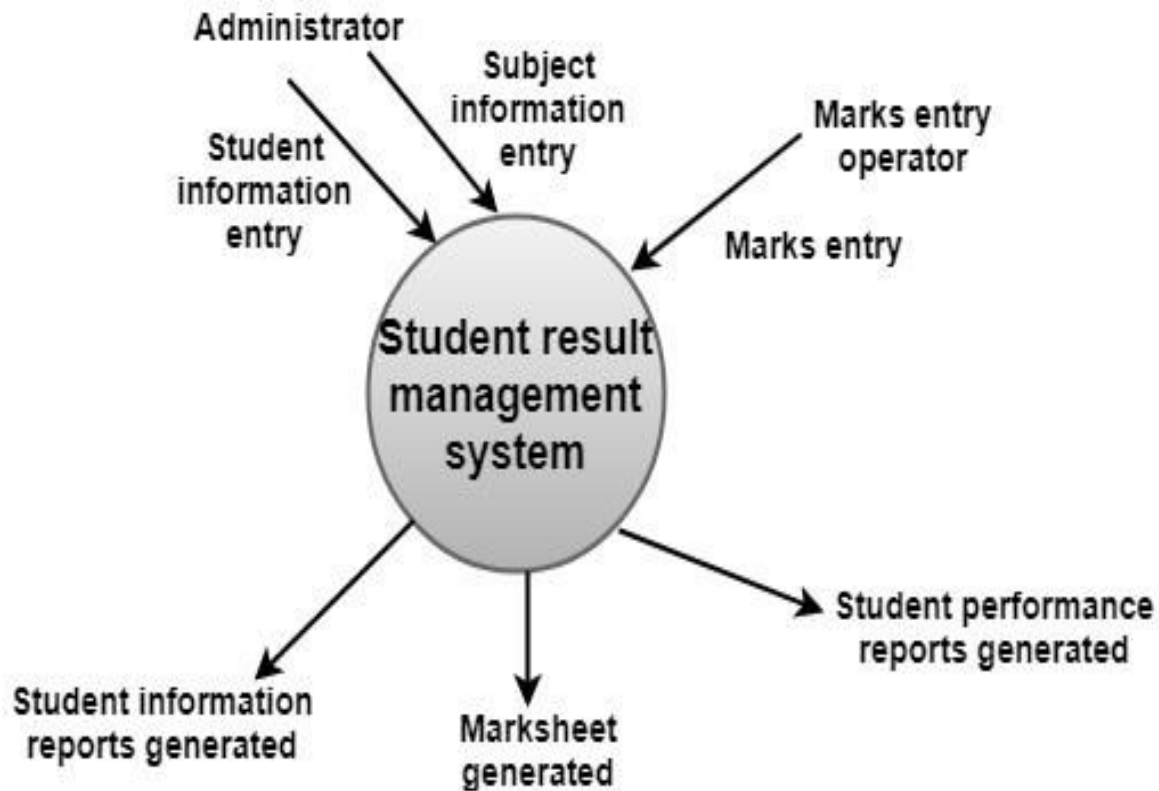
Requirements Analysis

Requirement analysis is significant and essential activity after elicitation. We analyze, refine, and scrutinize the gathered requirements to make consistent and unambiguous requirements. This activity reviews all requirements and may provide a graphical view of the entire system. After the completion of the analysis, it is expected that the understandability of the project may improve significantly. Here, we may also use the interaction with the customer to clarify points of confusion and to understand which requirements are more important than others.

The various steps of requirement analysis are shown in fig:



(i) Draw the context diagram: The context diagram is a simple model that defines the boundaries and interfaces of the proposed systems with the external world. It identifies the entities outside the proposed system that interact with the system. The context diagram of student result management system is given below:



(ii) Development of a Prototype (optional): One effective way to find out what the customer wants is to construct a prototype, something that looks and preferably acts as part of the system they say they want.

We can use their feedback to modify the prototype until the customer is satisfied continuously. Hence, the prototype helps the client to visualize the proposed system and increase the understanding of the requirements. When developers and users are not sure about some of the elements, a prototype may help both the parties to take a final decision.

Some projects are developed for the general market. In such cases, the prototype should be shown to some representative sample of the population of potential purchasers. Even though a person who tries out a prototype may not buy the final system, but their feedback may allow us to make the product more attractive to others.

The prototype should be built quickly and at a relatively low cost. Hence it will always have limitations and would not be acceptable in the final system. This is an optional activity.

(iii) Model the requirements: This process usually consists of various graphical representations of the functions, data entities, external entities, and the relationships between them. The graphical view may help to find incorrect, inconsistent, missing, and superfluous requirements. Such models include the Data Flow diagram, Entity-Relationship diagram, Data Dictionaries, State-transition diagrams, etc.

(iv) Finalise the requirements: After modeling the requirements, we will have a better understanding of the system behavior. The inconsistencies and ambiguities have been identified and corrected. The flow of data amongst various modules has been analyzed. Elicitation and analyze activities have provided better insight into the system. Now we finalize the analyzed requirements, and the next step is to document these requirements in a prescribed format.

Statement of system scope

When it comes to project planning, defining the project scope is the most critical step. In case if you start the project without knowing what you are supposed to be delivering at the end to the client and what the boundaries of the project are, there is a little chance for you to success. In most of the instances, you actually do not have any chance to success with this unorganized approach.

If you do not do a good job in project scope definition, project scope management during the project execution is almost impossible.

The main purpose of the scope definition is to clearly describe the boundaries of your project. Clearly describing the boundaries is not enough when it comes to project. You need to get the client's agreement as well.

Therefore, the defined scope of the project usually included into the contractual agreements between the client and the service provider. SOW, or in other words, Statement of Work, is one such document.

In the project scope definition, the elements within the scope and out of the scope are well defined in order to clearly understand what will be the area under the project control. Therefore, you should identify more elements in detailed manner and divide them among the scope and out of scope.

How to Define the Project Scope

When the project is about to be funded, there should be a set of defined deliveries and objectives for the project. There can be a high level-scope statement prepared at this level.

This high-level scope statement can be taken from the initial documents such as SOW. In addition to the SOW, you need to use any other document or information in order to further define the project scope at this level.

In case, if you feel that you do not have enough information to come up with a high-level scope statement, you should then work closely with the client in order to gather necessary information.

Project objectives can be used for defining the project scope. As a matter of fact, there should be one or more deliverables addressing each project objective in the project. By looking at the deliverables, you can actually gauge the project scope.

Once you get to know the main deliverables of the project, start asking questions about the other processes and different aspects of the project.

First identifying and clearly defining the out of scope also helps you to understand the scope of a project. When you go on defining the out of scope, you will automatically get an idea of the real project scope. In order to follow this method, you need to have a defined scope up to a certain level.

Whenever you identify an item for the scope or out-of-scope, make sure you document it then and there. Later, you can revisit these items and elaborate more on those.

Once you have successfully defined the scope of the project, you need to get the sign-off from the related and applicable parties. Without proper sign-off for the scope, the next phases of the project, i.e., requirements gathering, might have issues in executing.

Scope Creep

Scope creep is something common with every project. This refers to the incremental expansion of the project scope. Most of the time, the client may come back to the service provider during the project execution and add more requirements.

Most of such requirements haven't been in the initial requirements. As a result, change requests need to be raised in order to cover the increasing costs of the service provider.

Due to business cope creep, there can be technological scope creep as well. The project team may require new technologies in order to address some of the new requirements in the scope.

In such instances, the service provider may want to work with the client closely and make necessary logistic and financial arrangements.

Conclusion

Project scope definition is the most important factor when it comes to project requirements. It is vital for service providers to define the scope of the project in order to successfully enter into an agreement with the client.

In addition to this, the scope of the project gives an idea to the services provider about the estimated cost of the project. Therefore, service provider's profit margins are wholly dependent on the accuracy of the project scope definition.

Introduction

One of the biggest decisions that any organization would have to make is related to the projects they would undertake. Once a proposal has been received, there are numerous factors that need to be considered before an organization decides to take it up.

The most viable option needs to be chosen, keeping in mind the goals and requirements of the organization. How is it then that you decide whether a project is viable? How do you decide if the project at hand is worth approving? This is where project selection methods come in use.

Choosing a project using the right method is therefore of utmost importance. This is what will ultimately define the way the project is to be carried out.

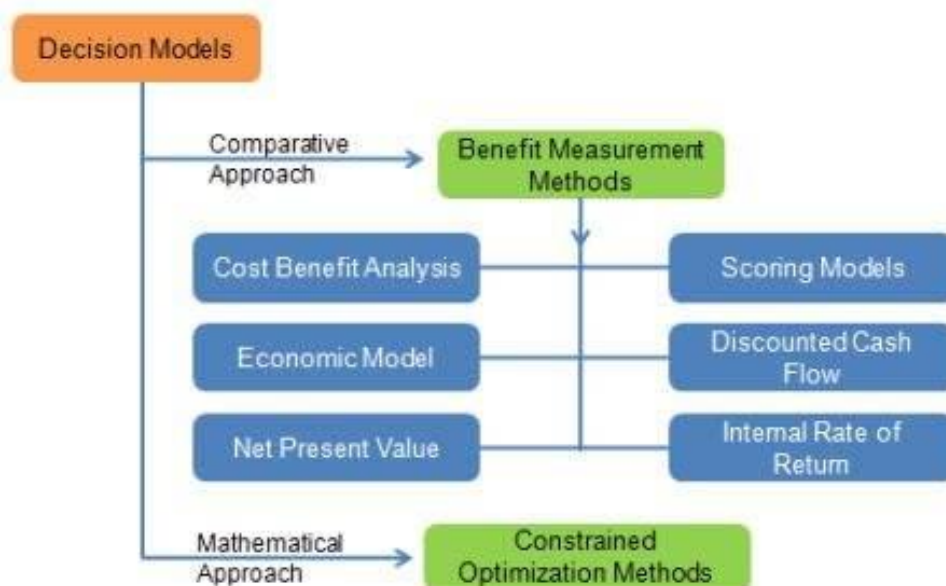
But the question then arises as to how you would go about finding the right methodology for your particular organization. At this instance, you would need careful guidance in the project selection criteria, as a small mistake could be detrimental to your project as a whole, and in the long run, the organization as well.

Selection Methods

There are various project selection methods practised by the modern business organizations. These methods have different features and characteristics. Therefore, each selection method is best for different organizations.

Although there are many differences between these project selection methods, usually the underlying concepts and principles are the same.

Following is an illustration of two of such methods (Benefit Measurement and Constrained Optimization methods):



As the value of one project would need to be compared against the other projects, you could use the benefit measurement methods. This could include various techniques, of which the following are the most common:

- You and your team could come up with certain criteria that you want your ideal project objectives to meet. You could then give each project scores based on how they rate in each of these criteria and then choose the project with the highest score.
- When it comes to the Discounted Cash flow method, the future value of a project is ascertained by considering the present value and the interest earned on the money. The higher the present value of the project, the better it would be for your organization.
- The rate of return received from the money is what is known as the IRR. Here again, you need to be looking for a high rate of return from the project.

The mathematical approach is commonly used for larger projects. The constrained optimization methods require several calculations in order to decide on whether or not a project should be rejected.

Cost-benefit analysis is used by several organizations to assist them to make their selections. Going by this method, you would have to consider all the positive aspects of the project which are the benefits and then deduct the negative aspects (or the costs) from the benefits. Based on the results you receive for different projects, you could choose which option would be the most viable and financially rewarding.

These benefits and costs need to be carefully considered and quantified in order to arrive at a proper conclusion. Questions that you may want to consider asking in the selection process are:

- Would this decision help me to increase organizational value in the long run?
- How long will the equipment last for?
- Would I be able to cut down on costs as I go along?

In addition to these methods, you could also consider choosing based on opportunity cost - When choosing any project, you would need to keep in mind the profits that you would make if you decide to go ahead with the project.

Profit optimization is therefore the ultimate goal. You need to consider the difference between the profits of the project you are primarily interested in and the next best alternative.

Implementation of the Chosen Method

The methods mentioned above can be carried out in various combinations. It is best that you try out different methods, as in this way you would be able to make the best decision for your organization considering a wide range of factors rather than concentrating on just a few. Careful consideration would therefore need to be given to each project.

Conclusion

In conclusion, you would need to remember that these methods are time-consuming, but are absolutely essential for efficient business planning.

It is always best to have a good plan from the inception, with a list of criteria to be considered and goals to be achieved. This will guide you through the entire selection process and will also ensure that you do make the right choice.

Isolation of top level processes and entities and their allocation to physical elements:-

Systems development is systematic process which includes phases such as planning, analysis, design, deployment, and maintenance. Here, in this tutorial, we will primarily focus on –

- Systems analysis
- Systems design

Systems Analysis

It is a process of collecting and interpreting facts, identifying the problems, and decomposition of a system into its components.

System analysis is conducted for the purpose of studying a system or its parts in order to identify its objectives. It is a problem solving technique that improves the system and ensures that all the components of the system work efficiently to accomplish their purpose.

Analysis specifies **what the system should do**.

Systems Design

It is a process of planning a new business system or replacing an existing system by defining its components or modules to satisfy the specific requirements. Before planning, you need to understand the old system thoroughly and determine how computers can best be used in order to operate efficiently.

System Design focuses on **how to accomplish the objective of the system**.

System Analysis and Design (SAD) mainly focuses on –

- Systems
- Processes
- Technology

What is a System?

The word System is derived from Greek word Systema, which means an organized relationship between any set of components to achieve some common cause or objective.

A system is “an orderly grouping of interdependent components linked together according to a plan to achieve a specific goal.”

Constraints of a System

A system must have three basic constraints –

- A system must have some **structure and behavior** which is designed to achieve a predefined objective.
- Interconnectivity and interdependence must exist among the system components.
- The objectives of the organization have a higher priority than the objectives of its subsystems.

For example, traffic management system, payroll system, automatic library system, human resources information system.

Properties of a System

A system has the following properties –

Organization

Organization implies structure and order. It is the arrangement of components that helps to achieve predetermined objectives.

Interaction

It is defined by the manner in which the components operate with each other.

For example, in an organization, purchasing department must interact with production department and payroll with personnel department.

Interdependence

Interdependence means how the components of a system depend on one another. For proper functioning, the components are coordinated and linked together according to a specified plan. The output of one subsystem is the required by other subsystem as input.

Integration

Integration is concerned with how a system components are connected together. It means that the parts of the system work together within the system even if each part performs a unique function.

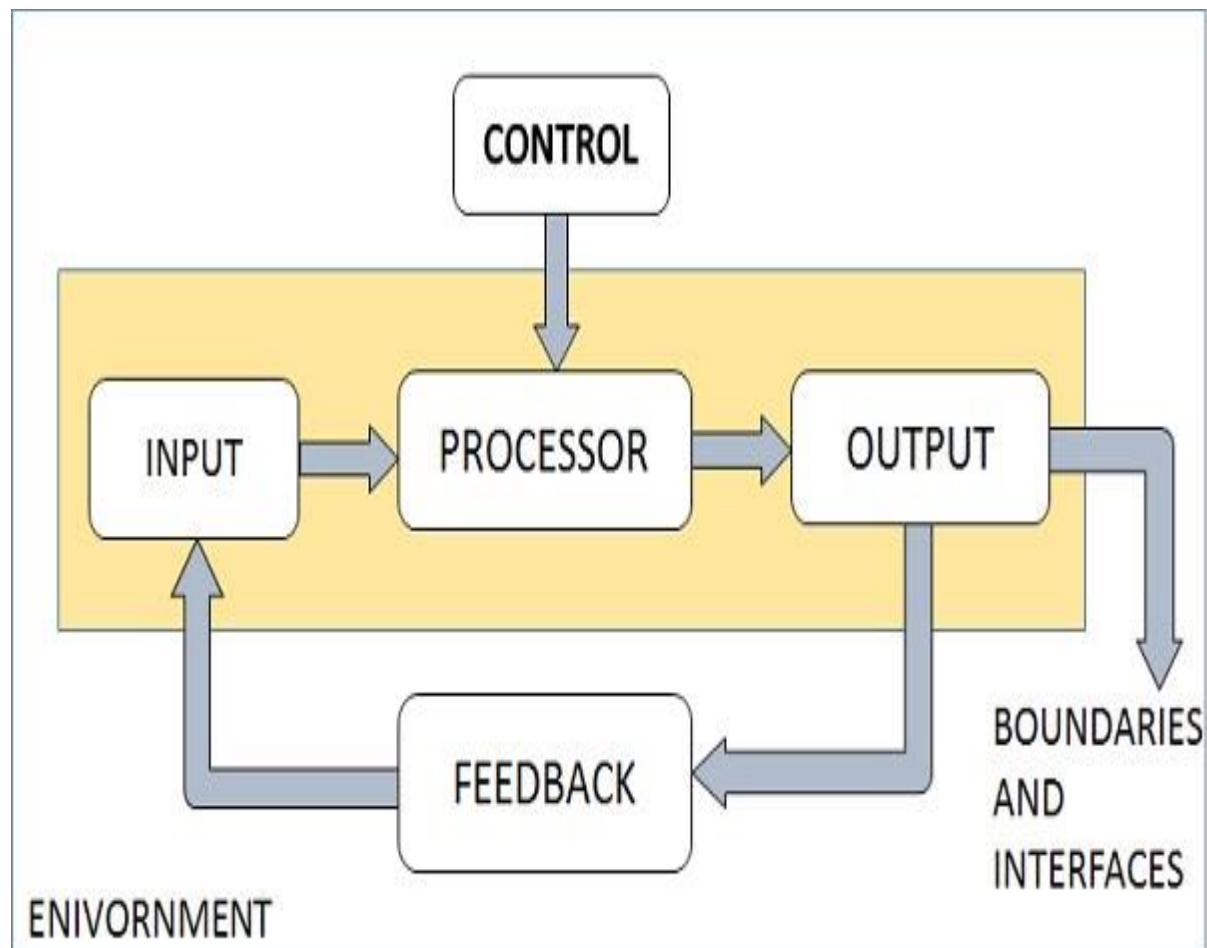
Central Objective

The objective of system must be central. It may be real or stated. It is not uncommon for an organization to state an objective and operate to achieve another.

The users must know the main objective of a computer application early in the analysis for a successful design and conversion.

Elements of a System

The following diagram shows the elements of a system –



Outputs and Inputs

- The main aim of a system is to produce an output which is useful for its user.
- Inputs are the information that enters into the system for processing.
- Output is the outcome of processing.

Processor(s)

- The processor is the element of a system that involves the actual transformation of input into output.
- It is the operational component of a system. Processors may modify the input either totally or partially, depending on the output specification.
- As the output specifications change, so does the processing. In some cases, input is also modified to enable the processor for handling the transformation.

Control

- The control element guides the system.
- It is the decision-making subsystem that controls the pattern of activities governing input, processing, and output.
- The behavior of a computer System is controlled by the Operating System and software. In order to keep system in balance, what and how much input is needed is determined by Output Specifications.

Feedback

- Feedback provides the control in a dynamic system.
- Positive feedback is routine in nature that encourages the performance of the system.
- Negative feedback is informational in nature that provides the controller with information for action.

Environment

- The environment is the “supersystem” within which an organization operates.
- It is the source of external elements that strike on the system.
- It determines how a system must function. For example, vendors and competitors of organization’s environment, may provide constraints that affect the actual performance of the business.

Boundaries and Interface

- A system should be defined by its boundaries. Boundaries are the limits that identify its components, processes, and interrelationship when it interfaces with another system.
- Each system has boundaries that determine its sphere of influence and control.

- The knowledge of the boundaries of a given system is crucial in determining the nature of its interface with other systems for successful design.

Types of Systems

The systems can be divided into the following types –

Physical or Abstract Systems

- Physical systems are tangible entities. We can touch and feel them.
- Physical System may be static or dynamic in nature. For example, desks and chairs are the physical parts of computer center which are static. A programmed computer is a dynamic system in which programs, data, and applications can change according to the user's needs.
- Abstract systems are non-physical entities or conceptual that may be formulas, representation or model of a real system.

Open or Closed Systems

- An open system must interact with its environment. It receives inputs from and delivers outputs to the outside of the system. For example, an information system which must adapt to the changing environmental conditions.
- A closed system does not interact with its environment. It is isolated from environmental influences. A completely closed system is rare in reality.

Adaptive and Non Adaptive System

- Adaptive System responds to the change in the environment in a way to improve their performance and to survive. For example, human beings, animals.
- Non Adaptive System is the system which does not respond to the environment. For example, machines.

Permanent or Temporary System

- Permanent System persists for long time. For example, business policies.
- Temporary System is made for specified time and after that they are demolished. For example, A DJ system is set up for a program and it is dissembled after the program.

Natural and Manufactured System

- Natural systems are created by the nature. For example, Solar system, seasonal system.

- Manufactured System is the man-made system. For example, Rockets, dams, trains.

Deterministic or Probabilistic System

- Deterministic system operates in a predictable manner and the interaction between system components is known with certainty. For example, two molecules of hydrogen and one molecule of oxygen makes water.
- Probabilistic System shows uncertain behavior. The exact output is not known. For example, Weather forecasting, mail delivery.

Social, Human-Machine, Machine System

- Social System is made up of people. For example, social clubs, societies.
- In Human-Machine System, both human and machines are involved to perform a particular task. For example, Computer programming.
- Machine System is where human interference is neglected. All the tasks are performed by the machine. For example, an autonomous robot.

Man-Made Information Systems

- It is an interconnected set of information resources to manage data for particular organization, under Direct Management Control (DMC).
- This system includes hardware, software, communication, data, and application for producing information according to the need of an organization.

Man-made information systems are divided into three types –

- **Formal Information System** – It is based on the flow of information in the form of memos, instructions, etc., from top level to lower levels of management.
- **Informal Information System** – This is employee based system which solves the day to day work related problems.
- **Computer Based System** – This system is directly dependent on the computer for managing business applications. For example, automatic library system, railway reservation system, banking system, etc.

Systems Models

Schematic Models

- A schematic model is a 2-D chart that shows system elements and their linkages.
- Different arrows are used to show information flow, material flow, and information feedback.

Flow System Models

- A flow system model shows the orderly flow of the material, energy, and information that hold the system together.
- Program Evaluation and Review Technique (PERT), for example, is used to abstract a real world system in model form.

Static System Models


- They represent one pair of relationships such as activity–time or cost–quantity.
- The Gantt chart, for example, gives a static picture of an activity-time relationship.

Dynamic System Models

- Business organizations are dynamic systems. A dynamic model approximates the type of organization or application that analysts deal with.
- It shows an ongoing, constantly changing status of the system. It consists of –
 - Inputs that enter the system
 - The processor through which transformation takes place
 - The program(s) required for processing
 - The output(s) that result from processing.

Categories of Information

There are three categories of information related to managerial levels and the decision managers make.

Volume of Information	Type of Information	Information Level	Management Level	System Support
Low Condensed	Unstructured		Upper	DSS
Medium Moderately Processed	Moderately Structured		Middle	MIS
Large Detail Reports	Highly Structured		Lower	DPS

Strategic Information

- This information is required by topmost management for long range planning policies for next few years. For example, trends in revenues, financial investment, and human resources, and population growth.
- This type of information is achieved with the aid of Decision Support System (DSS).

Managerial Information

- This type of Information is required by middle management for short and intermediate range planning which is in terms of months. For example, sales analysis, cash flow projection, and annual financial statements.
- It is achieved with the aid of Management Information Systems (MIS).

Operational information

- This type of information is required by low management for daily and short term planning to enforce day-to-day operational activities. For example, keeping

employee attendance records, overdue purchase orders, and current stocks available.

- It is achieved with the aid of Data Processing Systems (DPS).

System Development Life Cycle

An effective System Development Life Cycle (SDLC) should result in a high quality system that meets customer expectations, reaches completion within time and cost evaluations, and works effectively and efficiently in the current and planned Information Technology infrastructure.

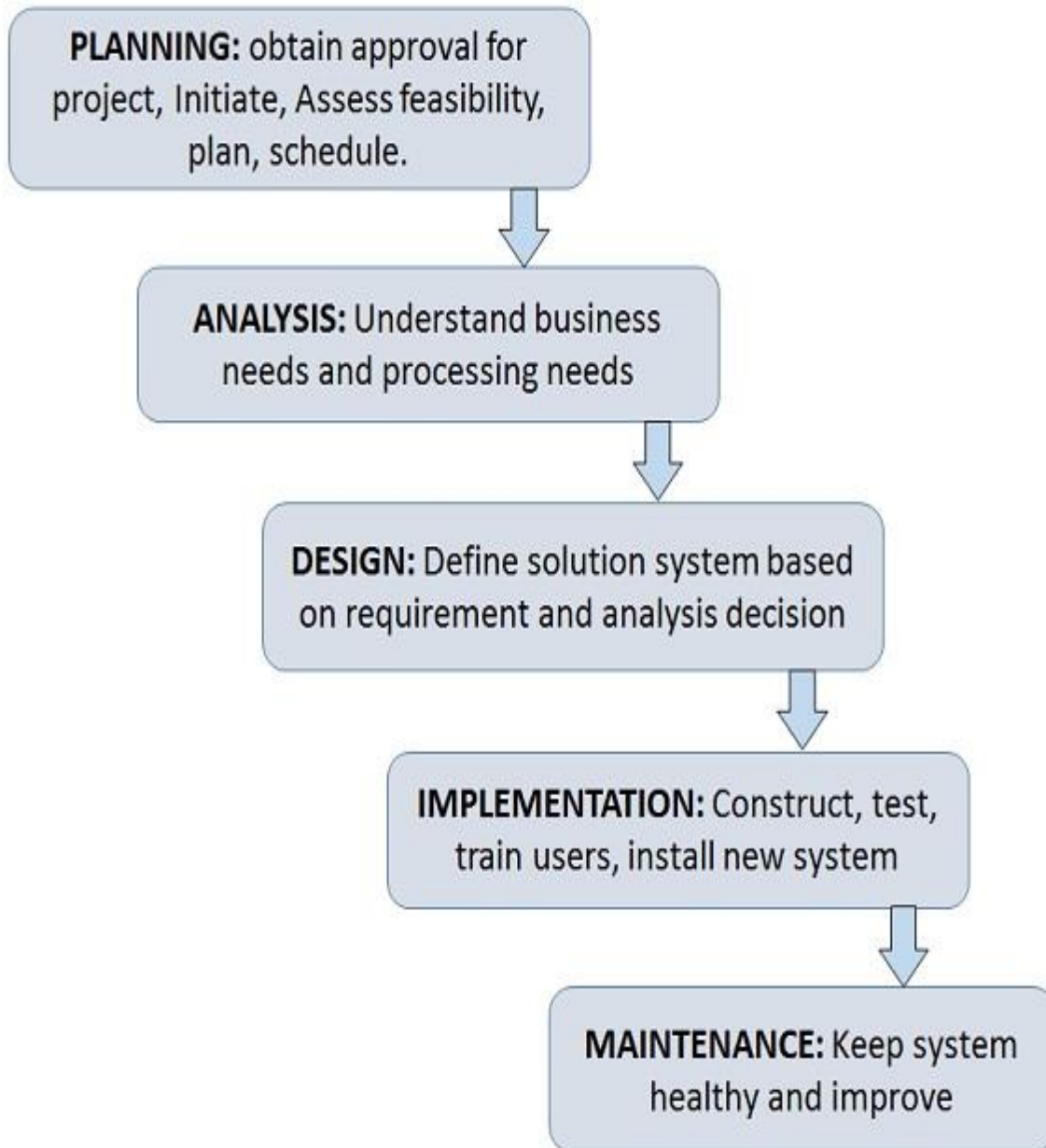
System Development Life Cycle (SDLC) is a conceptual model which includes policies and procedures for developing or altering systems throughout their life cycles.

SDLC is used by analysts to develop an information system. SDLC includes the following activities –

- Requirements
- Design
- Implementation
- Testing
- Deployment
- Operations
- maintenance

Phases of SDLC

Systems Development Life Cycle is a systematic approach which explicitly breaks down the work into phases that are required to implement either new or modified Information System.



Feasibility Study or Planning

- Define the problem and scope of existing system.
- Overview the new system and determine its objectives.
- Confirm project feasibility and produce the project Schedule.
- During this phase, threats, constraints, integration and security of system are also considered.
- A feasibility report for the entire project is created at the end of this phase.

Analysis and Specification

- Gather, analyze, and validate the information.
- Define the requirements and prototypes for new system.
- Evaluate the alternatives and prioritize the requirements.
- Examine the information needs of end-user and enhances the system goal.
- A Software Requirement Specification (SRS) document, which specifies the software, hardware, functional, and network requirements of the system is prepared at the end of this phase.

System Design

- Includes the design of application, network, databases, user interfaces, and system interfaces.
- Transform the SRS document into logical structure, which contains detailed and complete set of specifications that can be implemented in a programming language.
- Create a contingency, training, maintenance, and operation plan.
- Review the proposed design. Ensure that the final design must meet the requirements stated in SRS document.
- Finally, prepare a design document which will be used during next phases.

Implementation

- Implement the design into source code through coding.
- Combine all the modules together into training environment that detects errors and defects.
- A test report which contains errors is prepared through test plan that includes test related tasks such as test case generation, testing criteria, and resource allocation for testing.
- Integrate the information system into its environment and install the new system.

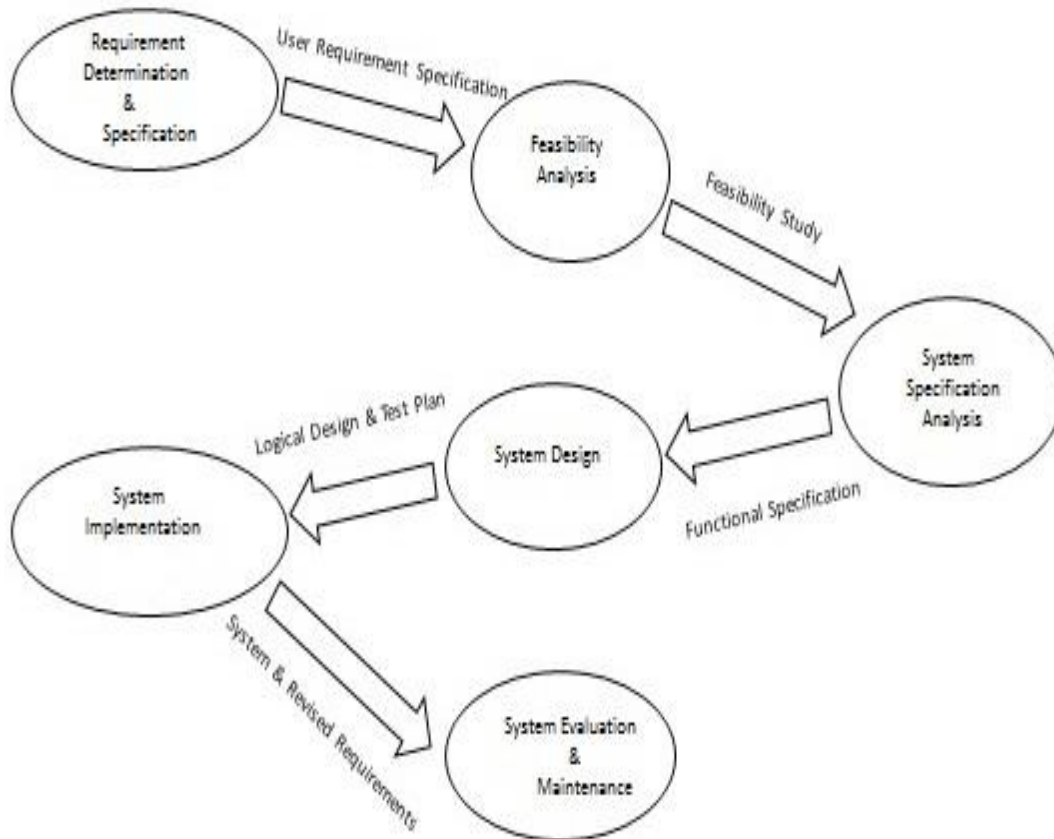
Maintenance/Support

- Include all the activities such as phone support or physical on-site support for users that is required once the system is installing.
- Implement the changes that software might undergo over a period of time, or implement any new requirements after the software is deployed at the customer location.

- It also includes handling the residual errors and resolve any issues that may exist in the system even after the testing phase.
- Maintenance and support may be needed for a longer time for large systems and for a short time for smaller systems.

Life Cycle of System Analysis and Design

The following diagram shows the complete life cycle of the system during analysis and design phase.



Role of System Analyst

The system analyst is a person who is thoroughly aware of the system and guides the system development project by giving proper directions. He is an expert having technical and interpersonal skills to carry out development tasks required at each phase.

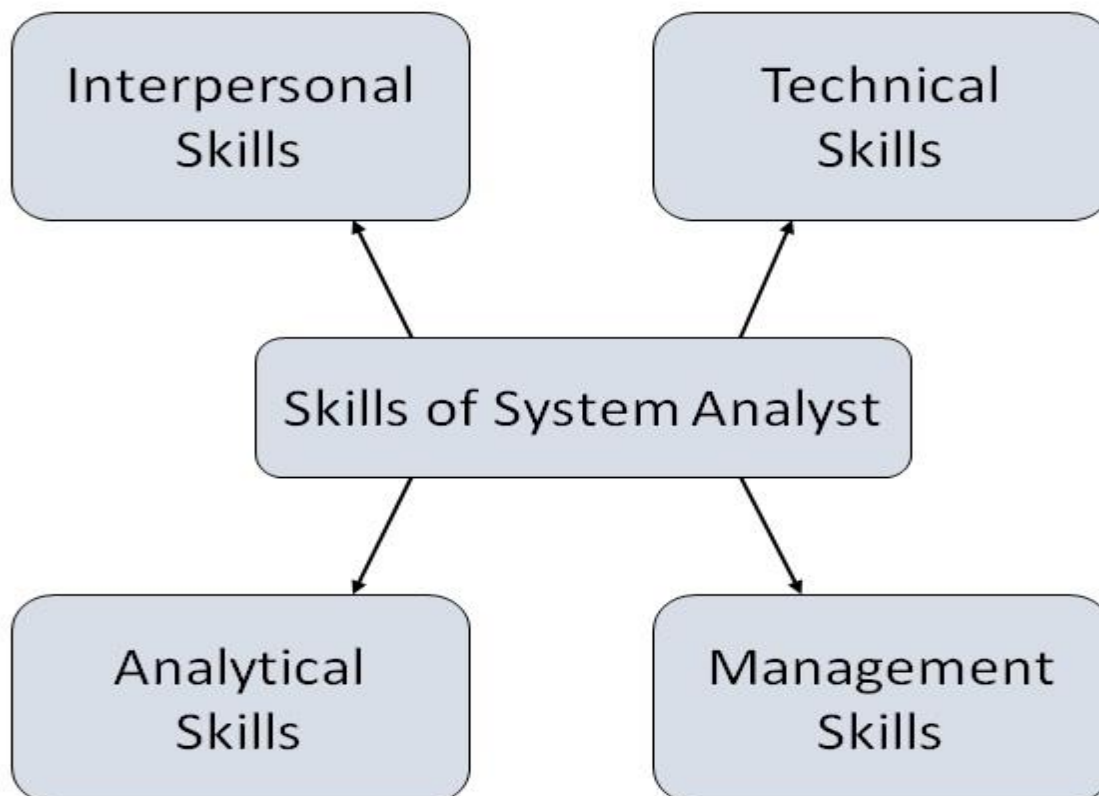
He pursues to match the objectives of information system with the organization goal.

Main Roles

- Defining and understanding the requirement of user through various Fact finding techniques.
- Prioritizing the requirements by obtaining user consensus.
- Gathering the facts or information and acquires the opinions of users.
- Maintains analysis and evaluation to arrive at appropriate system which is more user friendly.
- Suggests many flexible alternative solutions, pick the best solution, and quantify cost and benefits.
- Draw certain specifications which are easily understood by users and programmer in precise and detailed form.
- Implemented the logical design of system which must be modular.
- Plan the periodicity for evaluation after it has been used for some time, and modify the system as needed.

Attributes of a Systems Analyst

The following figure shows the attributes a systems analyst should possess –



Interpersonal Skills

- Interface with users and programmer.
- Facilitate groups and lead smaller teams.
- Managing expectations.
- Good understanding, communication, selling and teaching abilities.
- Motivator having the confidence to solve queries.

Analytical Skills

- System study and organizational knowledge
- Problem identification, problem analysis, and problem solving
- Sound commonsense
- Ability to access trade-off
- Curiosity to learn about new organization

Management Skills

- Understand users jargon and practices.
- Resource & project management.
- Change & risk management.
- Understand the management functions thoroughly.

Technical Skills

- Knowledge of computers and software.
- Keep abreast of modern development.
- Know of system design tools.
- Breadth knowledge about new technologies.

System Analysis & Design - System Planning

What is Requirements Determination?

A requirement is a vital feature of a new system which may include processing or capturing of data, controlling the activities of business, producing information and supporting the management.

Requirements determination involves studying the existing system and gathering details to find out what are the requirements, how it works, and where improvements should be made.

Major Activities in requirement Determination

Requirements Anticipation

- It predicts the characteristics of system based on previous experience which include certain problems or features and requirements for a new system.
- It can lead to analysis of areas that would otherwise go unnoticed by inexperienced analyst. But if shortcuts are taken and bias is introduced in conducting the investigation, then requirement Anticipation can be half-baked.

Requirements Investigation

- It is studying the current system and documenting its features for further analysis.
- It is at the heart of system analysis where analyst documenting and describing system features using fact-finding techniques, prototyping, and computer assisted tools.

Requirements Specifications

- It includes the analysis of data which determine the requirement specification, description of features for new system, and specifying what information requirements will be provided.
- It includes analysis of factual data, identification of essential requirements, and selection of Requirement-fulfillment strategies.

Information Gathering Techniques

The main aim of fact finding techniques is to determine the information requirements of an organization used by analysts to prepare a precise SRS understood by user.

Ideal SRS Document should –

- be complete, Unambiguous, and Jargon-free.
- specify operational, tactical, and strategic information requirements.
- solve possible disputes between users and analyst.
- use graphical aids which simplify understanding and design.

There are various information gathering techniques –

Interviewing

Systems analyst collects information from individuals or groups by interviewing. The analyst can be formal, legalistic, play politics, or be informal; as the success of an interview depends on the skill of analyst as interviewer.

It can be done in two ways –

- **Unstructured Interview** – The system analyst conducts question-answer session to acquire basic information of the system.
- **Structured Interview** – It has standard questions which user need to respond in either close (objective) or open (descriptive) format.

Advantages of Interviewing

- This method is frequently the best source of gathering qualitative information.
- It is useful for them, who do not communicate effectively in writing or who may not have the time to complete questionnaire.
- Information can easily be validated and cross checked immediately.
- It can handle the complex subjects.
- It is easy to discover key problem by seeking opinions.
- It bridges the gaps in the areas of misunderstandings and minimizes future problems.

Questionnaires

This method is used by analyst to gather information about various issues of system from large number of persons.

There are two types of questionnaires –

- **Open-ended Questionnaires** – It consists of questions that can be easily and correctly interpreted. They can explore a problem and lead to a specific direction of answer.
- **Closed-ended Questionnaires** – It consists of questions that are used when the systems analyst effectively lists all possible responses, which are mutually exclusive.

Advantages of questionnaires

- It is very effective in surveying interests, attitudes, feelings, and beliefs of users which are not co-located.
- It is useful in situation to know what proportion of a given group approves or disapproves of a particular feature of the proposed system.
- It is useful to determine the overall opinion before giving any specific direction to the system project.
- It is more reliable and provides high confidentiality of honest responses.
- It is appropriate for electing factual information and for statistical data collection which can be emailed and sent by post.

Review of Records, Procedures, and Forms

Review of existing records, procedures, and forms helps to seek insight into a system which describes the current system capabilities, its operations, or activities.

Advantages

- It helps user to gain some knowledge about the organization or operations by themselves before they impose upon others.
- It helps in documenting current operations within short span of time as the procedure manuals and forms describe the format and functions of present system.
- It can provide a clear understanding about the transactions that are handled in the organization, identifying input for processing, and evaluating performance.
- It can help an analyst to understand the system in terms of the operations that must be supported.
- It describes the problem, its affected parts, and the proposed solution.

Observation

This is a method of gathering information by noticing and observing the people, events, and objects. The analyst visits the organization to observe the working of current system and understands the requirements of the system.

Advantages

- It is a direct method for gleaning information.
- It is useful in situation where authenticity of data collected is in question or when complexity of certain aspects of system prevents clear explanation by end-users.
- It produces more accurate and reliable data.
- It produces all the aspect of documentation that are incomplete and outdated.

Joint Application Development (JAD)

It is a new technique developed by IBM which brings owners, users, analysts, designers, and builders to define and design the system using organized and intensive workshops. JAD trained analyst act as facilitator for workshop who has some specialized skills.

Advantages of JAD

- It saves time and cost by replacing months of traditional interviews and follow-up meetings.
- It is useful in organizational culture which supports joint problem solving.
- Fosters formal relationships among multiple levels of employees.
- It can lead to development of design creatively.
- It Allows rapid development and improves ownership of information system.

Secondary Research or Background Reading

This method is widely used for information gathering by accessing the gleaned information. It includes any previously gathered information used by the marketer from any internal or external source.

Advantages

- It is more openly accessed with the availability of internet.
- It provides valuable information with low cost and time.
- It act as forerunner to primary research and aligns the focus of primary research.
- It is used by the researcher to conclude if the research is worth it as it is available with procedures used and issues in collecting them.

Feasibility Study

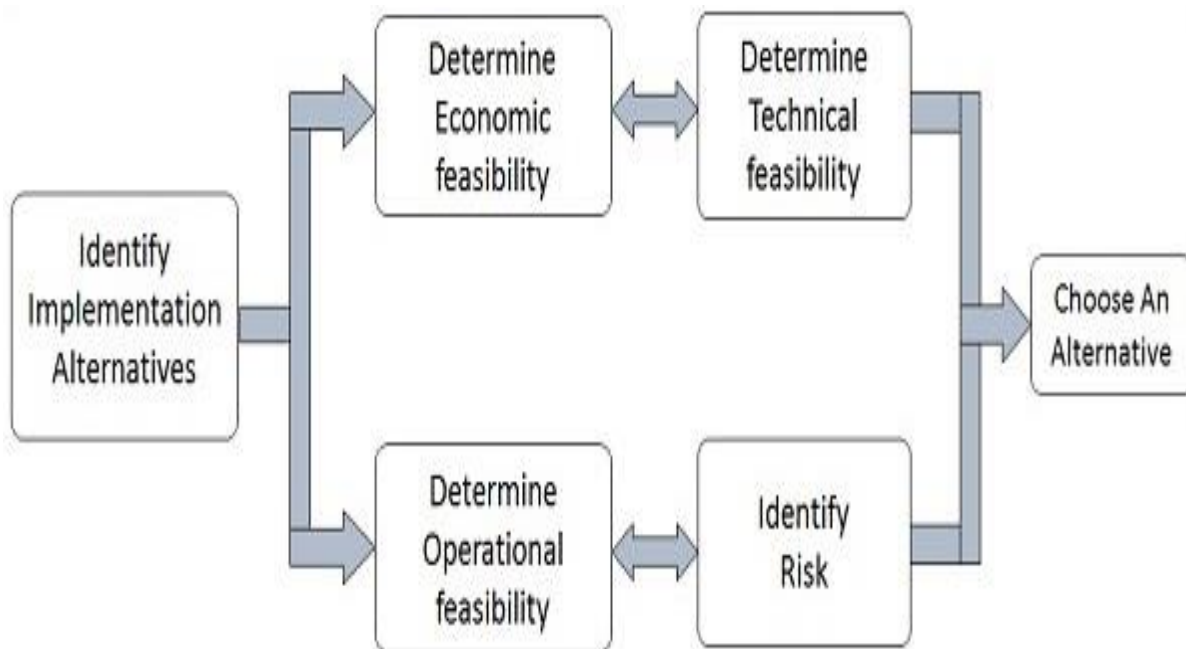
Feasibility Study can be considered as preliminary investigation that helps the management to take decision about whether study of system should be feasible for development or not.

- It identifies the possibility of improving an existing system, developing a new system, and produce refined estimates for further development of system.
- It is used to obtain the outline of the problem and decide whether feasible or appropriate solution exists or not.
- The main objective of a feasibility study is to acquire problem scope instead of solving the problem.
- The output of a feasibility study is a formal system proposal act as decision document which includes the complete nature and scope of the proposed system.

Steps Involved in Feasibility Analysis

The following steps are to be followed while performing feasibility analysis –

- Form a project team and appoint a project leader.
- Develop system flowcharts.
- Identify the deficiencies of current system and set goals.
- Enumerate the alternative solution or potential candidate system to meet goals.
- Determine the feasibility of each alternative such as technical feasibility, operational feasibility, etc.
- Weight the performance and cost effectiveness of each candidate system.
- Rank the other alternatives and select the best candidate system.
- Prepare a system proposal of final project directive to management for approval.



Types of Feasibilities

Economic Feasibility

- It is evaluating the effectiveness of candidate system by using cost/benefit analysis method.
- It demonstrates the net benefit from the candidate system in terms of benefits and costs to the organization.
- The main aim of Economic Feasibility Analysis (EFS) is to estimate the economic requirements of candidate system before investments funds are committed to proposal.
- It prefers the alternative which will maximize the net worth of organization by earliest and highest return of funds along with lowest level of risk involved in developing the candidate system.

Technical Feasibility

- It investigates the technical feasibility of each implementation alternative.
- It analyzes and determines whether the solution can be supported by existing technology or not.
- The analyst determines whether current technical resources be upgraded or added it that fulfill the new requirements.
- It ensures that the candidate system provides appropriate responses to what extent it can support the technical enhancement.

Operational Feasibility

- It determines whether the system is operating effectively once it is developed and implemented.
- It ensures that the management should support the proposed system and its working feasible in the current organizational environment.
- It analyzes whether the users will be affected and they accept the modified or new business methods that affect the possible system benefits.
- It also ensures that the computer resources and network architecture of candidate system are workable.

Behavioral Feasibility

- It evaluates and estimates the user attitude or behavior towards the development of new system.

- It helps in determining if the system requires special effort to educate, retrain, transfer, and changes in employee's job status on new ways of conducting business.

Schedule Feasibility

- It ensures that the project should be completed within given time constraint or schedule.
- It also verifies and validates whether the deadlines of project are reasonable or not.

Structured Analysis

Analysts use various tools to understand and describe the information system. One of the ways is using structured analysis.

What is Structured Analysis?

Structured Analysis is a development method that allows the analyst to understand the system and its activities in a logical way.

It is a systematic approach, which uses graphical tools that analyze and refine the objectives of an existing system and develop a new system specification which can be easily understandable by user.

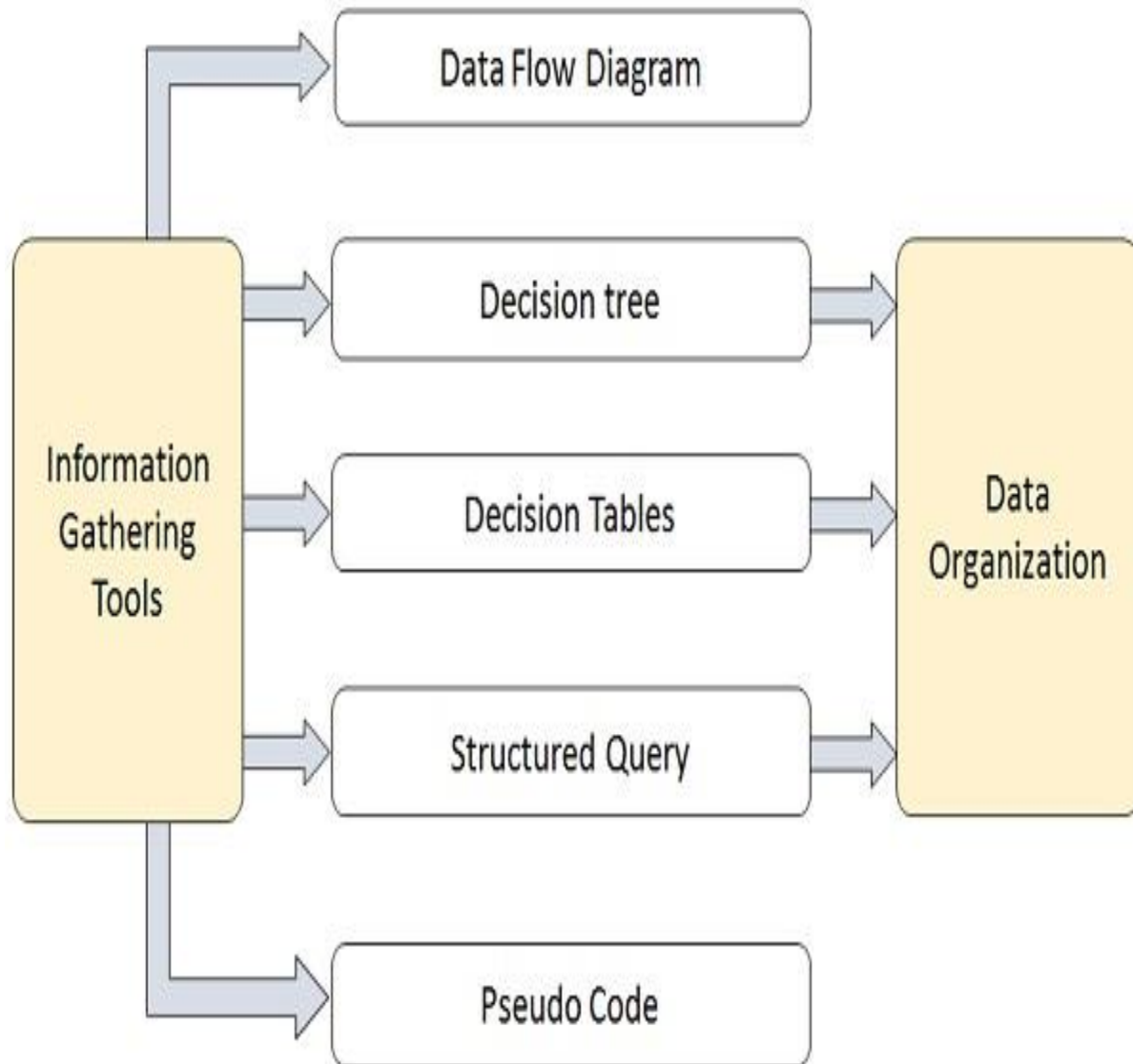
It has following attributes –

- It is graphic which specifies the presentation of application.
- It divides the processes so that it gives a clear picture of system flow.
- It is logical rather than physical i.e., the elements of system do not depend on vendor or hardware.
- It is an approach that works from high-level overviews to lower-level details.

Structured Analysis Tools

During Structured Analysis, various tools and techniques are used for system development. They are –

- Data Flow Diagrams
- Data Dictionary
- Decision Trees
- Decision Tables
- Structured English
- Pseudocode



Data Flow Diagrams (DFD) or Bubble Chart

It is a technique developed by Larry Constantine to express the requirements of system in a graphical form.

- It shows the flow of data between various functions of system and specifies how the current system is implemented.
- It is an initial stage of design phase that functionally divides the requirement specifications down to the lowest level of detail.
- Its graphical nature makes it a good communication tool between user and analyst or analyst and system designer.





- It gives an overview of what data a system processes, what transformations are performed, what data are stored, what results are produced and where they flow.

Basic Elements of DFD

DFD is easy to understand and quite effective when the required design is not clear and the user wants a notational language for communication. However, it requires a large number of iterations for obtaining the most accurate and complete solution.

The following table shows the symbols used in designing a DFD and their significance

–

Symbol Name	Symbol	Meaning
Square		Source or Destination of Data
Arrow		Data flow
Circle		Process transforming data flow
Open Rectangle		Data Store

Types of DFD

DFDs are of two types: Physical DFD and Logical DFD. The following table lists the points that differentiate a physical DFD from a logical DFD.

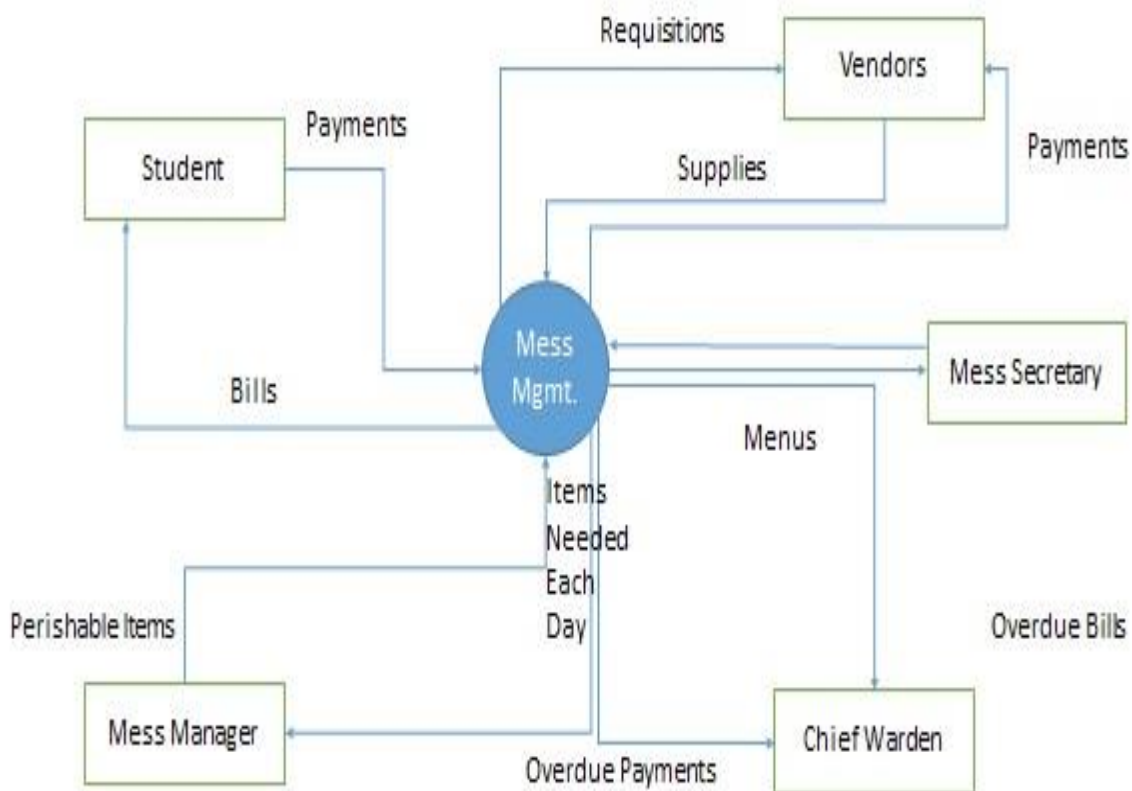
Physical DFD	Logical DFD
It is implementation dependent. It shows which functions are performed.	It is implementation independent. It focuses only on the flow of data between processes.

It provides low level details of hardware, software, files, and people.	It explains events of systems and data required by each event.
It depicts how the current system operates and how a system will be implemented.	It shows how business operates; not how the system can be implemented.

Context Diagram

A context diagram helps in understanding the entire system by one DFD which gives the overview of a system. It starts with mentioning major processes with little details and then goes onto giving more details of the processes with the top-down approach.

The context diagram of mess management is shown below.



Data Dictionary

A data dictionary is a structured repository of data elements in the system. It stores the descriptions of all DFD data elements that is, details and definitions of data flows, data stores, data stored in data stores, and the processes.

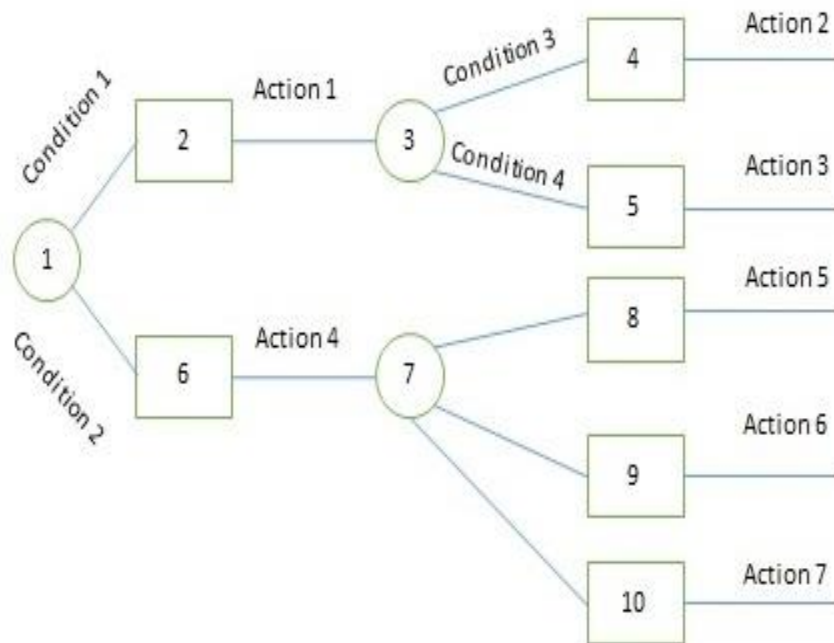
A data dictionary improves the communication between the analyst and the user. It plays an important role in building a database. Most DBMSs have a data dictionary as a standard feature. For example, refer the following table –

Sr.No.	Data Name	Description	No. of Characters
1	ISBN	ISBN Number	10
2	TITLE	title	60
3	SUB	Book Subjects	80
4	ANAME	Author Name	15

Decision Trees

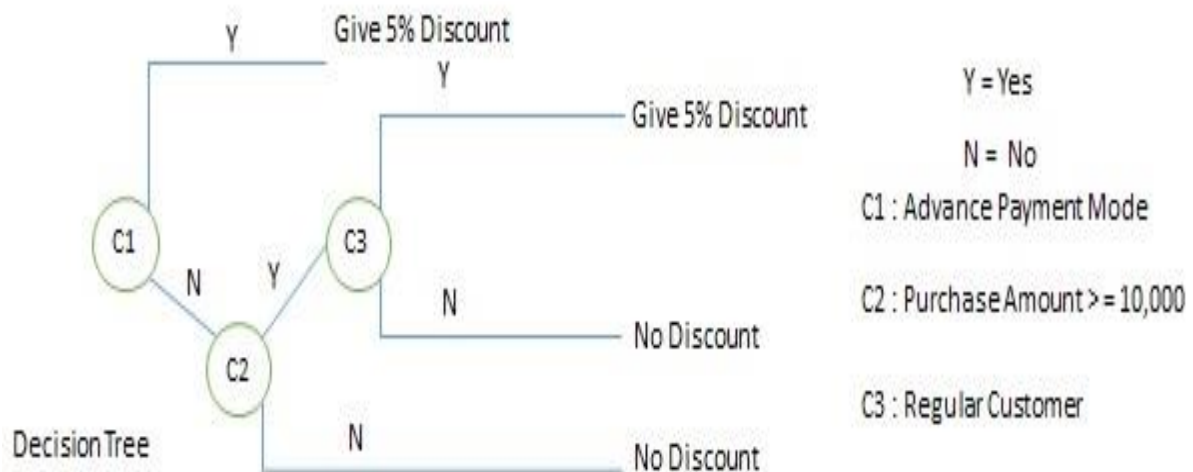
Decision trees are a method for defining complex relationships by describing decisions and avoiding the problems in communication. A decision tree is a diagram that shows alternative actions and conditions within horizontal tree framework. Thus, it depicts which conditions to consider first, second, and so on.

Decision trees depict the relationship of each condition and their permissible actions. A square node indicates an action and a circle indicates a condition. It forces analysts to consider the sequence of decisions and identifies the actual decision that must be made.



The major limitation of a decision tree is that it lacks information in its format to describe what other combinations of conditions you can take for testing. It is a single representation of the relationships between conditions and actions.

For example, refer the following decision tree –



Decision Tables

Decision tables are a method of describing the complex logical relationship in a precise manner which is easily understandable.

- It is useful in situations where the resulting actions depend on the occurrence of one or several combinations of independent conditions.
- It is a matrix containing row or columns for defining a problem and the actions.

Components of a Decision Table

- **Condition Stub** – It is in the upper left quadrant which lists all the condition to be checked.
- **Action Stub** – It is in the lower left quadrant which outlines all the action to be carried out to meet such condition.
- **Condition Entry** – It is in upper right quadrant which provides answers to questions asked in condition stub quadrant.
- **Action Entry** – It is in lower right quadrant which indicates the appropriate action resulting from the answers to the conditions in the condition entry quadrant.

The entries in decision table are given by Decision Rules which define the relationships between combinations of conditions and courses of action. In rules section,

- Y shows the existence of a condition.
- N represents the condition, which is not satisfied.
- A blank - against action states it is to be ignored.
- X (or a check mark will do) against action states it is to be carried out.

For example, refer the following table –

CONDITIONS	Rule 1	Rule 2	Rule 3	Rule 4
Advance payment made	Y	N	N	N
Purchase amount = Rs 10,000/-	-	Y	Y	N

Regular Customer	-	Y	N	-
ACTIONS				
Give 5% discount	X	X	-	-
Give no discount	-	-	X	X

Structured English

Structure English is derived from structured programming language which gives more understandable and precise description of process. It is based on procedural logic that uses construction and imperative sentences designed to perform operation for action.

- It is best used when sequences and loops in a program must be considered and the problem needs sequences of actions with decisions.
- It does not have strict syntax rule. It expresses all logic in terms of sequential decision structures and iterations.

For example, see the following sequence of actions –

```

if customer pays advance
then
  Give 5% Discount
else
  if purchase amount >=10,000
  then
    if the customer is a regular customer
    then Give 5% Discount
    else No Discount
  end if
  else No Discount
end if
end if

```

Pseudocode

A pseudocode does not conform to any programming language and expresses logic in plain English.

- It may specify the physical programming logic without actual coding during and after the physical design.
- It is used in conjunction with structured programming.
- It replaces the flowcharts of a program.

Guidelines for Selecting Appropriate Tools

Use the following guidelines for selecting the most appropriate tool that would suit your requirements –

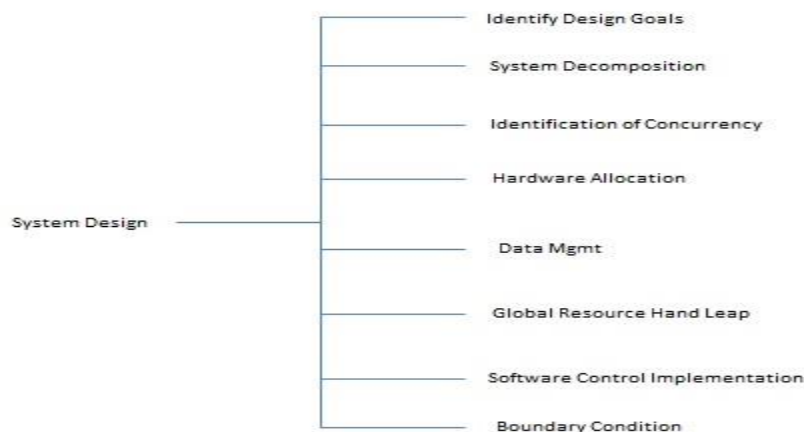
- Use DFD at high or low level analysis for providing good system documentations.
- Use data dictionary to simplify the structure for meeting the data requirement of the system.
- Use structured English if there are many loops and actions are complex.
- Use decision tables when there are a large number of conditions to check and logic is complex.
- Use decision trees when sequencing of conditions is important and if there are few conditions to be tested.

System Analysis & Design - System Design

System design is the phase that bridges the gap between problem domain and the existing system in a manageable way. This phase focuses on the solution domain, i.e. “how to implement?”

It is the phase where the SRS document is converted into a format that can be implemented and decides how the system will operate.

In this phase, the complex activity of system development is divided into several smaller sub-activities, which coordinate with each other to achieve the main objective of system development.



Inputs to System Design

System design takes the following inputs –

- Statement of work
- Requirement determination plan
- Current situation analysis
- Proposed system requirements including a conceptual data model, modified DFDs, and Metadata (data about data).

Outputs for System Design

System design gives the following outputs –

- Infrastructure and organizational changes for the proposed system.
- A data schema, often a relational schema.
- Metadata to define the tables/files and columns/data-items.
- A function hierarchy diagram or web page map that graphically describes the program structure.
- Actual or pseudocode for each module in the program.
- A prototype for the proposed system.

Types of System Design

Logical Design

Logical design pertains to an abstract representation of the data flow, inputs, and outputs of the system. It describes the inputs (sources), outputs (destinations), databases (data stores), procedures (data flows) all in a format that meets the user requirements.

While preparing the logical design of a system, the system analyst specifies the user needs at level of detail that virtually determines the information flow into and out of the system and the required data sources. Data flow diagram, E-R diagram modeling are used.

Physical Design

Physical design relates to the actual input and output processes of the system. It focuses on how data is entered into a system, verified, processed, and displayed as output.

It produces the working system by defining the design specification that specifies exactly what the candidate system does. It is concerned with user interface design, process design, and data design.

It consists of the following steps –

- Specifying the input/output media, designing the database, and specifying backup procedures.
- Planning system implementation.
- Devising a test and implementation plan, and specifying any new hardware and software.
- Updating costs, benefits, conversion dates, and system constraints.

Architectural Design

It is also known as high level design that focuses on the design of system architecture. It describes the structure and behavior of the system. It defines the structure and relationship between various modules of system development process.

Detailed Design

It follows Architectural design and focuses on development of each module.

Conceptual Data Modeling

It is representation of organizational data which includes all the major entities and relationship. System analysts develop a conceptual data model for the current system that supports the scope and requirement for the proposed system.

The main aim of conceptual data modeling is to capture as much meaning of data as possible. Most organization today use conceptual data modeling using E-R model which uses special notation to represent as much meaning about data as possible.

Entity Relationship Model







It is a technique used in database design that helps describe the relationship between various entities of an organization.




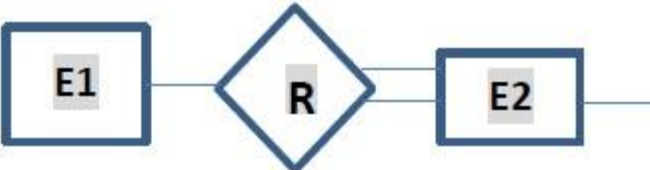
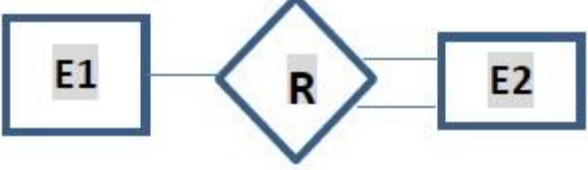
Terms used in E-R model

- **ENTITY** – It specifies distinct real world items in an application. For example: vendor, item, student, course, teachers, etc.
- **RELATIONSHIP** – They are the meaningful dependencies between entities. For example, vendor supplies items, teacher teaches courses, then supplies and course are relationship.

- **ATTRIBUTES** – It specifies the properties of relationships. For example, vendor code, student name. Symbols used in E-R model and their respective meanings –

The following table shows the symbols used in E-R model and their significance –

Symbol	Meaning
	Entity
	Weak Entity
	Relationship
	Identity Relationship
	Attributes
	Key Attributes

	Multivalued
	Composite Attribute
	Derived Attributes
	Total Participation of E2 in R
	Cardinality Ratio 1:N for E1:E2 in R

Three types of relationships can exist between two sets of data: one-to-one, one-to-many, and many-to-many.

File Organization

It describes how records are stored within a file.

There are four file organization methods –

- **Serial** – Records are stored in chronological order (in order as they are input or occur). **Examples** – Recording of telephone charges, ATM transactions, Telephone queues.
- **Sequential** – Records are stored in order based on a key field which contains a value that uniquely identifies a record. **Examples** – Phone directories.

- **Direct (relative)** – Each record is stored based on a physical address or location on the device. Address is calculated from the value stored in the record’s key field. Randomizing routine or hashing algorithm does the conversion.
- **Indexed** – Records can be processed both sequentially and non-sequentially using indexes.

Comparison

	Serial	Sequential	Direct	Index
Type of Access	Batch	Batch	Online	Batch or Online
Data Organization	Serial	Sequentially by key value	No particular order	Sequentially and by index
Flexibility in handling inquiries	No	No	Yes	Yes
Availability of up to date Data	No	No	Yes	Yes
Speed Retrieval	Slow	Slow	Very Fast	Fast
Activity	High	High	Low	High
Volatility	Low	Low	High	High
Example	ATM Transition Queue	Payroll process script billing operation	Online reservation and banking transaction	Customer ordering and billing

File Access

One can access a file using either Sequential Access or Random Access. File Access methods allow computer programs read or write records in a file.

Sequential Access

Every record on the file is processed starting with the first record until End of File (EOF) is reached. It is efficient when a large number of the records on the file need to

be accessed at any given time. Data stored on a tape (sequential access) can be accessed only sequentially.

Direct (Random) Access

Records are located by knowing their physical locations or addresses on the device rather than their positions relative to other records. Data stored on a CD device (direct-access) can be accessed either sequentially or randomly.

Types of Files used in an Organization System

Following are the types of files used in an organization system –

- **Master file** – It contains the current information for a system. For example, customer file, student file, telephone directory.
- **Table file** – It is a type of master file that changes infrequently and stored in a tabular format. For example, storing Zipcode.
- **Transaction file** – It contains the day-to-day information generated from business activities. It is used to update or process the master file. For example, Addresses of the employees.
- **Temporary file** – It is created and used whenever needed by a system.
- **Mirror file** – They are the exact duplicates of other files. Help minimize the risk of downtime in cases when the original becomes unusable. They must be modified each time the original file is changed.
- **Log files** – They contain copies of master and transaction records in order to chronicle any changes that are made to the master file. It facilitates auditing and provides mechanism for recovery in case of system failure.
- **Archive files** – Backup files that contain historical versions of other files.

Documentation Control

Documentation is a process of recording the information for any reference or operational purpose. It helps users, managers, and IT staff, who require it. It is important that prepared document must be updated on regular basis to trace the progress of the system easily.

After the implementation of system if the system is working improperly, then documentation helps the administrator to understand the flow of data in the system to correct the flaws and get the system working.

Programmers or systems analysts usually create program and system documentation. Systems analysts usually are responsible for preparing documentation to help users learn the system. In large companies, a technical support team that includes technical writers might assist in the preparation of user documentation and training materials.

Advantages

- It can reduce system downtime, cut costs, and speed up maintenance tasks.
- It provides the clear description of formal flow of present system and helps to understand the type of input data and how the output can be produced.
- It provides effective and efficient way of communication between technical and nontechnical users about system.
- It facilitates the training of new user so that he can easily understand the flow of system.
- It helps the user to solve the problems such as troubleshooting and helps the manager to take better final decisions of the organization system.
- It provides better control to the internal or external working of the system.

Types of Documentations

When it comes to System Design, there are following four main documentations –

- Program documentation
- System documentation
- Operations documentation
- User documentation

Program Documentation

- It describes inputs, outputs, and processing logic for all the program modules.
- The program documentation process starts in the system analysis phase and continues during implementation.
- This documentation guides programmers, who construct modules that are well supported by internal and external comments and descriptions that can be understood and maintained easily.

Operations Documentation

Operations documentation contains all the information needed for processing and distributing online and printed output. Operations documentation should be clear, concise, and available online if possible.

It includes the following information –

- Program, systems analyst, programmer, and system identification.

- Scheduling information for printed output, such as report, execution frequency, and deadlines.
- Input files, their source, output files, and their destinations.
- E-mail and report distribution lists.
- Special forms required, including online forms.
- Error and informational messages to operators and restart procedures.
- Special instructions, such as security requirements.

User Documentation

It includes instructions and information to the users who will interact with the system. For example, user manuals, help guides, and tutorials. User documentation is valuable in training users and for reference purpose. It must be clear, understandable, and readily accessible to users at all levels.

The users, system owners, analysts, and programmers, all put combined efforts to develop a user's guide.

A user documentation should include –

- A system overview that clearly describes all major system features, capabilities, and limitations.
- Description of source document content, preparation, processing, and, samples.
- Overview of menu and data entry screen options, contents, and processing instructions.
- Examples of reports that are produced regularly or available at the user's request, including samples.
- Security and audit trail information.
- Explanation of responsibility for specific input, output, or processing requirements.
- Procedures for requesting changes and reporting problems.
- Examples of exceptions and error situations.
- Frequently asked questions (FAQs).
- Explanation of how to get help and procedures for updating the user manual.

System Documentation

System documentation serves as the technical specifications for the IS and how the objectives of the IS are accomplished. Users, managers and IS owners need never reference system documentation. System documentation provides the basis for understanding the technical aspects of the IS when modifications are made.

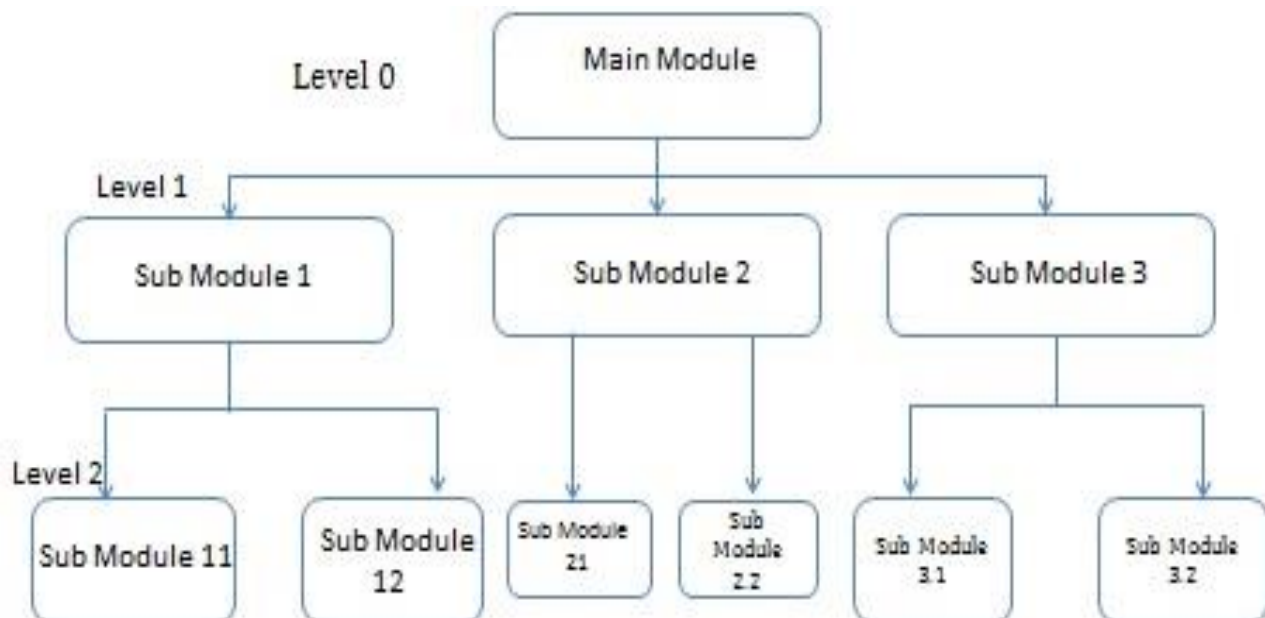
- It describes each program within the IS and the entire IS itself.
- It describes the system's functions, the way they are implemented, each program's purpose within the entire IS with respect to the order of execution, information passed to and from programs, and overall system flow.
- It includes data dictionary entries, data flow diagrams, object models, screen layouts, source documents, and the systems request that initiated the project.
- Most of the system documentation is prepared during the system analysis and system design phases.
- During systems implementation, an analyst must review system documentation to verify that it is complete, accurate, and up-to-date, and including any changes made during the implementation process.

Design Strategies

Top-Down Strategy

The top-down strategy uses the modular approach to develop the design of a system. It is called so because it starts from the top or the highest-level module and moves towards the lowest level modules.

In this technique, the highest-level module or main module for developing the software is identified. The main module is divided into several smaller and simpler submodules or segments based on the task performed by each module. Then, each submodule is further subdivided into several submodules of next lower level. This process of dividing each module into several submodules continues until the lowest level modules, which cannot be further subdivided, are not identified.

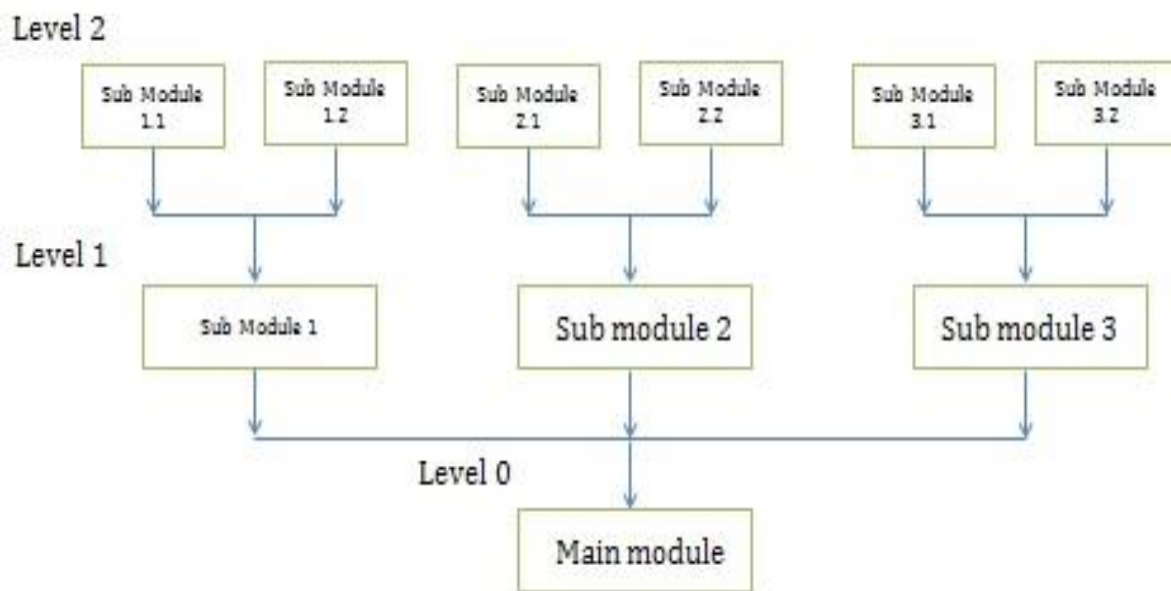


Bottom-Up Strategy

Bottom-Up Strategy follows the modular approach to develop the design of the system. It is called so because it starts from the bottom or the most basic level modules and moves towards the highest level modules.

In this technique,

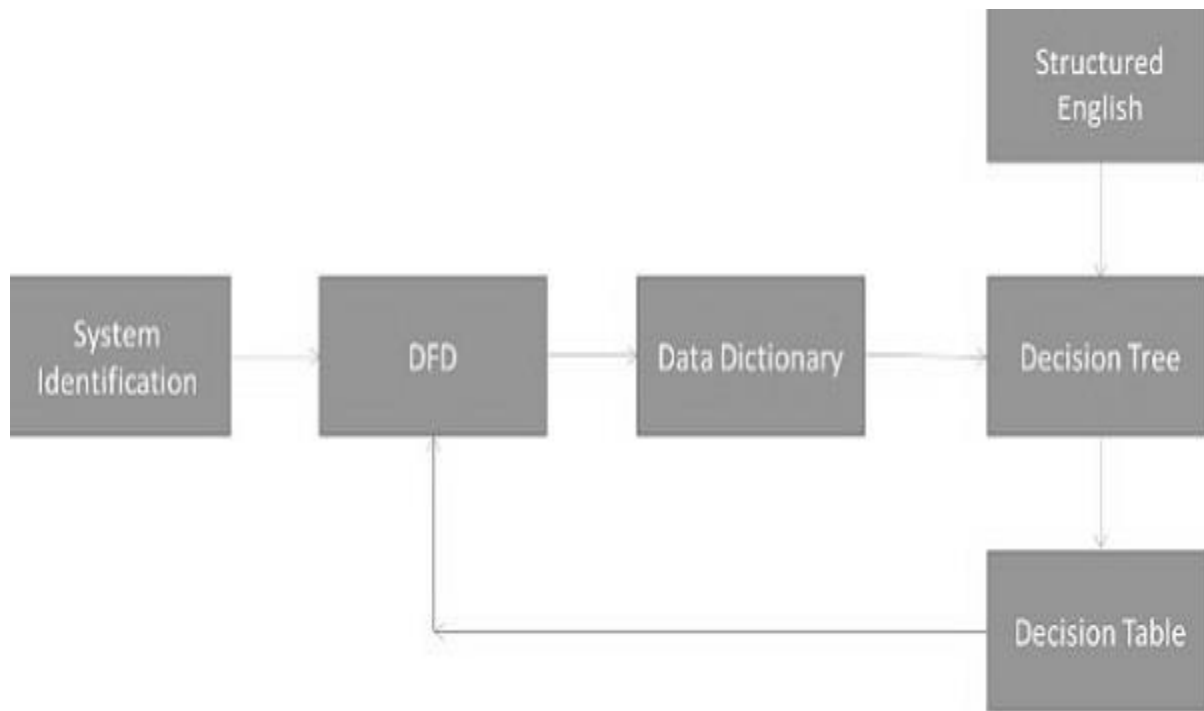
- The modules at the most basic or the lowest level are identified.
- These modules are then grouped together based on the function performed by each module to form the next higher-level modules.
- Then, these modules are further combined to form the next higher-level modules.
- This process of grouping several simpler modules to form higher level modules continues until the main module of system development process is achieved.



Structured Design

Structured design is a data-flow based methodology that helps in identifying the input and output of the developing system. The main objective of structured design is to minimize the complexity and increase the modularity of a program. Structured design also helps in describing the functional aspects of the system.

In structured designing, the system specifications act as a basis for graphically representing the flow of data and sequence of processes involved in a software development with the help of DFDs. After developing the DFDs for the software system, the next step is to develop the structure chart.



Modularization

Structured design partitions the program into small and independent modules. These are organized in top down manner with the details shown in bottom.

Thus, structured design uses an approach called Modularization or decomposition to minimize the complexity and to manage the problem by subdividing it into smaller segments.

Advantages

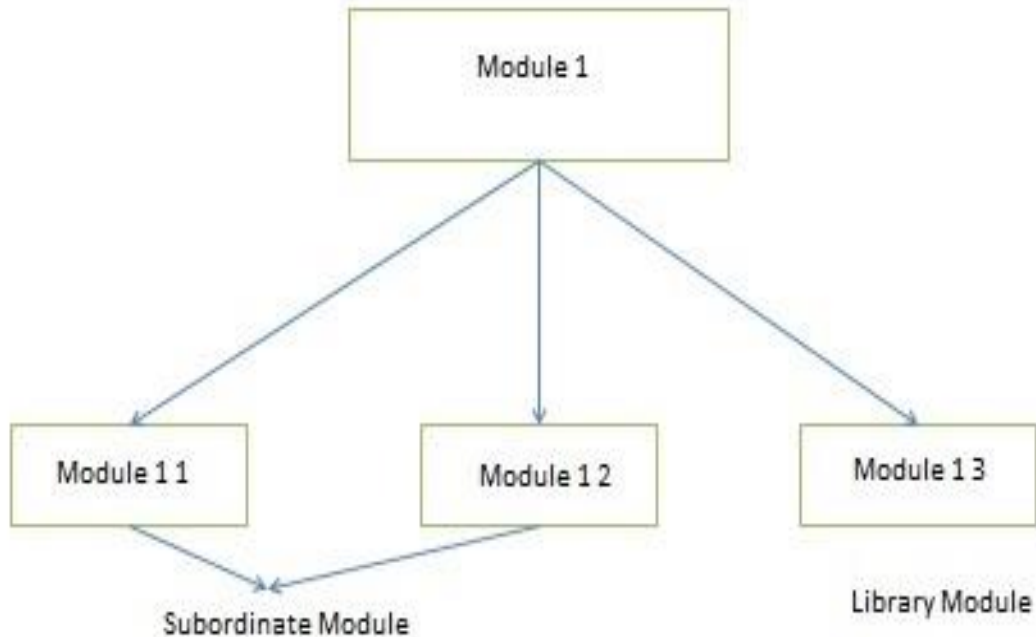
- Critical interfaces are tested first.
- It provide abstraction.
- It allows multiple programmers to work simultaneously.
- It allows code reuse.
- It provides control and improves morale.
- It makes identifying structure easier.

Structured Charts

Structured charts are a recommended tool for designing a modular, top down systems which define the various modules of system development and the relationship between each module. It shows the system module and their relationship between them.

It consists of diagram consisting of rectangular boxes that represent the modules, connecting arrows, or lines.

- **Control Module** – It is a higher-level module that directs lower-level modules, called **subordinate modules**.
- **Library Module** – It is a reusable module and can be invoked from more than one point in the chart.



We have two different approaches to design a structured chart –

- **Transform-Centered Structured Charts** – They are used when all the transactions follow same path.
- **Transaction-Centered Structured Charts** – They are used when all the transactions do not follow the same path.

Objectives of Using Structure Flowcharts

- To encourage a top-down design.
- To support the concept of modules and identify the appropriate modules.
- To show the size and complexity of the system.
- To identify the number of readily identifiable functions and modules within each function.
- To depict whether each identifiable function is a manageable entity or should be broken down into smaller components.

Factors Affecting System Complexity

To develop good quality of system software, it is necessary to develop a good design. Therefore, the main focus on while developing the design of the system is the quality of the software design. A good quality software design is the one, which minimizes the complexity and cost expenditure in software development.

The two important concepts related to the system development that help in determining the complexity of a system are **coupling** and **cohesion**.

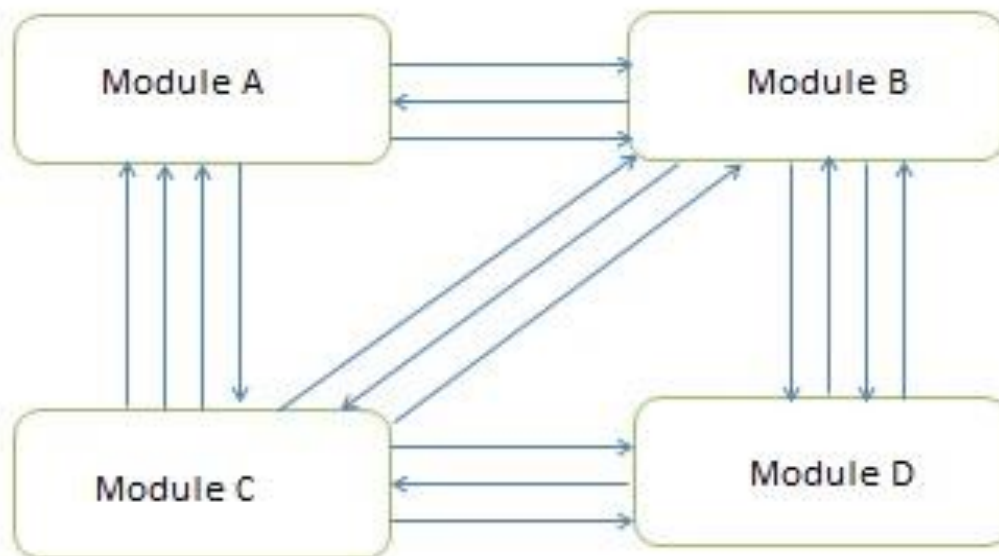
Coupling

Coupling is the measure of the independence of components. It defines the degree of dependency of each module of system development on the other. In practice, this means the stronger the coupling between the modules in a system, the more difficult it is to implement and maintain the system.

Each module should have simple, clean interface with other modules, and that the minimum number of data elements should be shared between modules.

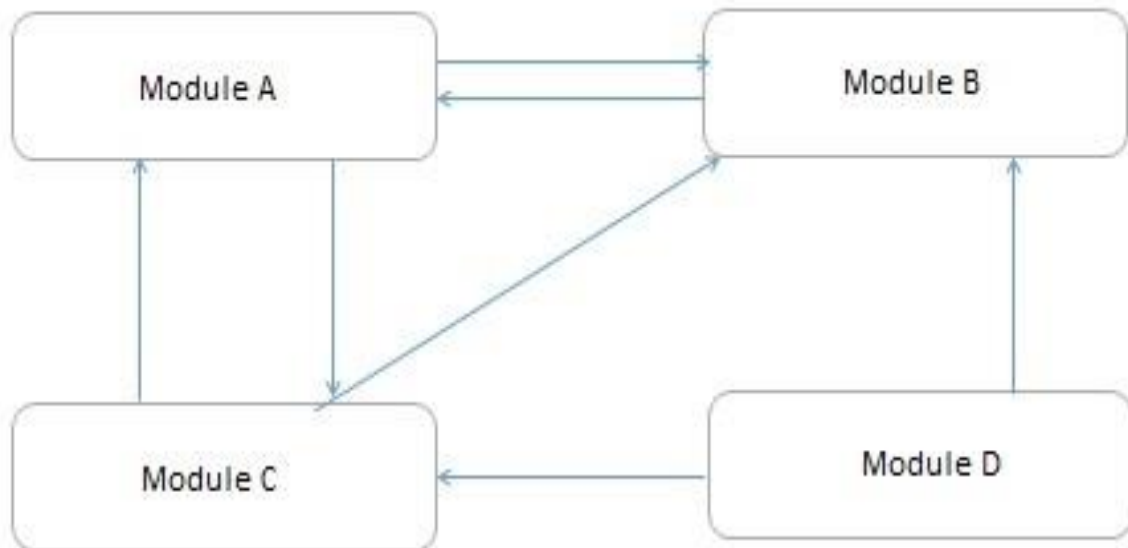
High Coupling

These type of systems have interconnections with program units dependent on each other. Changes to one subsystem leads to high impact on the other subsystem.



Low Coupling

These type of systems are made up of components which are independent or almost independent. A change in one subsystem does not affect any other subsystem.



Coupling Measures

- **Content Coupling** – When one component actually modifies another, then the modified component is completely dependent on modifying one.
- **Common Coupling** – When amount of coupling is reduced somewhat by organizing system design so that data are accessible from a common data store.
- **Control Coupling** – When one component passes parameters to control the activity of another component.
- **Stamp Coupling** – When data structures is used to pass information from one component to another.
- **Data Coupling** – When only data is passed then components are connected by this coupling.

Cohesion

Cohesion is the measure of closeness of the relationship between its components. It defines the amount of dependency of the components of a module on one another. In practice, this means the systems designer must ensure that –

- They do not split essential processes into fragmented modules.
- They do not gather together unrelated processes represented as processes on the DFD into meaningless modules.

The best modules are those that are functionally cohesive. The worst modules are those that are coincidentally cohesive.

The worst degree of cohesion

Coincidental cohesion is found in a component whose parts are unrelated to another.

- **Logical Cohesion** – It is where several logically related functions or data elements are placed in same component.
- **Temporal Cohesion** – It is when a component that is used to initialize a system or set variables performs several functions in sequence, but the functions are related by timing involved.
- **Procedurally Cohesion** – It is when functions are grouped together in a component just to ensure this order.
- **Sequential Cohesion** – It is when the output from one part of a component is the input to the next part of it.

Input / Output & Forms Design

Input Design

In an information system, input is the raw data that is processed to produce output. During the input design, the developers must consider the input devices such as PC, MICR, OMR, etc.

Therefore, the quality of system input determines the quality of system output. Well designed input forms and screens have following properties –

- It should serve specific purpose effectively such as storing, recording, and retrieving the information.
- It ensures proper completion with accuracy.
- It should be easy to fill and straightforward.
- It should focus on user's attention, consistency, and simplicity.
- All these objectives are obtained using the knowledge of basic design principles regarding –

- What are the inputs needed for the system?
- How end users respond to different elements of forms and screens.

Objectives for Input Design

The objectives of input design are –

- To design data entry and input procedures
- To reduce input volume
- To design source documents for data capture or devise other data capture methods
- To design input data records, data entry screens, user interface screens, etc.
- To use validation checks and develop effective input controls.

Data Input Methods

It is important to design appropriate data input methods to prevent errors while entering data. These methods depend on whether the data is entered by customers in forms manually and later entered by data entry operators, or data is directly entered by users on the PCs.

A system should prevent user from making mistakes by –

- Clear form design by leaving enough space for writing legibly.
- Clear instructions to fill form.
- Clear form design.
- Reducing key strokes.
- Immediate error feedback.

Some of the popular data input methods are –

- Batch input method (Offline data input method)
- Online data input method
- Computer readable forms
- Interactive data input

Input Integrity Controls

Input integrity controls include a number of methods to eliminate common input errors by end-users. They also include checks on the value of individual fields; both for format and the completeness of all inputs.

Audit trails for data entry and other system operations are created using transaction logs which gives a record of all changes introduced in the database to provide security and means of recovery in case of any failure.

Output Design

The design of output is the most important task of any system. During output design, developers identify the type of outputs needed, and consider the necessary output controls and prototype report layouts.

Objectives of Output Design

The objectives of input design are –

- To develop output design that serves the intended purpose and eliminates the production of unwanted output.
- To develop the output design that meets the end users requirements.
- To deliver the appropriate quantity of output.
- To form the output in appropriate format and direct it to the right person.
- To make the output available on time for making good decisions.

Let us now go through various types of outputs –

External Outputs

Manufacturers create and design external outputs for printers. External outputs enable the system to leave the trigger actions on the part of their recipients or confirm actions to their recipients.

Some of the external outputs are designed as turnaround outputs, which are implemented as a form and re-enter the system as an input.

Internal outputs

Internal outputs are present inside the system, and used by end-users and managers. They support the management in decision making and reporting.

There are three types of reports produced by management information –

- **Detailed Reports** – They contain present information which has almost no filtering or restriction generated to assist management planning and control.

- **Summary Reports** – They contain trends and potential problems which are categorized and summarized that are generated for managers who do not want details.
- **Exception Reports** – They contain exceptions, filtered data to some condition or standard before presenting it to the manager, as information.

Output Integrity Controls

Output integrity controls include routing codes to identify the receiving system, and verification messages to confirm successful receipt of messages that are handled by network protocol.

Printed or screen-format reports should include a date/time for report printing and the data. Multipage reports contain report title or description, and pagination. Pre-printed forms usually include a version number and effective date.

Forms Design

Both forms and reports are the product of input and output design and are business document consisting of specified data. The main difference is that forms provide fields for data input but reports are purely used for reading. For example, order forms, employment and credit application, etc.

- During form designing, the designers should know –
 - who will use them
 - where would they be delivered
 - the purpose of the form or report
- During form design, automated design tools enhance the developer's ability to prototype forms and reports and present them to end users for evaluation.

Objectives of Good Form Design

A good form design is necessary to ensure the following –

- To keep the screen simple by giving proper sequence, information, and clear captions.
- To meet the intended purpose by using appropriate forms.
- To ensure the completion of form with accuracy.
- To keep the forms attractive by using icons, inverse video, or blinking cursors etc.
- To facilitate navigation.

Types of Forms

Flat Forms

- It is a single copy form prepared manually or by a machine and printed on a paper. For additional copies of the original, carbon papers are inserted between copies.
- It is a simplest and inexpensive form to design, print, and reproduce, which uses less volume.

Unit Set/Snap out Forms

- These are papers with one-time carbons interleaved into unit sets for either handwritten or machine use.
- Carbons may be either blue or black, standard grade medium intensity. Generally, blue carbons are best for handwritten forms while black carbons are best for machine use.

Continuous strip/Fanfold Forms

- These are multiple unit forms joined in a continuous strip with perforations between each pair of forms.
- It is a less expensive method for large volume use.

No Carbon Required (NCR) Paper

- They use carbonless papers which have two chemical coatings (capsules), one on the face and the other on the back of a sheet of paper.
- When pressure is applied, the two capsules interact and create an image.

Testing and Quality Assurance

The software system needs to be checked for its intended behavior and direction of progress at each development stage to avoid duplication of efforts, time and cost overruns, and to assure completion of the system within stipulated time. The software system needs to be checked for its intended behavior and direction of progress at each development stage to avoid duplication of efforts, time and cost overruns, and to assure completion of the system within stipulated time.

System testing and quality assurance come to aid for checking the system. It includes

–

- Product level quality (Testing)
- Process level quality.

Let us go through them briefly –

Testing

Testing is the process or activity that checks the functionality and correctness of software according to specified user requirements in order to improve the quality and reliability of system. It is an expensive, time consuming, and critical approach in system development which requires proper planning of overall testing process.

A successful test is one that finds the errors. It executes the program with explicit intention of finding error, i.e., making the program fail. It is a process of evaluating system with an intention of creating a strong system and mainly focuses on the weak areas of the system or software.

Characteristics of System Testing

System testing begins at the module level and proceeds towards the integration of the entire software system. Different testing techniques are used at different times while testing the system. It is conducted by the developer for small projects and by independent testing groups for large projects.

Stages of System Testing

The following stages are involved in testing –

Test Strategy

It is a statement that provides information about the various levels, methods, tools, and techniques used for testing the system. It should satisfy all the needs of an organization.

Test Plan

It provides a plan for testing the system and verifies that the system under testing fulfils all the design and functional specifications. The test plan provides the following information –

- Objectives of each test phase
- Approaches and tools used for testing
- Responsibilities and time required for each testing activity
- Availability of tools, facilities, and test libraries
- Procedures and standards required for planning and conducting the tests
- Factors responsible for successful completion of testing process

Test Case Design

- A number of test cases are identified for each module of the system to be tested.
- Each test case will specify how the implementation of a particular requirement or design decision is to be tested and the criteria for the success of the test.
- The test cases along with the test plan are documented as a part of a system specification document or in a separate document called test specification or test description.

Test Procedures

It consists of the steps that should be followed to execute each of the test cases. These procedures are specified in a separate document called test procedure specification. This document also specifies any special requirements and formats for reporting the result of testing.

Test Result Documentation

Test result file contains brief information about the total number of test cases executed, the number of errors, and nature of errors. These results are then assessed against criteria in the test specification to determine the overall outcome of the test.

Types of Testing

Testing can be of various types and different types of tests are conducted depending on the kind of bugs one seeks to discover –

Unit Testing

Also known as Program Testing, it is a type of testing where the analyst tests or focuses on each program or module independently. It is carried out with the intention of executing each statement of the module at least once.

- In unit testing, accuracy of program cannot be assured and it is difficult to conduct testing of various input combination in detail.
- It identifies maximum errors in a program as compared to other testing techniques.

Integration Testing

In Integration Testing, the analyst tests multiple module working together. It is used to find discrepancies between the system and its original objective, current specifications, and systems documentation.

- Here the analysts are try to find areas where modules have been designed with different specifications for data length, type, and data element name.

- It verifies that file sizes are adequate and that indices have been built properly.

Functional Testing

Function testing determines whether the system is functioning correctly according to its specifications and relevant standards documentation. Functional testing typically starts with the implementation of the system, which is very critical for the success of the system.

Functional testing is divided into two categories –

- **Positive Functional Testing** – It involves testing the system with valid inputs to verify that the outputs produced are correct.
- **Negative Functional Testing** – It involves testing the software with invalid inputs and undesired operating conditions.

Rules for System Testing

To carry out system testing successfully, you need to follow the given rules –

- Testing should be based on the requirements of user.
- Before writing testing scripts, understand the business logic should be understood thoroughly.
- Test plan should be done as soon as possible.
- Testing should be done by the third party.
- It should be performed on static software.
- Testing should be done for valid and invalid input conditions.
- Testing should be reviewed and examined to reduce the costs.
- Both static and dynamic testing should be conducted on the software.
- Documentation of test cases and test results should be done.

Quality Assurance

It is the review of system or software products and its documentation for assurance that system meets the requirements and specifications.

- Purpose of QA is to provide confidence to the customers by constant delivery of product according to specification.
- Software quality Assurance (SQA) is a techniques that includes procedures and tools applied by the software professionals to ensure that software meet the specified standard for its intended use and performance.
- The main aim of SQA is to provide proper and accurate visibility of software project and its developed product to the administration.

- It reviews and audits the software product and its activities throughout the life cycle of system development.

Objectives of Quality Assurance

The objectives of conducting quality assurance are as follows –

- To monitor the software development process and the final software developed.
- To ensure whether the software project is implementing the standards and procedures set by the management.
- To notify groups and individuals about the SQA activities and results of these activities.
- To ensure that the issues, which are not solved within the software are addressed by the upper management.
- To identify deficiencies in the product, process, or the standards, and fix them.

Levels of Quality Assurance

There are several levels of QA and testing that need to be performed in order to certify a software product.

Level 1 – Code Walk-through

At this level, offline software is examined or checked for any violations of the official coding rules. In general, the emphasis is placed on examination of the documentation and level of in-code comments.

Level 2 – Compilation and Linking

At this level, it is checked that the software can compile and link all official platforms and operating systems.

Level 3 – Routine Running

At this level, it is checked that the software can run properly under a variety of conditions such as certain number of events and small and large event sizes etc.

Level 4 – Performance test

At this final level, it is checked that the performance of the software satisfies the previously specified performance level.

System Implementation and Maintenance

Implementation is a process of ensuring that the information system is operational. It involves –

- Constructing a new system from scratch

- Constructing a new system from the existing one.

Implementation allows the users to take over its operation for use and evaluation. It involves training the users to handle the system and plan for a smooth conversion.

Training

The personnel in the system must know in detail what their roles will be, how they can use the system, and what the system will or will not do. The success or failure of well-designed and technically elegant systems can depend on the way they are operated and used.

Training Systems Operators

Systems operators must be trained properly such that they can handle all possible operations, both routine and extraordinary. The operators should be trained in what common malfunctions may occur, how to recognize them, and what steps to take when they come.

Training involves creating troubleshooting lists to identify possible problems and remedies for them, as well as the names and telephone numbers of individuals to contact when unexpected or unusual problems arise.

Training also involves familiarization with run procedures, which involves working through the sequence of activities needed to use a new system.

User Training

- End-user training is an important part of the computer-based information system development, which must be provided to employees to enable them to do their own problem solving.
- User training involves how to operate the equipment, troubleshooting the system problem, determining whether a problem that arose is caused by the equipment or software.
- Most user training deals with the operation of the system itself. The training courses must be designed to help the user with fast mobilization for the organization.

Training Guidelines

- Establishing measurable objectives
- Using appropriate training methods
- Selecting suitable training sites

- Employing understandable training materials

Training Methods

Instructor-led training

It involves both trainers and trainees, who have to meet at the same time, but not necessarily at the same place. The training session could be one-on-one or collaborative. It is of two types –

Virtual Classroom

In this training, trainers must meet the trainees at the same time, but are not required to be at the same place. The primary tools used here are: video conferencing, text based Internet relay chat tools, or virtual reality packages, etc.

Normal Classroom

The trainers must meet the trainees at the same time and at the same place. Their primary tools used here are blackboard, overhead projectors, LCD projector, etc.

Self-Paced Training

It involves both trainers and trainees, who do not need to meet at the same place or at the same time. The trainees learn the skills themselves by accessing the courses at their own convenience. It is of two types –

Multimedia Training

In this training, courses are presented in multimedia format and stored on CD-ROM. It minimizes the cost in developing an in-house training course without assistance from external programmers.

Web-based Training

In this training, courses are often presented in hyper media format and developed to support internet and intranet. It provides just-in-time training for end users and allow organization to tailor training requirements.

Conversion

It is a process of migrating from the old system to the new one. It provides understandable and structured approach to improve the communication between management and project team.

Conversion Plan

It contains description of all the activities that must occur during implementation of the new system and put it into operation. It anticipates possible problems and solutions to deal with them.

It includes the following activities –

- Name all files for conversions.
- Identifying the data requirements to develop new files during conversion.
- Listing all the new documents and procedures that are required.
- Identifying the controls to be used in each activity.
- Identifying the responsibility of person for each activity.
- Verifying conversion schedules.

Conversion Methods

The four methods of conversion are –

- Parallel Conversion
- Direct Cutover Conversion
- Pilot Approach
- Phase-In Method

Method	Description	Advantages	Disadvantages
Parallel Conversion	Old and new systems are used simultaneously.	Provides fallback when new system fails. Offers greatest security and ultimately testing of new system.	Causes cost overruns. New system may not get fair trail.
Direct Cutover Conversion	New system is implemented and old system is replaced completely.	Forces users to make new system work Immediate benefit	No fall back if problems arise with new system Requires most

		from new methods and control.	careful planning
Pilot Approach	Supports phased approach that gradually implement system across all users	Allows training and installation without unnecessary use of resources. Avoid large contingencies from risk management.	A long term phase in causes a problem of whether conversion goes well or not.
Phase-In Method	Working version of system implemented in one part of organization based on feedback, it is installed throughout the organization all alone or stage by stage.	Provides experience and line test before implementation When preferred new system involves new technology or drastic changes in performance.	Gives impression that old system is erroneous and it is not reliable.

File Conversion

It is a process of converting one file format into another. For example, file in WordPerfect format can be converted into Microsoft Word.

For successful conversion, a conversion plan is required, which includes –

- Knowledge of the target system and understanding of the present system
- Teamwork
- Automated methods, testing and parallel operations
- Continuous support for correcting problems
- Updating systems/user documentation, etc

Many popular applications support opening and saving to other file formats of the same type. For example, Microsoft Word can open and save files in many other word processing formats.

Post-Implementation Evaluation Review (PIER)

PIER is a tool or standard approach for evaluating the outcome of the project and determine whether the project is producing the expected benefits to the processes, products or services. It enables the user to verify that the project or system has achieved its desired outcome within specified time period and planned cost.

PIER ensures that the project has met its goals by evaluating the development and management processes of the project.

Objectives of PIER

The objectives of having a PIER are as follows –

- To determine the success of a project against the projected costs, benefits, and timelines.
- To identify the opportunities to add additional value to the project.
- To determine strengths and weaknesses of the project for future reference and appropriate action.
- To make recommendations on the future of the project by refining cost estimating techniques.

The following staff members should be included in the review process –

- Project team and Management
- User staff
- Strategic Management Staff
- External users

System Maintenance / Enhancement

Maintenance means restoring something to its original conditions. Enhancement means adding, modifying the code to support the changes in the user specification. System maintenance conforms the system to its original requirements and enhancement adds to system capability by incorporating new requirements.

Thus, maintenance changes the existing system, enhancement adds features to the existing system, and development replaces the existing system. It is an important part of system development that includes the activities which corrects errors in system design and implementation, updates the documents, and tests the data.

Maintenance Types

System maintenance can be classified into three types –

- **Corrective Maintenance** – Enables user to carry out the repairing and correcting leftover problems.
- **Adaptive Maintenance** – Enables user to replace the functions of the programs.
- **Perfective Maintenance** – Enables user to modify or enhance the programs according to the users' requirements and changing needs.

System Security and Audit

System Audit

It is an investigation to review the performance of an operational system. The objectives of conducting a system audit are as follows –

- To compare actual and planned performance.
- To verify that the stated objectives of system are still valid in current environment.
- To evaluate the achievement of stated objectives.
- To ensure the reliability of computer based financial and other information.
- To ensure all records included while processing.
- To ensure protection from frauds.

Audit of Computer System Usage

Data processing auditors audits the usage of computer system in order to control it. The auditor need control data which is obtained by computer system itself.

The System Auditor

The role of auditor begins at the initial stage of system development so that resulting system is secure. It describes an idea of utilization of system that can be recorded which helps in load planning and deciding on hardware and software specifications. It gives an indication of wise use of the computer system and possible misuse of the system.

Audit Trial

An audit trial or audit log is a security record which is comprised of who has accessed a computer system and what operations are performed during a given period of time. Audit trials are used to do detailed tracing of how data on the system has changed.

It provides documentary evidence of various control techniques that a transaction is subject to during its processing. Audit trails do not exist independently. They are carried out as a part of accounting for recovering lost transactions.

Audit Methods

Auditing can be done in two different ways –

Auditing around the Computer

- Take sample inputs and manually apply processing rules.
- Compare outputs with computer outputs.

Auditing through the Computer

- Establish audit trail which allows examining selected intermediate results.
- Control totals provide intermediate checks.

Audit Considerations

Audit considerations examine the results of the analysis by using both the narratives and models to identify the problems caused due to misplaced functions, split processes or functions, broken data flows, missing data, redundant or incomplete processing, and nonaddressed automation opportunities.

The activities under this phase are as follows –

- Identification of the current environment problems
- Identification of problem causes
- Identification of alternative solutions
- Evaluation and feasibility analysis of each solution
- Selection and recommendation of most practical and appropriate solution
- Project cost estimation and cost benefit analysis

Security

System security refers to protecting the system from theft, unauthorized access and modifications, and accidental or unintentional damage. In computerized systems, security involves protecting all the parts of computer system which includes data,

software, and hardware. Systems security includes system privacy and system integrity.

- **System privacy** deals with protecting individuals systems from being accessed and used without the permission/knowledge of the concerned individuals.
- **System integrity** is concerned with the quality and reliability of raw as well as processed data in the system.

Control Measures

There are variety of control measures which can be broadly classified as follows –

Backup

- Regular backup of databases daily/weekly depending on the time criticality and size.
- Incremental back up at shorter intervals.
- Backup copies kept in safe remote location particularly necessary for disaster recovery.
- Duplicate systems run and all transactions mirrored if it is a very critical system and cannot tolerate any disruption before storing in disk.

Physical Access Control to Facilities

- Physical locks and Biometric authentication. For example, finger print
- ID cards or entry passes being checked by security staff.
- Identification of all persons who read or modify data and logging it in a file.

Using Logical or Software Control

- Password system.
- Encrypting sensitive data/programs.
- Training employees on data care/handling and security.
- Antivirus software and Firewall protection while connected to internet.

Risk Analysis

A risk is the possibility of losing something of value. Risk analysis starts with planning for secure system by identifying the vulnerability of system and impact of this. The plan is then made to manage the risk and cope with disaster. It is done to accesses the probability of possible disaster and their cost.

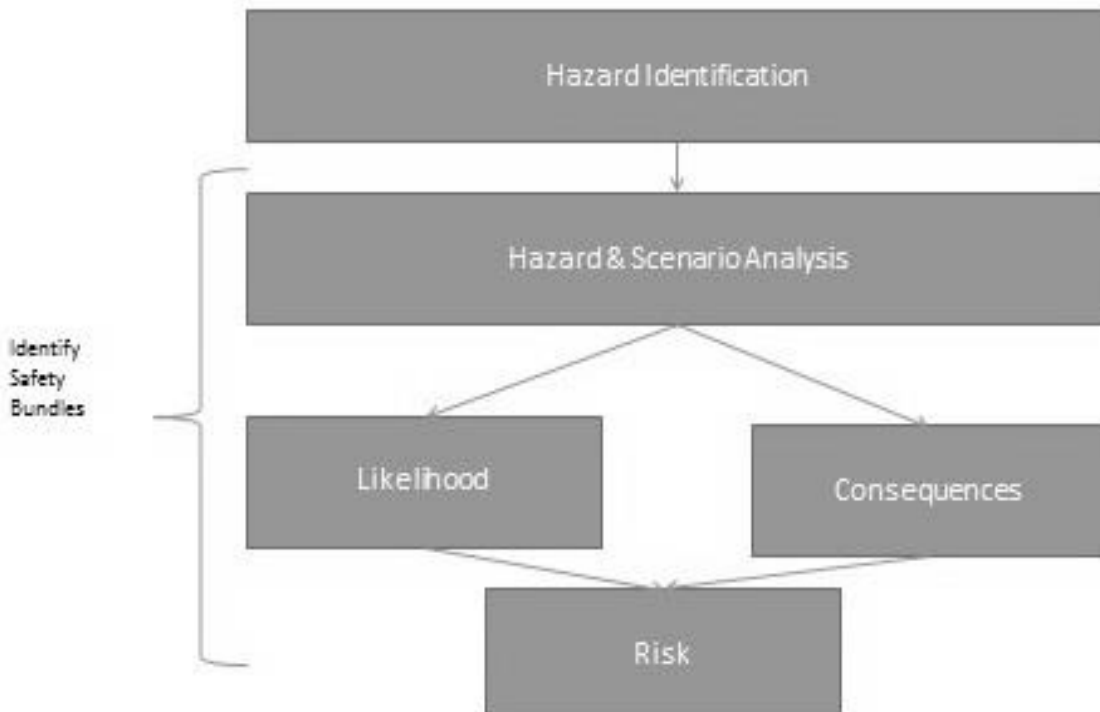
Risk analysis is a teamwork of experts with different backgrounds like chemicals, human error, and process equipment.

The following steps are to be followed while conducting risk analysis –

- Identification of all the components of computer system.
- Identification of all the threats and hazards that each of the components faces.
- Quantify risks i.e. assessment of loss in the case threats become reality.

Risk Analysis – Main Steps

As the risks or threats are changing and the potential loss are also changing, management of risk should be performed on periodic basis by senior managers.



Risk management is a continuous process and it involves the following steps –

- Identification of security measures.
- Calculation of the cost of implementation of security measures.
- Comparison of the cost of security measures with the loss and probability of threats.
- Selection and implementation of security measures.
- Review of the implementation of security measures.

Object Oriented Approach

In the object-oriented approach, the focus is on capturing the structure and behavior of information systems into small modules that combines both data and process. The main aim of Object Oriented Design (OOD) is to improve the quality and productivity of system analysis and design by making it more usable.

In analysis phase, OO models are used to fill the gap between problem and solution. It performs well in situation where systems are undergoing continuous design, adaption, and maintenance. It identifies the objects in problem domain, classifying them in terms of data and behavior.

The OO model is beneficial in the following ways –

- It facilitates changes in the system at low cost.
- It promotes the reuse of components.
- It simplifies the problem of integrating components to configure large system.
- It simplifies the design of distributed systems.

Elements of Object-Oriented System

Let us go through the characteristics of OO System –

- **Objects** – An object is something that exists within problem domain and can be identified by data (attribute) or behavior. All tangible entities (student, patient) and some intangible entities (bank account) are modeled as object.
- **Attributes** – They describe information about the object.
- **Behavior** – It specifies what the object can do. It defines the operation performed on objects.
- **Class** – A class encapsulates the data and its behavior. Objects with similar meaning and purpose grouped together as class.
- **Methods** – Methods determine the behavior of a class. They are nothing more than an action that an object can perform.
- **Message** – A message is a function or procedure call from one object to another. They are information sent to objects to trigger methods. Essentially, a message is a function or procedure call from one object to another.

Features of Object-Oriented System

An object-oriented system comes with several great features which are discussed below.

Encapsulation

Encapsulation is a process of information hiding. It is simply the combination of process and data into a single entity. Data of an object is hidden from the rest of the system and available only through the services of the class. It allows improvement or modification of methods used by objects without affecting other parts of a system.

Abstraction

It is a process of taking or selecting necessary method and attributes to specify the object. It focuses on essential characteristics of an object relative to perspective of user.

Relationships

All the classes in the system are related with each other. The objects do not exist in isolation, they exist in relationship with other objects.

There are three types of object relationships –

- **Aggregation** – It indicates relationship between a whole and its parts.
- **Association** – In this, two classes are related or connected in some way such as one class works with another to perform a task or one class acts upon other class.
- **Generalization** – The child class is based on parent class. It indicates that two classes are similar but have some differences.

Inheritance

Inheritance is a great feature that allows to create sub-classes from an existing class by inheriting the attributes and/or operations of existing classes.

Polymorphism and Dynamic Binding

Polymorphism is the ability to take on many different forms. It applies to both objects and operations. A polymorphic object is one who true type hides within a super or parent class.

In polymorphic operation, the operation may be carried out differently by different classes of objects. It allows us to manipulate objects of different classes by knowing only their common properties.

Structured Approach Vs. Object-Oriented Approach

The following table explains how the object-oriented approach differs from the traditional structured approach –

Structured Approach	Object Oriented Approach
It works with Top-down approach.	It works with Bottom-up approach.
Program is divided into number of submodules or functions.	Program is organized by having number of classes and objects.
Function call is used.	Message passing is used.
Software reuse is not possible.	Reusability is possible.
Structured design programming usually left until end phases.	Object oriented design programming done concurrently with other phases.
Structured Design is more suitable for offshoring.	It is suitable for in-house development.
It shows clear transition from design to implementation.	Not so clear transition from design to implementation.
It is suitable for real time system, embedded system and projects where objects are not the most useful level of abstraction.	It is suitable for most business applications, game development projects, which are expected to customize or extended.
DFD & E-R diagram model the data.	Class diagram, sequence diagram, state chart diagram, and use cases all contribute.
In this, projects can be managed easily due to clearly identifiable phases.	In this approach, projects can be difficult to manage due to uncertain transitions between phase.

Unified Modeling Language (UML)

UML is a visual language that lets you to model processes, software, and systems to express the design of system architecture. It is a standard language for designing and documenting a system in an object oriented manner that allow technical architects to communicate with developer.

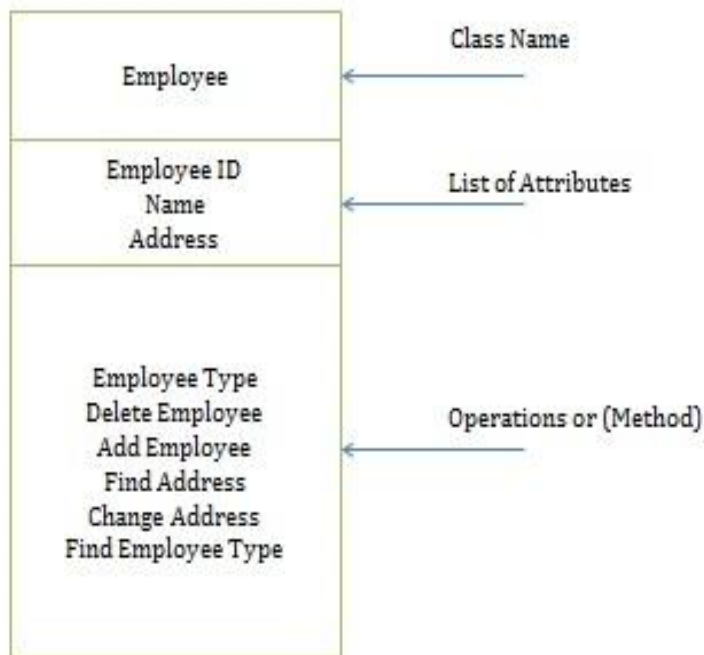
It is defined as set of specifications created and distributed by Object Management Group. UML is extensible and scalable.

The objective of UML is to provide a common vocabulary of object-oriented terms and diagramming techniques that is rich enough to model any systems development project from analysis through implementation.

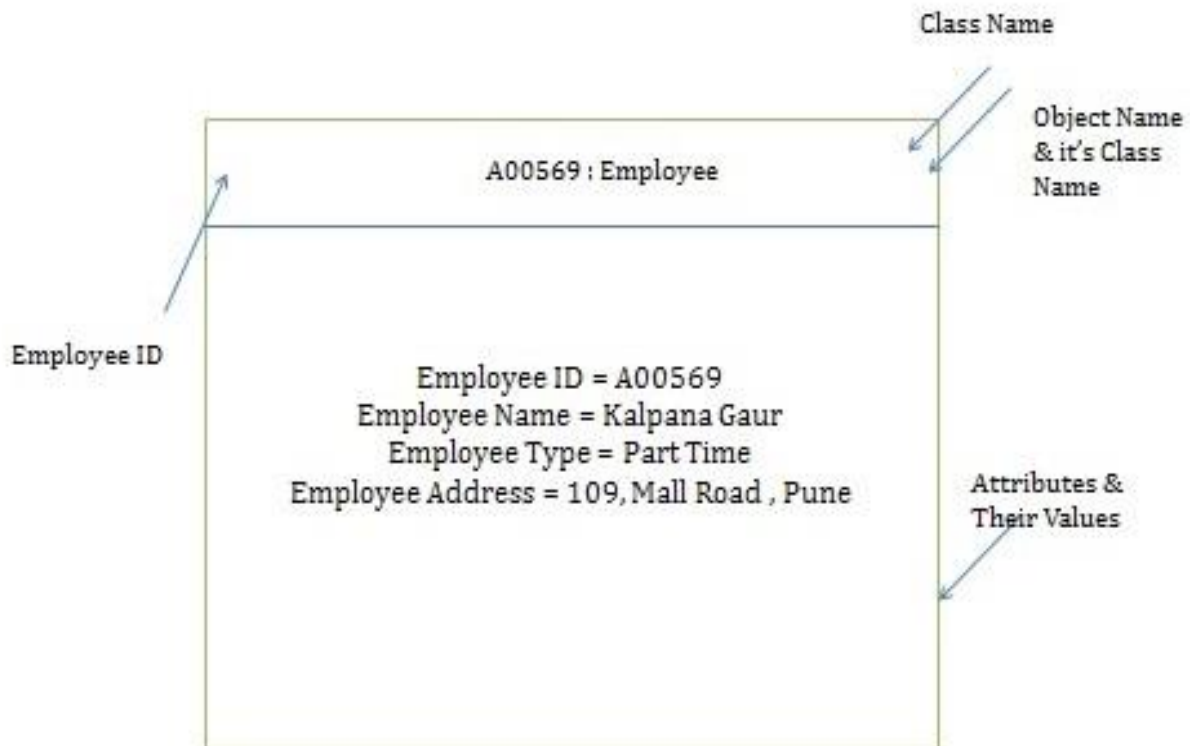
UML is made up of –

- **Diagrams** – It is a pictorial representations of process, system, or some part of it.
- **Notations** – It consists of elements that work together in a diagram such as connectors, symbols, notes, etc.

Example of UML Notation for class



Instance diagram-UML notation



Operations Performed on Objects

The following operations are performed on the objects –

- **Constructor/Destructor** – Creating new instances of a class and deleting existing instances of a class. For example, adding a new employee.
- **Query** – Accessing state without changing value, has no side effects. For example, finding address of a particular employee.
- **Update** – Changes value of one or more attributes & affect state of object For example, changing the address of an employee.

Uses of UML

UML is quite useful for the following purposes –

- Modeling the business process
- Describing the system architecture

- Showing the application structure
- Capturing the system behavior
- Modeling the data structure
- Building the detailed specifications of the system
- Sketching the ideas
- Generating the program code

Static Models

Static models show the structural characteristics of a system, describe its system structure, and emphasize on the parts that make up the system.

- They are used to define class names, attributes, methods, signature, and packages.
- UML diagrams that represent static model include class diagram, object diagram, and use case diagram.

Dynamic Models

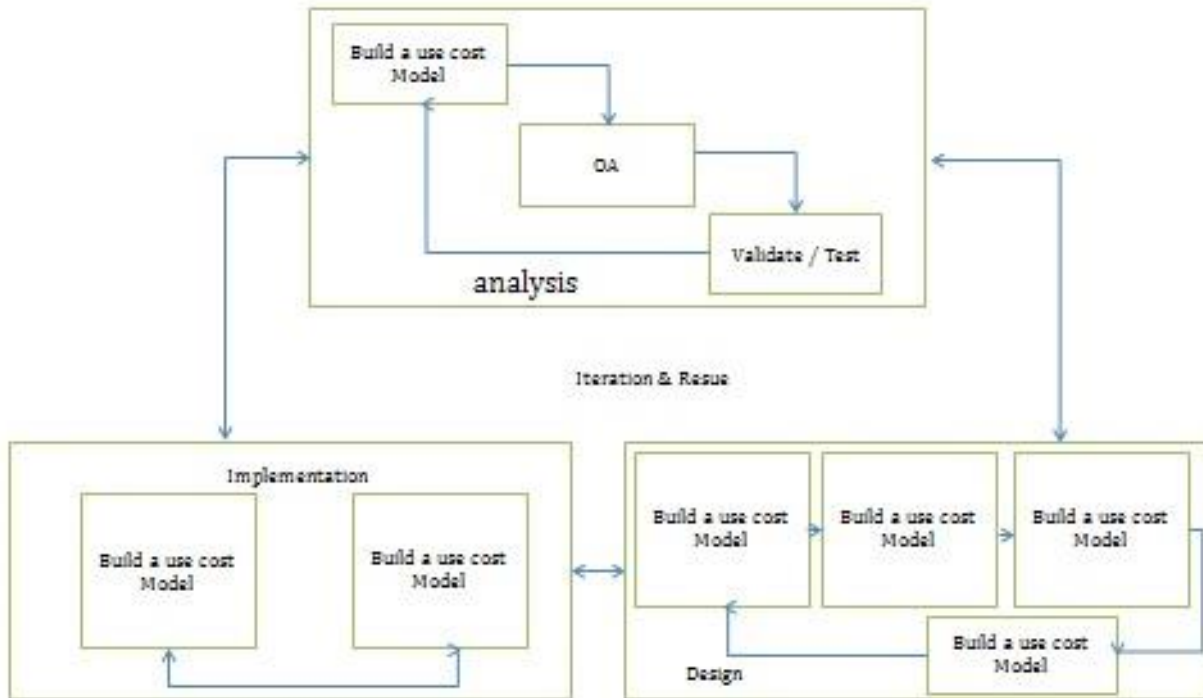
Dynamic models show the behavioral characteristics of a system, i.e., how the system behaves in response to external events.

- Dynamic models identify the object needed and how they work together through methods and messages.
- They are used to design the logic and behavior of system.
- UML diagrams represent dynamic model include sequence diagram, communication diagram, state diagram, activity diagram.

Object Oriented System Development Life Cycle

It consists of three macro processes –

- Object Oriented Analysis (OOA)
- Object oriented design (OOD)
- Object oriented Implementation (OOI)



Object Oriented Systems Development Activities

Object-oriented systems development includes the following stages –

- Object-oriented analysis
- Object-oriented design
- Prototyping
- Implementation
- Incremental testing

Object-Oriented Analysis

This phase concerns with determining the system requirements and to understand the system requirements build a **use-case model**. A use-case is a scenario to describe the interaction between user and computer system. This model represents the user needs or user view of system.

It also includes identifying the classes and their relationships to the other classes in the problem domain, that make up an application.

Object-Oriented Design

The objective of this phase is to design and refine the classes, attributes, methods, and structures that are identified during the analysis phase, user interface, and data access. This phase also identifies and defines the additional classes or objects that support implementation of the requirement.

Prototyping

Prototyping enables to fully understand how easy or difficult it will be to implement some of the features of the system.

It can also give users a chance to comment on the usability and usefulness of the design. It can further define a use-case and make use-case modeling much easier.

Implementation

It uses either Component-Based Development (CBD) or Rapid Application Development (RAD).

Component-based development (CBD)

CODD is an industrialized approach to the software development process using various range of technologies like CASE tools. Application development moves from custom development to assembly of pre-built, pre-tested, reusable software components that operate with each other. A CBD developer can assemble components to construct a complete software system.

Rapid Application Development (RAD)

RAD is a set of tools and techniques that can be used to build an application faster than typically possible with traditional methods. It does not replace SDLC but complements it, since it focuses more on process description and can be combined perfectly with the object oriented approach.

Its task is to build the application quickly and incrementally implement the user requirements design through tools such as visual basic, power builder, etc.

Incremental Testing

Software development and all of its activities including testing are an iterative process. Therefore, it can be a costly affair if we wait to test a product only after its complete development. Here incremental testing comes into picture wherein the product is tested during various stages of its development.

refinement and review

Making customer reviews part of your marketing strategy is essential, especially since 92% of people hesitate to buy a product that doesn't have any. Reviews build trust and factor into SEO, but more importantly, they give you data directly from customers. That feedback provides an opportunity to turn negatives into positives and improve operational inefficiencies.

The holidays are just around the corner and to make the most out of the most wonderful (and lucrative) time of the year, it's crucial that marketers understand the impact of local search. More than three-quarters of location searches on a smartphone result in visit to a physical store within a day, according to Google data.

To determine what gets to the top of those smartphone search engine results pages (SERP), Google takes about 200 ranking factors into account, such as the quality of content, page speed and customer reviews. But reviews provide more than an SEO boost. They also give you data about how to improve your business from the most valuable source: the customers themselves.

Content produced in association with Brandify.

Reviewing your areas for improvement

A review platform can automate the customer feedback process; monitor third-party sites such as Google and Yelp; and aggregate everything in one centralized place. That's important because reviews are even more beneficial for you than they are consumers, 92% of whom will at least hesitate to buy a product if it doesn't have any reviews.

Think of reviews as an advanced form of social listening, a way for you to pull back the curtain and find out exactly what your customers think, in their own words. That feedback is invaluable in identifying and addressing any operational inefficiencies.

One restaurant analyzed reviews across 200 locations, sending those related to food directly to the quality assurance department to make operational enhancements. This helped the brand implement better inventory planning to ensure items did not run out mid-service.

What are the common themes you're seeing when analyzing your reviews? Where can you improve?

Reviews affect brand perception

One challenge customer reviews pose is just how many of them there are; a year and a half ago, Yelp celebrated its hundred millionth review. Simply reading the vast number of reviews across multiple platforms can be overwhelming, without factoring in how time-consuming it is to manage and respond to them.

And responding is a must. Convince and Convert research found that more than half of consumers expect businesses to respond to their negative reviews; 21% expect a response within 24 hours.

With reviews being so important to people, negative ones obviously hurt your brand perception. This is multiplied if they are left without responses. Consumers are reading about other people's bad experiences and on top of that, they're getting the impression that you don't care enough to address those concerns.

Analyzing a problem

Funded projects are usually proposed to address and/or solve identified problems. Problem analysis therefore involves identifying the overriding problem and establishing the causes and effects related to that problem. A key element of this analysis will ensure that "root causes," not just the symptoms of the problem, are identified and subsequently addressed in the project design. Projects that only address the effects of the problem, and not its underlying causes, are unlikely to produce sustainable benefits. One important tool for identifying your project's overriding problem and its root causes is the "problem tree." Click here [\[BROKEN LINK!!\]](#) to see a simplified example of a problem tree.

Some important suggestions for creating problem trees

1. •Involve stakeholders who can contribute relevant technical and local knowledge
2. •Complete several problem tree exercises with different stakeholder groups, to help determine different perspectives and differing priorities
3. •Recognize that the process is as important as the product. The exercise should be presented as a learning experience for all those involved, and as an opportunity for different views and interests to be presented and discussed. However, don't expect from all stakeholders complete agreement about the problems and their relative importance
4. •Recognize that the product (the problem tree diagram) should provide a simplified but nevertheless robust version of reality
5. •Aim for simplicity. If the exercise is too complicated, it is likely to be less useful in providing direction to subsequent steps in the analysis

Before constructing the problem tree

1. •Clarify the scope of the investigation or analysis. If you have not been involved at the very beginning of the project-planning process, understand that others will have already identified (at least to some extent) the major problems the proposed project will address. This understanding will help you focus and structure the direction of the analysis. You will not want, or be able, to deal with a limitless range of problems.
2. Inform yourself further. Collect and review existing background information on the major issue(s) of concern. Are you clear what the major issues are or are likely to be?
3. •Identify the group(s). Determine whom you need to bring together to ensure the group is well informed and can help to analyze and discuss the major issues that the analysis will focus on? For example, if you are addressing a health and sanitation problem that might require a water supply as part of the solution, make sure that you have available to join you a water supply engineer and an environmental health officer (among others). Also, be sure to involve community representatives that you believe would be willing and able to contribute to this kind of exercise. A representative and technically competent group is required to help fully identify, analyze, and organize ideas. Participants need to be informed to be useful and productive. They should know why they are doing the analysis, what the process involves, and what information they are expected to contribute.

Constructing the problem tree

1. Identify and list the main problems

1. Explain the purpose of the exercise and the context within which it is taking place—e.g., preparation of a primary health care project.
2. Explain the problem tree method and the input expected from participants.
3. Provide some examples of the cause and effect relationship before starting, emphasizing the importance of identifying root causes.
4. Using contributions from the group, list all the negative statements about the situation you are analyzing. This can be undertaken as a brainstorming session.
5. Print each problem statement in clear language on a card and display it on some suitable wall space.

2. Identify the overriding problem

1. Through discussions, identify a consensus overriding problem—the one that appears to be linked to most negative statements. Print a precise definition of the overriding problem on a card (if the existing statement requires further clarification). Display the card on a wall (or on the floor) so that the whole group can clearly see it

3. Identify cause and effect

1. Place the negative statement cards in their proper position within the “problem bundle” (causes/problem/effects) according to whether they are “causes”—that is, leading to the overriding problem, or “effects”—that is, resulting from the overriding problem. Continue until all causes are below the overriding problem and all effects are above the overriding problem.
2. Identify constraints. At any stage in the exercise, those statements considered to be unclear or too general should be more clearly specified, discarded, or moved to a category called “constraints.” Constraints are statements that are clear but very general and which not only affect the existing problem bundle but would apply to many other problem bundles. In identifying constraints, ask yourself if the statement is one that can be addressed by a project-based solution. If not, it is a constraint; it is not within the “scope” of your project. Examples of constraints include institutional corruption, lack of government revenue, and high population pressure. If necessary, you will return to the constraints later, when you conduct a risk analysis. To keep the problem tree focused and manageable, move all the constraints to the side of the problem tree.

3. Specify the relationships among causes and effects. Once you are comfortable that you've identified the overriding problem and its causes and effects, specify the relationships among them. Choose any cause or effect and ask, "What causes that?" and "What effect is produced by that?" This process will inevitably result in your identifying additional causes and effects. Now, draw lines between the statements indicating their causal relationships. Draw vertical links to show cause-effect relationships and horizontal links to show joint causes and combined effects.
4. Review. After you have identified the relationships, pause to review, and revise.

4. Check the logic

1. At each stage, invite participants to move the cards—that is, to suggest or hypothesize other relationships. When you have placed all cards, review the structure to ensure that related streams of cause and effect are close to each other on the problem tree.
2. Choose one of the cards at the top line of your problem tree, and work back through the tree by asking, "What leads to or causes that?" in order to check the logic or completeness of your cause-effect structure.
3. By clicking here [\[BROKEN LINK!!\]](#), you can see an example of a completed problem tree involving a project for a U.S. Community College.

creating a software specification document

Software development requirements specify what features the software product should have and what the product's objective is.

How you approach these requirements can make all the difference for the development process and, ultimately, for the end-product as well.

Clearly defining software development requirements matters, because this can:

- **Ensure project consistency:** Defining specific software requirements is the beginning of a software development process and the guarantee of its consistency in later stages. After a prolonged period of development, stakeholders can get confused about what the software should do. Requirements that are well-defined, clear and measurable relate to the business needs and provide clarity and focus to the entire project and everyone involved.

- **Save time and money:**When you define and structure your software requirements, the stage is set for developing the actual product. Knowing in advance as much as possible about what the software needs to do and what features it should have will create positive results quicker and with less expenditure.
- **Provide a base for collaboration:**Teams working on software development often consist of members with very particular and specific knowledge. This especially goes for teams using agile development methodology. Defining software development requirements helps keep them all on the same page. Requirements provide a source of truth and general guidelines for the project by describing all aspects of a product. This makes it easier for every individual to see where their role is in the bigger picture.
- **Provide stability in case of unexpected changes:** Every development process is prone to sudden and unexpected changes: defects in design, test failures, management changes, altered functionality objectives and so on. Change management is important because it can control the rising cost of the project and make sure the delivery of the product is not delayed. Your software development requirements should coordinate and anticipate these possible changes to identify what the possible impact could be.
- **Make sure the entire software project doesn't fail:**Poorly defined or undefined software requirements that are unprioritized, unclear, incomplete or inconsistent jeopardize the entire software development projects.

What Is The Software Requirements Specification Document?

Software Requirements Specification (SRS) document outlines the functions and purpose of the future software product, what it will do and how it will perform.

It is the backbone of the software development project as it lays the foundation and guidelines all parties involved in the project should follow.

The software requirements specification document describes the functionalities the product must have to meet the expectations of its future users.

This document should always include:

- An overall description
- The purpose of the product
- Software's specific requirements

In addition to these, an SRS document needs to establish how the software integrates with the hardware or connects with other software systems.

Outlining SRS document can provide valuable insight such as:

- How to minimize development time and cost
- How and when to make a decision about software product's lifecycle

This document provides essential information about the development projects to various sectors, keeping them on the same page. These sectors include:

- Design
- Development
- QA testing
- Operations
- Maintenance

Even though the terms “software” and “system” are sometimes used interchangeably, there are differences between software requirements specification and system requirement specification.

While software requirements specifications describe the software that will be developed, a system requirements specification document collects information on system requirements.

Before actually defining software requirements in the specification document, there are several things you should establish and understand first.

1. Understand The Software Development Process

The type of software development process depends on the project that needs to be completed and the team that develops it.

The process outlines the steps of the software development lifecycle and every step creates the product that is needed for the next stage in the cycle.

Software development process consists of these six basic stages:

- Gathering of software requirements and analysis of the project
- Product design
- Implementation/Coding
- Testing

- Deployment
- Maintenance

Each subsequent step is dependent on the previous and creates a workflow. Gathered requirements create a basis for the product layout and design. The development phase - Implementation and coding - depends on the design.

The testing process that checks whether the requirements are met either approves or declines the resulting product from the development stage.

If the product meets the requirements, the product is ready to be deployed to the market with subsequent maintenance processes waiting in line.

2. Define The Business Requirements For Your Software Solution

Every software product is created as a response to a certain business need. The procedure of defining and analyzing the software requirements is related to a specific business objective.

The process of defining software's business requirements can help your business determine the scope of the project.

This, in turn, helps with estimating the resources and timeframes needed for its completion.

Knowing the business requirements of a software solution leads to a better understanding of business needs that can be broken down into specific details.

If a problem exists and is identified at the analysis stage, it's much cheaper to fix it then and there rather than when the product is launched.

Follow these steps to define your software solution's business requirements:

- **Identify stakeholders and groups that will benefit from the software product:** These include project sponsors and clients that have the final say on what the project's scope includes. These are also the end-users of the software solution which needs to meet their needs.
- **Capture their requirements:** What do the above groups expect from this software solution? What are their own requirements from the product? Understanding the different perspectives of every stakeholder group helps build a complete picture of what the project should achieve.

- **Categorize their requirements:** Grouping requirements into several categories such as the ones below makes your analysis procedure easier.
 - Functional requirements
 - Operational requirements
 - Technical requirements
 - Transitional requirements
- **Interpret their requirements:** Once their requirements and expectations are collected and categorized, it's important to establish which of them are achievable and how your product can deliver them. You should:
 - Prioritize certain expectations
 - Make sure they are clearly worded, sufficiently detailed, related to business needs and not vague
 - Resolve conflicting issues
 - Analyze feasibility

3. Define Your Preferred Tech Stack And Development Methodology (If Any)

Depending on your software product's goals, development team's size and other factors, you may want to consider several development methodologies that will bring the best results in the given circumstances.

These are the most widely used development methods that you can opt for when developing software.

- **Feature-driven development:** This methodology's goal is delivering the working software frequently and is client-centric. It's a good fit for smaller development teams and is a precursor to agile and lean methodologies.
- **Waterfall:** The traditional way of developing software, this is a plan-driven approach that requires a lot of rigid structure and documentation in advance. In its first stage, it requires a full understanding of the project's demands. Good for large, plan-driven teams that don't sway from their original ideas.
- **Agile:** The opposite of waterfall, agile methodology is flexible and accommodates the possibility of changes during the development process. It values individual team members and their interactions, as well as customer collaboration. Great for teams that collaborate heavily.
- **Scrum:** This methodology adopts agile's notion that team members should collaborate closely and develops software with an iterative approach. Developers break down end goals into smaller goals and work on them using sprints to build software. A useful approach for disciplined smaller teams.

- **Lean:** This method's basic principles are optimization of the whole, elimination of the waste, creating knowledge, delivering fast and deferring commitment. It incorporates manufacturing practices and takes agile methodologies to scale them across the organization and apply them outside of the development job.

How To Define And Document Software Development Requirements In 5 Steps

Once you understand the software development process and have defined the business requirements and development methodology, you are ready to document the software development requirements.

Follow these five steps to create a quality software requirements specification document for the product you mean to build.

1. Make A Software Requirements Specification Outline

The first step in defining the document software development requirements is to create an outline for the SRS.

This outline should include these chapters:

- Product's Purpose
 - Audience
 - Use
 - Scope of the Product
- Product Overview
 - Users' needs
 - Assumptions and Dependencies
- System Requirements and Features
 - System Features
 - Market Requirements
 - Business Requirements
 - UI Requirements
 - Functional Requirements
 - Nonfunctional Requirements

Defining each of these items in your software requirement specifications outline and filling them out means you are ready to move on to the next step.

2. Define The Purpose And Expectations Of The Product

The very first chapter in your SRS documents concerns the product's purpose. It sets the expectations for the software solution that you're building.

- **Audience and use:** In this segment, you need to outline the people in the entire project that will have access to the document and how they should use it. These could be developers, project managers, testers, sales and marketing people or stakeholders in other departments.
- **Scope of the Product:** This segment is for defining the product that you're specifying. It should outline the objectives of the software solution and its benefits.

3. Create An Overview Of A Finished Software Product

The overview or the description of the product part of the SRS should outline the software that you are building.

In order for everyone on the project to know what they're building, you should answer these questions in advance:

- Is the product a new kind of solution?
- It is an update or a take on an existing product?
- Is it an add-on for an already created product?

Answering the above questions help with defining the following:

- **User needs:** Your target audience - the people who will be using your software solution - belongs in this segment. Defining users that need the software product you're building is vital: there are primary and secondary users who will be using the solution regularly and there might be separate buyers whose needs you also need to define.
- **Assumptions and dependencies:** This particular section should outline the factors that could affect the fulfillment of the SRS requirements. It should also include assumptions that STS is making and that could be false. Also, make note of any external factors that the software development project depends on.

4. Get Very Specific About Your Requirements

The development team will make great use of this particular section, because this is where you need to detail the specific requirements for building the software solution.

They consist of functional and nonfunctional requirements, which we will cover in-depth later in the article. There are also:

- **Business requirements:** High-level business goals of the business that is building the software solution.
- **Market requirements:** Requirements that outline the needs of the market and target audiences.
- **External interface requirements:** Types of functional requirements that outline how the product will integrate with other software.
- **User interface requirements:** Specifications that outline how UI will look and feel like. This determines the user experience of the product.
- **System feature requirements:** These outline the features needed in order for the product to function.

5. Have Stakeholders Approve The Software Development Requirements

Once you define and document your software development requirements in your SRS document, the final step that remains is to send it to stakeholders for revision and approval.

Everyone should review the final version of this document - the development and design team that worked on it, the business or a company that commissioned it, the sponsors that funded it as well as a target audience sample to review its functions and features.

This is the final step of making sure everyone is on the same page before the production of the solution begins.

This is when SRS reviewers can file in their last-minute suggestions, complaints and ideas for the improvement of the process and the finished product.

What Are The Non-functional Requirements In Software Development?

In software development, there are two types of requirements: functional and nonfunctional.

- **Functional requirements:** These are the product features that the development team is going to design, code and test. They define the functionality of the software product that will help in solving users' pain points. These requirements are defined by "what" questions such as:
 - What should the software system do?
 - What functions or functionality will the product support?

- What information or data will it manage?
- **Nonfunctional requirements:** These describe how each feature should behave under certain conditions and what limitations they should have. They serve as a description of the functions that are important for the stakeholders. These requirements are defined by “how” questions, like: “How will the system do what it is designed for?” They establish standards for
 - Security
 - Design
 - Accessibility
 - Performance
 - Reliability

Nonfunctional requirements complement functional requirements. The former are the list of specific features, while the latter outline the functionality of the software.

To illustrate, a functional requirement could be the ability of the software solution to send messages or transfer files.

A nonfunctional requirement would be offering these functional requirements in all the major browsers and operating systems or supporting them in the mobile device layout.

7 Risks Of Having Undocumented Software Requirements

It is not possible to know if the software product and its features are developed properly without having specified and documented software parameters.

A lot of things can go wrong if software requirements are not thoroughly analyzed and documented.

Having no official software requirements specifications can result in the following ways:

1. Bugs and errors escalate in the system
2. Developers need to discern the specific features based on spoken instructions and how they understood them
3. There is no official, recorded agreement on what makes the final product
4. The client doesn't know what end-product to expect
5. Cases of miscommunication happen across the entire project and in all of its sectors

6. As a result of miscommunication and poor development, bug fixes and reworks are needed
7. Costs go up and it's very difficult to meet the deadlines

Takeaways On Software Requirements Specification

When it comes to outlining and defining your software product's requirements, it is of most paramount importance to:

- Understand the purpose of the product and the development process
- Define the business requirements
- Decide on the development methodology
- Define the functional and nonfunctional requirements
- Create a comprehensive schedule
- Set priorities
- Have stakeholders review the software requirements document

review for correctness

We rely on software to control almost everything around us. Interaction with embedded systems such as cars, pacemakers, and satellites is part of everyday life. Runtime errors within these systems create liabilities. Preventing these cases before they impact people's lives is critical. This leaves engineers with the major challenge of demonstrating **software correctness**.

What is correctness?

A program is correct when it behaves exactly as intended, and no other way. A minimum prerequisite to reach correctness is **safety**. Engineers should not trust their own code or let others think it is trustworthy without evidence. For software to be considered correct, an engineer must demonstrate that their program works as intended, matching

specifications. There are several tools that can help an engineer demonstrate the correctness of their code.

Tools for correctness

The first way an engineer can ensure software correctness is through **Deep Specification**. DeepSpec establishes the attributes of a comprehensive (deep) specification. In order for a specification to be deep, it must be:

1. **Rich:** Describes complex component behavior in detail
2. **Live:** Connected to the implementation
3. **Formal:** Written in a language with clear semantics that can be consumed by provers and testing tools
4. **Two-sided:** Connected to both implementations and clients

Two common types of tools to create and reason about Deep Specification are **interactive proof assistants** and **property-based testing**.

Interactive proof assistants help engineers develop formal proofs about their code. Engineers can use tools like Coq to develop code, write and verify proofs about the code, and then extract the verified code. An engineer has evidence their program behaves as intended when proofs about the code are machine-checked.

Check out this content from our parent company, Galois, on Formal Verification

Property-based testing shows that specification is met over many computer-generated inputs. First, software engineers develop test harnesses in their native software stack.

Test harnesses include:

1. Description of data to be generated
2. Software to test
3. Specification written in the native language as boolean expression

Engineers use property-based testing tools like DeepState and Hypothesis to create test harnesses and generate inputs. Property-based testing tools can reveal if program outputs are true to specification.

On their own, property-based testing tools can't provide evidence at the level of interactive proof assistants. Pairing them with instrumentation such as ASan and UBSan can build confidence in correctness.

What's the risk?

It's important to consider risk when reasoning about software correctness. The soundness of tools used to check correctness and program transformations (such as compilation), is important to consider. Property-based testing cannot guarantee specifications will be met, but interactive proof assistants can. If a proof is machine-checked, then the result should be considered correct. The tradeoff is the **ease of use** of property-based testing tools versus **confidence of correctness** with interactive proof assistants. Software engineers can execute test harnesses and type check proofs to ensure confidence about the correctness of their software.

Correctness is possible

It is possible to have and demonstrate confidence in software correctness. **Writing deep specification** for your software and **verifying** or **testing** it is the key.

Consistency

In an ideal world consistency in software is having one and only one way to do anything in your company, a single build pipeline, a single database technology, a single API framework, a single way to log, a single way to document, a single way to deploy...

Technical debt is often talked about in terms of old technologies, legacy codebases, monoliths and tight coupling. All these things and others are worth thinking about when it comes to technical debt, but consistency above all else should be the highest concern. At its core, technical debt is borrowing from tomorrow's velocity to speed up today's. Knowingly only changing one part of the system for some benefit, when a global update would be much more expensive is the easiest way to accrue this debt.

Adding, updating and maintaining functionality in a consistent system tends to have a linear, purely additive cost — it's just yet another API endpoint, database call, or test case to support. Introducing inconsistency has a multiplicative cost — in the future it's now twice as hard to understand and support two different logging platforms, monitor two different queue technologies, or remember the two code paths that create users in your system.

Inconsistency also bleeds your engineering organisation in other ways; it creates differences between teams, making it harder to move engineers around when different product opportunities arise. Production support is more complex as engineers don't want to, or aren't capable of, supporting systems that work differently to theirs. New work is

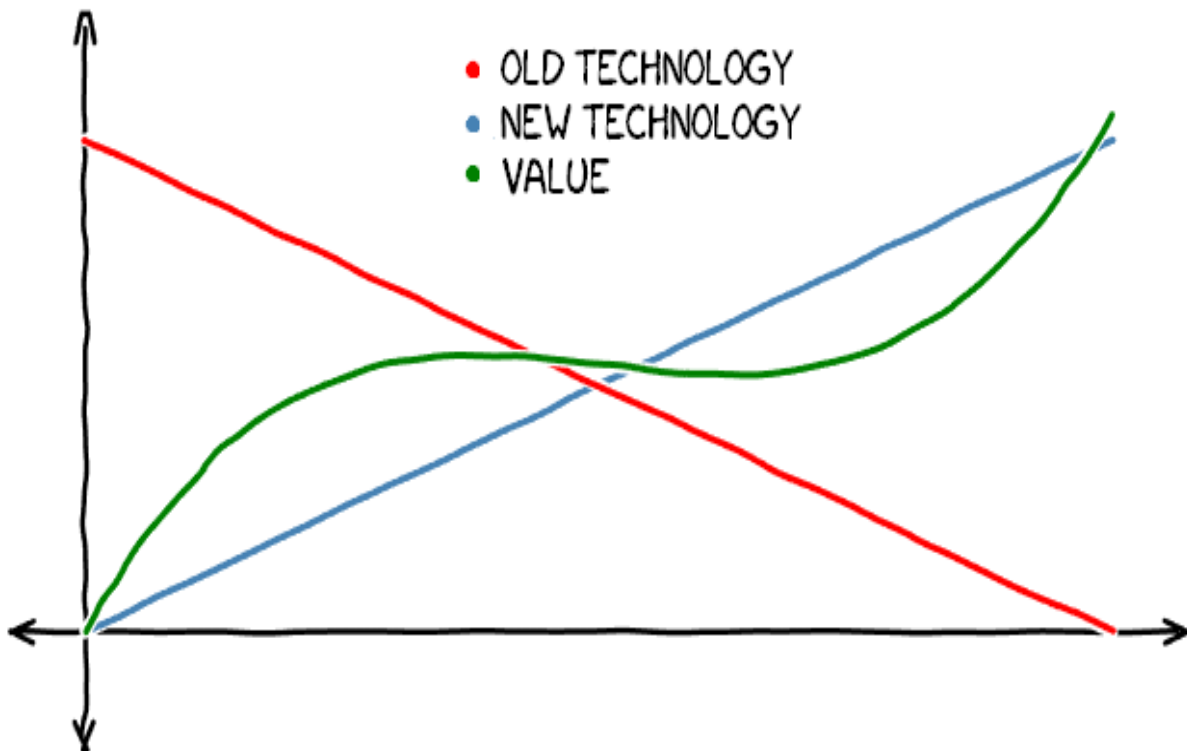
more complicated as all the competing choices need to be evaluated to see which is the correct new best practice to copy. It promotes pockets of favouritism and emotional investment when someone's choice eventually needs to be deprecated.

All of these downsides are forms of, or effects from, knowledge silo-ing that will also mean that you are much more susceptible to employee churn. Handovers become more critical, and more often required. It is then seen as simpler to just use something new than try to fully understand that landscape of what already exists without a guide. Knowledge gets lost and the inconsistency multiplies.

As framing it as 'technical debt' implies, inconsistency charges interest, but consistent software can actually be thought of as an asset, with all the benefits of compound interest. It's much easier to take a consistent system from one consistent state to another rather than try to impose consistency on a mess competing choices. Migrating to a new database technology is easier in an organisation if you're not using three different ones already. Both technically (tech-specific migration tooling) and organisationally (different teams might have sunk cost in their own choices). In a consistent organisation any improvements or optimisations you make will be applicable to your entire estate, and you'll get the benefits of economies of scale. The longer you keep consistency, the more system-wide upgrades you can make at a lower cost.

While consistent systems make changing easier, they also remove choice. An odd sounding benefit, but a valuable one. There is no use in every team spending time evaluating the same decision, like which logging framework to use, when they could be writing valuable business logic. In a consistent system, whichever technology you do settle on benefits from multiple team's investment. It becomes easy to make the right choice, it comes with so much well trodden ground it's a no-brainer. If another option looks better for you, then it should probably be evaluated and rolled out for everyone.

If it's so obviously better, why do systems get inconsistent? New technologies bring large peaks of value at two stages. When they are first used and the benefits over the old technology are put to work, and when the final piece of the old technology is killed off, freeing everyone's head-space up from its idiosyncrasies. Between these two peaks is potentially a long slog, involving multiple teams working together. It is much easier for an individual or a single team to access the value from the first peak — often you can introduce something new, alone. However only through collaboration can you reach the second peak, and often organisations aren't up for that challenge.



The argument against consistency tends to be one of agility. Individual teams making choices for their benefit so they can move fast. Effectively focusing on that first peak of value. There is nothing wrong with this argument, it's why inconsistency is technical debt.

An engineering loan you can make for the short term. It however needs to be entered into like any loan; with full knowledge of the terms of the borrowing.

How do you move to have more consistency? Simply, and flippantly, just finish what you started. More seriously, almost every large scale software challenge is an organisational challenge in disguise. This challenge breaks down into three parts; decisions, follow-through and prioritisation.

Maybe architects, maybe a chapter, maybe management, but somehow force a decision making process. Decisions need to be able to be made and choices enforced and stuck to long enough to make it through to the second peak of value — consistency. This often rubs engineers up the wrong way, who feel strongly about freedom of choice and expanding their skill set. An effective counter to this is to delegate responsibility. Think about all the different individual tools, languages, frameworks that could be consistent and hand out responsibility of each of these choices to individuals. A senior engineer can be responsible for logging and its optimisation and their work will influence every engineering team in your business. It gives people the opportunity to deepen, rather than broaden their skill sets and provides strong ownership.

The second part is follow-through. Whenever your preferred decision making process comes up with the next great thing, cost out the entire migration process and make plan that takes you all the way to consistency. Too often engineering departments fail to communicate correctly to the rest of the business the benefits of consistency and even more often fail to actually plan and pitch for consistency in the first place. It's okay to compromise to inconsistency because the bills need to be paid, but you should aim for the full migration and know where you would spend any time you do get — a hit list of inconsistencies. Be sure to pitch this work, the second peak of value, in the name of future velocity, it's an investment with clear, long term benefits to the company, the opposite of over-engineering indulgence.

Finally, where do you start? Look to see where you find it hardest to estimate work. Where teams are spending the most time on non-product work. What work gets the most pull-request conversations. What is hard to organise support for. Where is work slowest. What has the highest bus factor. These are many signals of friction caused by inconsistency.

A useful exercise to go through is to ask which teams is it hardest to move engineers between (within roughly similar skill sets), and keep asking why. You'll uncover inconsistencies through the learning curve they need to go through to become an effective team member. What do you think they'll find most confusing or difficult to adapt to, that's probably a large source of inconsistency in your business. Likely culprits are programming language, deployment pipeline or production monitoring.

Every time you do engineering work, be it choosing frameworks, designing services or just writing code, ask yourself or your team— "Are we leaving the system in a more or less consistent state". If the answer is less, at what point will consistency increase, do you have a plan to get there and are you all happy with the trade-off you're making in the meantime.

Completeness

Use case models are well-known specification documents technique used to describe the functional requirements of a software system. The rationale of use cases is to specify requirements in some natural language text such that it can also be used for communication with the stakeholders. However, different variations of use cases have been proposed and used in different software development activities. And, this raises an issue about the completeness of specification "which variation of use case to be used?", "does the use of a specific variant affect the specification completeness", and so on. In this work, we performed an analysis of different variants of use cases with respect to their functional completeness. The materials related to study and examples of the use case variants are placed in this page**.

This page consists of all the study materials used to evaluate the functional completeness of the use case specification. Specifically, we present

- Problem specifications of five different systems in a plain text format
- Example of the eight variants of use case templates used in the study
- Example of the use case questionnaires generated using W⁵H² analysis approach (What, Who, When, Where, Why, How, How much)
- Example of the implementation of W⁵H² approach in a Cockburn use case template
- Example of the modified Cockburn's use case template by adding four missing significant use case elements (identified from our study findings).

UNIT - III

Designing Software Solutions

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 Design Levels

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 components interacting with each other. At this level, the designers get the idea of proposed solution domain.
- **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 each other.
- **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 other modules.

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. These modules may work as basic constructs for the entire software. Designers tend to design modules such that they can be executed and/or compiled separately and independently.

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

Advantage of modularization:

- Smaller components are easier to maintain
- Program can be divided based on functional aspects
- Desired level of abstraction can be brought in the program
- Components with high cohesion can be re-used again
- Concurrent execution can be made possible
- Desired from security aspect

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 after another 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 of execution.

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 and Cohesion

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, considered as single entity but may refer to each other to work together. There are measures by which the quality of a design of modules and their interaction among them can be measured. These measures are called coupling and cohesion.

Cohesion

Cohesion is a measure that defines the degree of intra-dependability within elements of a module. The greater the cohesion, the better is the program design.

There are seven types of cohesion, namely –

- **Co-incident 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 generally not-accepted.
- **Logical cohesion** - When logically categorized elements are put together into a module, it is called logical cohesion.
- **Temporal Cohesion** - When elements of module are organized such that they are processed at a similar point in time, it is called temporal cohesion.
- **Procedural cohesion** - When elements of module are grouped together, which are executed sequentially in order to perform a task, it is called procedural cohesion.
- **Communicational cohesion** - When elements of module are grouped together, which are executed sequentially and work on same data (information), it is called communicational cohesion.
- **Sequential cohesion** - When elements of module are grouped because the output of one element serves as input to another and so on, it is called sequential cohesion.
- **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 be reused.

Coupling

Coupling is a measure that defines the level of inter-dependability among modules of a program. It tells at what level the modules interfere and interact with each other. The lower the coupling, the better the program.

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 level coupling.
- **Common coupling**- When multiple modules have read and write access to some global data, it is called common or global coupling.
- **Control coupling**- Two modules are called control-coupled if one of them decides the function of the other module or changes its flow of execution.
- **Stamp coupling**- When multiple modules share common data structure and work on different part of it, it is called stamp coupling.
- **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.

Design Verification

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 notation form, then their associated tools for verification should be used otherwise a thorough design review can be used for verification and validation.

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.

Refining the software Specification

Software Design is the process to transform the user requirements into some suitable form, which helps the programmer in software coding and implementation. During the software design phase, the design document is produced, based on the customer requirements as documented in the SRS document. Hence the aim of this phase is to transform the SRS document into the design document.

The following items are designed and documented during the design phase:

- Different modules required.
- Control relationships among modules.
- Interface among different modules.
- Data structure among the different modules.
- Algorithms required to implement among the individual modules.

Objectives of Software Design:

1. Correctness:

A good design should be correct i.e. it should correctly implement all the functionalities of the system.

2. Efficiency:

A good software design should address the resources, time and cost optimization issues.

3. Understandability:

A good design should be easily understandable, for which it should be modular and all the modules are arranged in layers.

4. Completeness:

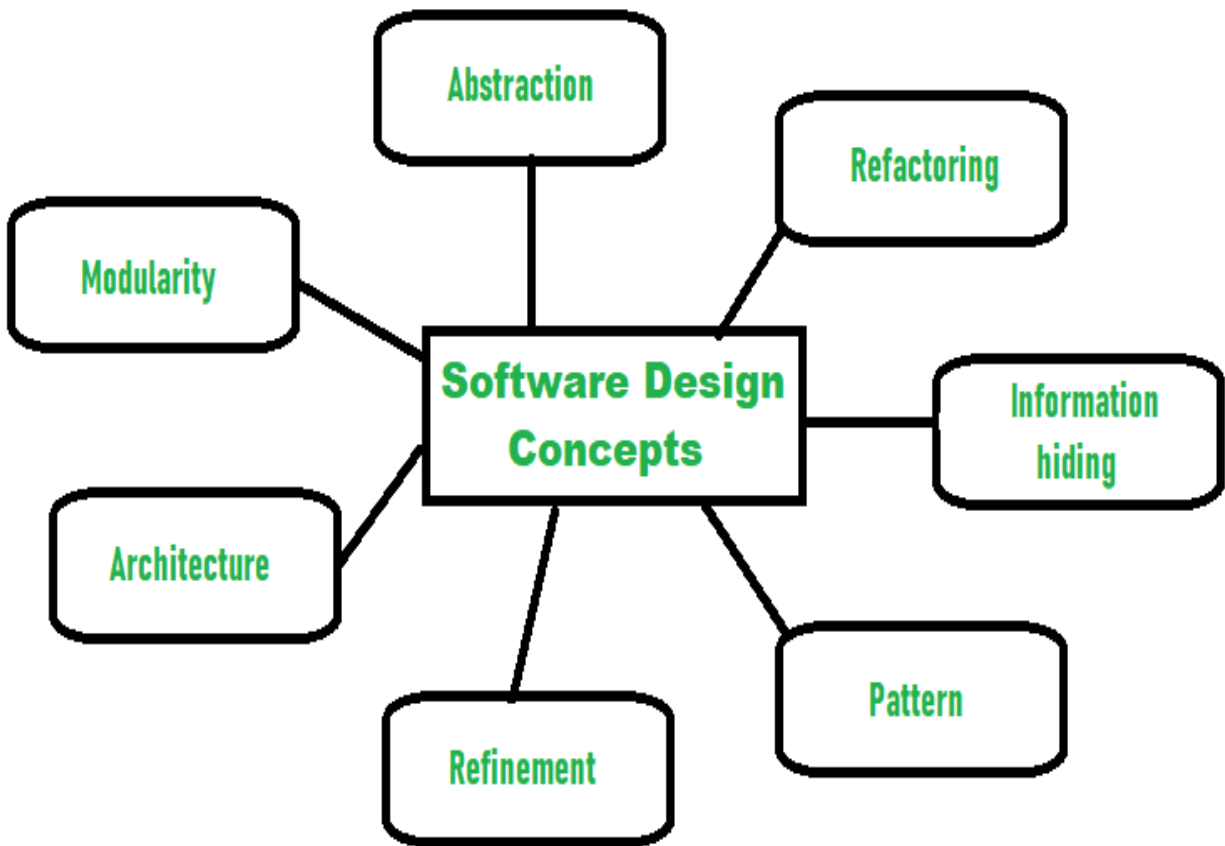
The design should have all the components like data structures, modules, and external interfaces, etc.

5. Maintainability:

A good software design should be easily amenable to change whenever a change request is made from the customer side.

Software Design Concepts:

Concepts are defined as a principal idea or invention that comes in our mind or in thought to understand something. The **software design concept** simply means the idea or principle behind the design. It describes how you plan to solve the problem of designing software, the logic, or thinking behind how you will design software. It allows the software engineer to create the model of the system or software or product that is to be developed or built. The software design concept provides a supporting and essential structure or model for developing the right software. There are many concepts of software design and some of them are given below:



Following points should be considered while designing a Software:

1. Abstraction- hide relevant data

Abstraction simply means to hide the details to reduce complexity and increases efficiency or quality. Different levels of Abstraction are necessary and must be applied at each stage of the design process so that any error that is present can be removed to increase the efficiency of the software solution and to refine the software solution. The solution should be described in broadways that cover a wide range of different things at a higher level of abstraction and a more detailed description of a solution of software should be given at the lower level of abstraction.

2. Modularity- subdivide the system

Modularity simply means to divide the system or project into smaller parts to reduce the complexity of the system or project. In the same way, modularity in design means to subdivide a system into smaller parts so that these parts can be created independently and then use these parts in different systems to perform different functions. It is necessary to divide the software into components known as modules

because nowadays there are different software available like Monolithic software that is hard to grasp for software engineers. So, modularity in design has now become a trend and is also important.

3. Architecture- design a structure of something

Architecture simply means a technique to design a structure of something. Architecture in designing software is a concept that focuses on various elements and the data of the structure. These components interact with each other and use the data of the structure in architecture.

4. Refinement- removes impurities

Refinement simply means to refine something to remove any impurities if present and increase the quality. The refinement concept of software design is actually a process of developing or presenting the software or system in a detailed manner that means to elaborate a system or software. Refinement is very necessary to find out any error if present and then to reduce it.

5. Pattern- a repeated form

The pattern simply means a repeated form or design in which the same shape is repeated several times to form a pattern. The pattern in the design process means the repetition of a solution to a common recurring problem within a certain context.

6. Information Hiding- hide the information

Information hiding simply means to hide the information so that it cannot be accessed by an unwanted party. In software design, information hiding is achieved by designing the modules in a manner that the information gathered or contained in one module is hidden and it can't be accessed by any other modules.

7. Refactoring- reconstruct something

Refactoring simply means to reconstruct something in such a way that it does not affect the behavior or any other features. Refactoring in software design means to reconstruct the design to reduce complexity and simplify it without affecting the behavior or its functions. Fowler has defined refactoring as “the process of changing a software system in a way that it won't affect the behavior of the design and improves the internal structure”.

Different levels of Software Design:

There are three different levels of software design. They are:

1. **Architectural Design:**

The architecture of a system can be viewed as the overall structure of the system & the way in which structure provides conceptual integrity of the system. The architectural design identifies the software as a system with many components interacting with each other. At this level, the designers get the idea of the proposed solution domain.

2. **Preliminary or high-level design:**

Here the problem is decomposed into a set of modules, the control relationship among various modules identified and also the interfaces among various modules are identified. The outcome of this stage is called the program architecture. Design representation techniques used in this stage are structure chart and UML.

3. **Detailed design:**

Once the high level design is complete, detailed design is undertaken. In detailed design, each module is examined carefully to design the data structure and algorithms. The stage outcome is documented in the form of a module specification document.

Application of fundamental design concept for data

- The main aim of design engineering is to generate a model which shows firmness, delight and commodity.
- Software design is an iterative process through which requirements are translated into the blueprint for building the software.

Software quality guidelines

- A design is generated using the recognizable architectural styles and compose a good design characteristic of components and it is implemented in evolutionary manner for testing.
- A design of the software must be modular i.e the software must be logically partitioned into elements.
- In design, the representation of data , architecture, interface and components should be distinct.
- A design must carry appropriate data structure and recognizable data patterns.
- Design components must show the independent functional characteristic.

- A design creates an interface that reduce the complexity of connections between the components.
- A design must be derived using the repeatable method.
- The notations should be use in design which can effectively communicates its meaning.

Quality attributes

The attributes of design name as 'FURPS' are as follows:

Functionality:

It evaluates the feature set and capabilities of the program.

Usability:

It is accessed by considering the factors such as human factor, overall aesthetics, consistency and documentation.

Reliability:

It is evaluated by measuring parameters like frequency and security of failure, output result accuracy, the mean-time-to-failure(MTTF), recovery from failure and the the program predictability.

Performance:

It is measured by considering processing speed, response time, resource consumption, throughput and efficiency.

Supportability:

- It combines the ability to extend the program, adaptability, serviceability. These three term defines the maintainability.
- Testability, compatibility and configurability are the terms using which a system can be easily installed and found the problem easily.
- Supportability also consists of more attributes such as compatibility, extensibility, fault tolerance, modularity, reusability, robustness, security, portability, scalability.

Design concepts

The set of fundamental software design concepts are as follows:

1. Abstraction

- A solution is stated in large terms using the language of the problem environment at the highest level abstraction.
- The lower level of abstraction provides a more detail description of the solution.
- A sequence of instruction that contain a specific and limited function refers in a procedural abstraction.
- A collection of data that describes a data object is a data abstraction.

2. Architecture

- The complete structure of the software is known as software architecture.
- Structure provides conceptual integrity for a system in a number of ways.
- The architecture is the structure of program modules where they interact with each other in a specialized way.
- The components use the structure of data.
- The aim of the software design is to obtain an architectural framework of a system.
- The more detailed design activities are conducted from the framework.

3. Patterns

A design pattern describes a design structure and that structure solves a particular design problem in a specified content.

4. Modularity

- A software is separately divided into name and addressable components. Sometime they are called as modules which integrate to satisfy the problem requirements.
- Modularity is the single attribute of a software that permits a program to be managed easily.

5. Information hiding

Modules must be specified and designed so that the information like algorithm and data presented in a module is not accessible for other modules not requiring that information.

6. Functional independence

- The functional independence is the concept of separation and related to the concept of modularity, abstraction and information hiding.
- The functional independence is accessed using two criteria i.e Cohesion and coupling.
Cohesion

- Cohesion is an extension of the information hiding concept.
- A cohesive module performs a single task and it requires a small interaction with the other components in other parts of the program.

Coupling

Coupling is an indication of interconnection between modules in a structure of software.

7. Refinement

- Refinement is a top-down design approach.
- It is a process of elaboration.
- A program is established for refining levels of procedural details.
- A hierarchy is established by decomposing a statement of function in a stepwise manner till the programming language statement are reached.

8. Refactoring

- It is a reorganization technique which simplifies the design of components without changing its function behaviour.
- Refactoring is the process of changing the software system in a way that it does not change the external behaviour of the code still improves its internal structure.

9. Design classes

- The model of software is defined as a set of design classes.
- Every class describes the elements of problem domain and that focus on features of the problem which are user visible.

architectural and procedural designs using software blue print methodology and object oriented design paradigm

The object-oriented (OO) paradigm took its shape from the initial concept of a new programming approach, while the interest in design and analysis methods came much later. OO analysis and design paradigm is the logical result of the wide adoption of OO programming languages.

- The first object-oriented language was **Simula** (Simulation of real systems) that was developed in 1960 by researchers at the Norwegian Computing Center.
- In 1970, **Alan Kay** and his research group at Xerox PARC created a personal computer named **Dynabook** and the first pure object-oriented programming language (OOPL) - Smalltalk, for programming the Dynabook.
- In the 1980s, **Grady Booch** published a paper titled Object Oriented Design that mainly presented a design for the programming language, Ada. In the ensuing editions, he extended his ideas to a complete object-oriented design method.
- In the 1990s, **Coad** incorporated behavioral ideas to object-oriented methods.

The other significant innovations were Object Modeling Techniques (OMT) by **James Rum Baugh** and Object-Oriented Software Engineering (OOSE) by **Ivar Jacobson**.

Introduction to OO Paradigm

OO paradigm is a significant methodology for the development of any software. Most of the architecture styles or patterns such as pipe and filter, data repository, and component-based can be implemented by using this paradigm.

Basic concepts and terminologies of object-oriented systems –

Object

An object is a real-world element in an object-oriented environment that may have a physical or a conceptual existence. Each object has –

- Identity that distinguishes it from other objects in the system.
- State that determines characteristic properties of an object as well as values of properties that the object holds.
- Behavior that represents externally visible activities performed by an object in terms of changes in its state.

Objects can be modeled according to the needs of the application. An object may have a physical existence, like a customer, a car, etc.; or an intangible conceptual existence, like a project, a process, etc.

Class

A class represents a collection of objects having same characteristic properties that exhibit common behavior. It gives the blueprint or the description of the objects that can be created from it. Creation of an object as a member of a class is called instantiation. Thus, an object is an **instance** of a class.

The constituents of a class are –

- A set of attributes for the objects that are to be instantiated from the class. Generally, different objects of a class have some difference in the values of the attributes. Attributes are often referred as class data.
- A set of operations that portray the behavior of the objects of the class. Operations are also referred as functions or methods.

Example

Let us consider a simple class, Circle, that represents the geometrical figure circle in a two-dimensional space. The attributes of this class can be identified as follows –

- x-coord, to denote x-coordinate of the center
- y-coord, to denote y-coordinate of the center
- a, to denote the radius of the circle

Some of its operations can be defined as follows –

- findArea(), a method to calculate area
- findCircumference(), a method to calculate circumference

- `scale()`, a method to increase or decrease the radius

Encapsulation

Encapsulation is the process of binding both attributes and methods together within a class. Through encapsulation, the internal details of a class can be hidden from outside. It permits the elements of the class to be accessed from outside only through the interface provided by the class.

Polymorphism

Polymorphism is originally a Greek word that means the ability to take multiple forms. In object-oriented paradigm, polymorphism implies using operations in different ways, depending upon the instances they are operating upon. Polymorphism allows objects with different internal structures to have a common external interface. Polymorphism is particularly effective while implementing inheritance.

Example

Let us consider two classes, Circle and Square, each with a method `findArea()`. Though the name and purpose of the methods in the classes are same, the internal implementation, i.e., the procedure of calculating an area is different for each class. When an object of class Circle invokes its `findArea()` method, the operation finds the area of the circle without any conflict with the `findArea()` method of the Square class.

Relationships

In order to describe a system, both dynamic (behavioral) and static (logical) specification of a system must be provided. The dynamic specification describes the relationships among objects e.g. message passing. And static specification describe the relationships among classes, e.g. aggregation, association, and inheritance.

Message Passing

Any application requires a number of objects interacting in a harmonious manner. Objects in a system may communicate with each other by using message passing. Suppose a system has two objects – `obj1` and `obj2`. The object `obj1` sends a message to object `obj2`, if `obj1` wants `obj2` to execute one of its methods.

Composition or Aggregation

Aggregation or composition is a relationship among classes by which a class can be made up of any combination of objects of other classes. It allows objects to be placed directly within the body of other classes. Aggregation is referred as a “part-of” or “has-a” relationship, with the ability to navigate from the whole to its parts. An aggregate object is an object that is composed of one or more other objects.

Association

Association is a group of links having common structure and common behavior. Association depicts the relationship between objects of one or more classes. A link can be defined as an instance of an association. The Degree of an association denotes the number of classes involved in a connection. The degree may be unary, binary, or ternary.

- A unary relationship connects objects of the same class.
- A binary relationship connects objects of two classes.
- A ternary relationship connects objects of three or more classes.

Inheritance

It is a mechanism that permits new classes to be created out of existing classes by extending and refining its capabilities. The existing classes are called the base classes/parent classes/super-classes, and the new classes are called the derived classes/child classes/subclasses.

The subclass can inherit or derive the attributes and methods of the super-class (es) provided that the super-class allows so. Besides, the subclass may add its own attributes and methods and may modify any of the super-class methods. Inheritance defines a “is – a” relationship.

Example

From a class Mammal, a number of classes can be derived such as Human, Cat, Dog, Cow, etc. Humans, cats, dogs, and cows all have the distinct characteristics of mammals. In addition, each has its own particular characteristics. It can be said that a cow “is – a” mammal.

OO Analysis

In object-oriented analysis phase of software development, the system requirements are determined, the classes are identified, and the relationships among classes are acknowledged. The aim of OO analysis is to understand the application domain and specific requirements of the system. The result of this phase is requirement specification and initial analysis of logical structure and feasibility of a system.

The three analysis techniques that are used in conjunction with each other for object-oriented analysis are object modeling, dynamic modeling, and functional modeling.

Object Modeling

Object modeling develops the static structure of the software system in terms of objects. It identifies the objects, the classes into which the objects can be grouped into

and the relationships between the objects. It also identifies the main attributes and operations that characterize each class.

The process of object modeling can be visualized in the following steps –

- Identify objects and group into classes
- Identify the relationships among classes
- Create a user object model diagram
- Define a user object attributes
- Define the operations that should be performed on the classes

Dynamic Modeling

After the static behavior of the system is analyzed, its behavior with respect to time and external changes needs to be examined. This is the purpose of dynamic modeling.

Dynamic Modeling can be defined as “a way of describing how an individual object responds to events, either internal events triggered by other objects, or external events triggered by the outside world.”

The process of dynamic modeling can be visualized in the following steps –

- Identify states of each object
- Identify events and analyze the applicability of actions
- Construct a dynamic model diagram, comprising of state transition diagrams
- Express each state in terms of object attributes
- Validate the state–transition diagrams drawn

Functional Modeling

Functional Modeling is the final component of object-oriented analysis. The functional model shows the processes that are performed within an object and how the data changes, as it moves between methods. It specifies the meaning of the operations of an object modeling and the actions of a dynamic modeling. The functional model corresponds to the data flow diagram of traditional structured analysis.

The process of functional modeling can be visualized in the following steps –

- Identify all the inputs and outputs
- Construct data flow diagrams showing functional dependencies
- State the purpose of each function
- Identify the constraints
- Specify optimization criteria

Object-Oriented Design

After the analysis phase, the conceptual model is developed further into an object-oriented model using object-oriented design (OOD). In OOD, the technology-independent concepts in the analysis model are mapped onto implementing classes, constraints are identified, and interfaces are designed, resulting in a model for the solution domain. The main aim of OO design is to develop the structural architecture of a system.

The stages for object-oriented design can be identified as –

- Defining the context of the system
- Designing the system architecture
- Identification of the objects in the system
- Construction of design models
- Specification of object interfaces

OO Design can be divided into two stages – Conceptual design and Detailed design.

Conceptual design

In this stage, all the classes are identified that are needed for building the system. Further, specific responsibilities are assigned to each class. Class diagram is used to clarify the relationships among classes, and interaction diagram are used to show the flow of events. It is also known as **high-level design**.

Detailed design

In this stage, attributes and operations are assigned to each class based on their interaction diagram. State machine diagram are developed to describe the further details of design. It is also known as **low-level design**.

Design Principles

Following are the major design principles –

Principle of Decoupling

It is difficult to maintain a system with a set of highly interdependent classes, as modification in one class may result in cascading updates of other classes. In an OO design, tight coupling can be eliminated by introducing new classes or inheritance.

Ensuring Cohesion

A cohesive class performs a set of closely related functions. A lack of cohesion means — a class performs unrelated functions, although it does not affect the operation of the whole system. It makes the entire structure of software hard to manage, expand, maintain, and change.

Open-closed Principle

According to this principle, a system should be able to extend to meet the new requirements. The existing implementation and the code of the system should not be modified as a result of a system expansion. In addition, the following guidelines have to be followed in open-closed principle –

- For each concrete class, separate interface and implementations have to be maintained.
- In a multithreaded environment, keep the attributes private.
- Minimize the use of global variables and class variables.

creating design document

Today, Internet is considered as a knowledge-base. Anyone can access any kind of information using the Internet, such as documents, view hypertext, and multimedia (audio and video) through web server database.

Also, it has become necessary and critical for any organization to provide public access to corporate web sites, which required constant, reliable, interactive web form, authentic transaction and relevant documents. This has driven the organization to adapt to Web Content Management systems.

Gradually this led to Knowledge Management system and Knowledge Base system.

Although Knowledge management system precede the internet, it became easy

for the user to access information using internet, because it was like a distributed database on the internet.

In the due course, distinction between Knowledge Management System and Knowledge Base system became very minimal. Although these systems are still considered as content repositories, which allows you to store, query and take appropriate decisions.

So, in Knowledge Management System, you just store them (manuals, procedures, policies, best practices, reusable designs and code, etc.) in the database.

In Knowledge Based system, you meticulously classify and categorize them (manuals, procedures, policies, best practices, reusable designs and code, etc.) appropriately in meaningful sections, sub sections and groups.

Important

There are many knowledge base software (For more information refer to List of Knowledge Base Software) available which is listed , but, this document will specifically guide you in understanding how to create a software documentation for Knowledge Base product, and also, aims at bringing you the best and powerful Document360 features that you can depend on to control, manage, and maintain your Knowledge Base artifacts.

Why you need knowledge base software?

Good documentation practices are important for the success of the software. Documentation must comprise an interactive User Experience, Information Architecture, and good understanding of your audience.

Note: It is recommended that you suggest building the documentation deliverable into your development process, while attempting to use the Agile methodologies for software development.

It needs to serve the purpose of resolving the issues, when encountered by the developer, end user or while during customer facing the Knowledge Base.

Software Documentation

Software documentation is a part of any software. Appropriate details and description need to be in the documented to achieve the following goals:

- Resolve issue encountered by the developer during the development process
- Help end-user to understand the product
- Assist customers and the support team to find the information.

Documentation can be related to an API documentation (which can be used to either incorporate in the code, or to extend the functionality of the existing application, release notes that serves what bugs had been fixed in the current release, and what code had been refracted) and, or customer-facing help content to easily find required information immediately.

Software documentation helps you to understand the product, interface, capability, ability to fulfill a task, and quickly search and find a particular section within the document, or find resolution when encountered using the product.

Note: Even when there are knowledge workers, yet, 51% of people prefer to receive technical support through a Knowledge Base, and yet producing the relevant documentation is challenging for any companies.

Types of Software Documentation

Many types of documents are required and delivered during the product development life cycle and software development life cycle, like Software documentation, Developer documentation, Software requirement document, and design documentation and audience analysis.

User Documentation

This document is mostly delivered for end-user who actually want use the product themselves, to understand and complete a certain task.

- How-to guides – Guides the user to complete a task or a predetermined goal.
- Tutorials – Learns a concept by following a series of steps concept

- Reference docs – Describes the technical detail of the product (Software requirement specification, software design documents and so on)
- Just-in-time document: Specifies how to resolve a particular issue, but not part of User documentation
- Administration Guide: Enables the administrator to refer to this after installing an application
- Configuration Guide: Allows the administrator to refer to this document for configuration parameters.

Developer Documentation

This documentation refers to system related documentation.

- API documentation –Specifies how to invoke API calls and classes, or how to include API in the code that is being developed.
- Release notes: Describes about the latest software, feature releases, and what bugs have been fixed. Usually this document is a text file with a filename extension (.txt).
- README: A high-level overview of the software, usually alongside the source code.
- System documentation – Describes the system requirements, includes design documents and UML diagrams.

Just-in-time Documentation

Situation may arise, where a just-in-time document quickly serves the support for customer-facing documentation. The user need not have to refer to any documents or FAQs for information.

It is recommended that documentation tools be common across your development team, so that it can be easily be accessible within the environment, and you need to initiate that the documentation becomes a mandatory part of the Software development life cycle process. For example, GitHub is a cloud-based application, which serves the purpose for code developers and authors.

List of Knowledge Base software

Following is the list of popular Knowledge Base software

1. Document360
2. KnowAll on WordPress
3. LiveAgent
4. Helpjuice
5. ZenDesk
6. ServiceNow
7. ProProf Knowledge Base

This document mainly aimed at Document360, which you can use to create powerful Knowledge Base documents.

Document360 supports the following task:

- Custom Dashboard
- File Manager
- Article Redirect
- Team Management
- Team Auditing
- Secure Hosting
- Backup and Restore
- Custom Security
- In-app Assistance

6 Overview of Document360 Knowledge Base application interface

Document360 is a cloud-based application, with a simple interface, which helps user to quickly understand the overall functionality and the layout. The following Document360 screen explains the various elements of the interface.

Build Multilingual Knowledge Base documents using Localization feature

The Localization feature is tightly integrated into Document360, which enables you to build multilingual knowledge base documents for the different customer base. For example, your international customers across the globe may want to read the document in their own native language, even when the document is available in English.

You can take the help of an in-house translator to translate the article manually for public facing page or use Google translator to translate the copied content from the source, and then paste the translated content back in to the editor. If you are unhappy with the translation, you can use Document30 built-in Machine Translation using artificial intelligence (AI) to instantly convert the document, or literally convert many documents in attempt using the bulk operation.

Create custom access to Knowledge Base using Team or Readers Accounts

This section describes how to create custom roles to control access to Document360 Knowledge Base.

There are two types of user accounts in Document360 – Team Accounts (Used by your internal team in the organization that builds and manages Knowledge Base) and Readers Accounts (End-users for whom the Knowledge Base is being built).

There are three tabs in the Team Management page.

Team Members: You can invite new member(s) to the team, assign the level of privileges (Access rights – Admin, Owner, Draft Writer or Editor)) and allocate security measures (to a complete project, version or category) to this member.

Security Groups: You can create predefined security profile that can later be

associated to a particular team member or applied to a group consisting predetermined team members.

Roles: You can edit any of the roles listed in the roles screen.

Create custom Site domain in Document360 Knowledge Base

When you create a document in Document360, you can have a public facing URL, which can help user(s) to access to your Knowledge Base.

You can perform the following in the Site Domain page:

- Site Domain Hosting: You can create a custom URL for an existing URL
- Sub-Folder Hosting: Customized URL. You can use sub-folder as a hosting page to mark these documents internally within the organization. For example, <https://docs.startup.com> to <https://startup.com/docs>
The following screen displays the Site Domain page.

Backup and Restore your Knowledge Base in Document360

It is necessary to keep your data secure each day. The Backup and Restore function is integrated with Document360, which can help you restore earlier versions of your document from the backup list.

Create Custom knowledge base using Document360

You can create a new design for the current site and define the navigation flow method using the Site Design & Navigations button under the KNOWLEDGE BASE SITE drop-down menu.

Quickly Adapt to Agile or DevOps Methodology for documentation

Most of the companies adapt, or have adapted to Agile and DevOps methodology from the traditional Waterfall method. In waterfall method, the project is planned upfront and the complex entity relationship diagram are created even before the software development begins. This methodology was found to be inappropriate because, it was found difficult to incorporate any new desired changes to the existing product design. This led to Agile methodology, where modular development was considered and new

changes could be implemented during the development process.

This approach was well accepted in today's software development life cycle and documentation development life cycle.

The Agile and DevOps methodology has the following benefits:

- Team members of the project frequently interact and update the status each day on a regular basis
- Update the status, and stories in the JIRA tool
- Quickly respond to change
- Meet project timelines, meet within the budget cost and efficient management.

Regularly Interact with Subject-Matter Experts (SME)

The developers have in-depth knowledge about the product, so, writers often find it hard to approach them quite often to get information for the related document. So, one possible way is to, synchronize the effort estimate of documentation with the Software development process, so that the resources (documentation team, engineers, document reviewers and support) can collaborate regularly, and acquire substantial knowledge to meet the documentation goal.

Provides Seamless and Accessibility support in Document360

When you get to a certain point in your documentation, you need to seriously consider how people with different needs consider using your documentation.

For example, whether your users belong to international audiences, you necessarily avoid the usage of idioms and references to them.

Accessibility relates to how the user experiences the documentation tool in the first place. For example, consider whether the user will be accessing the application using the screen reader. Enabling the screen reader option allows the computer to itself read the text aloud.

Images with text overlaid are not accessible, so think about your screenshots and make sure they have accompanying text.

Constantly Improve and update document(s) Knowledge Base

Documentation is an iterative process. It may need to be improved based on customer feedback or may require refracting some content already included in the document. The Knowledge Base can include customers frequently asked questions or more references to solutions that may need to be included to increase efficiency, productivity, and reduce cost to the company.

Discover how customer reached your Knowledgebase content

Your Knowledge Base software should be indexable by search engines, with all the correct meta tags. You should also link to your documentation from your software app, since this is where users will naturally get stuck.

Few customers will consider your Knowledge Base as a whole, and hardly anyone will arrive at your carefully constructed homepage.

Your software documentation is no good if nobody can find it, but there are a number of ways to promote your content. In fact, Google's search engine is often "page one" for many users.

Periodically update information in Knowledgebase using Customer Feedback loop

Before the first release of the product, the organization invites and shares the beta product to the customers and users to perform testing. Based on their feedback, and subsequent feedback after the release, the documentation must be updated to complement the latest product release.

There are several ways to collect feedback:

- Using a Web form or an Interactive form
- Activate inline comments in the documentation before sharing to the customers
- Receive ratings about the usefulness of the content on specific page

Create your own Style Guide to create consistent document

There are couple of ways to create consistent look and feel of your document.

- Template: Create a template with predetermines styles prior to creating a document.

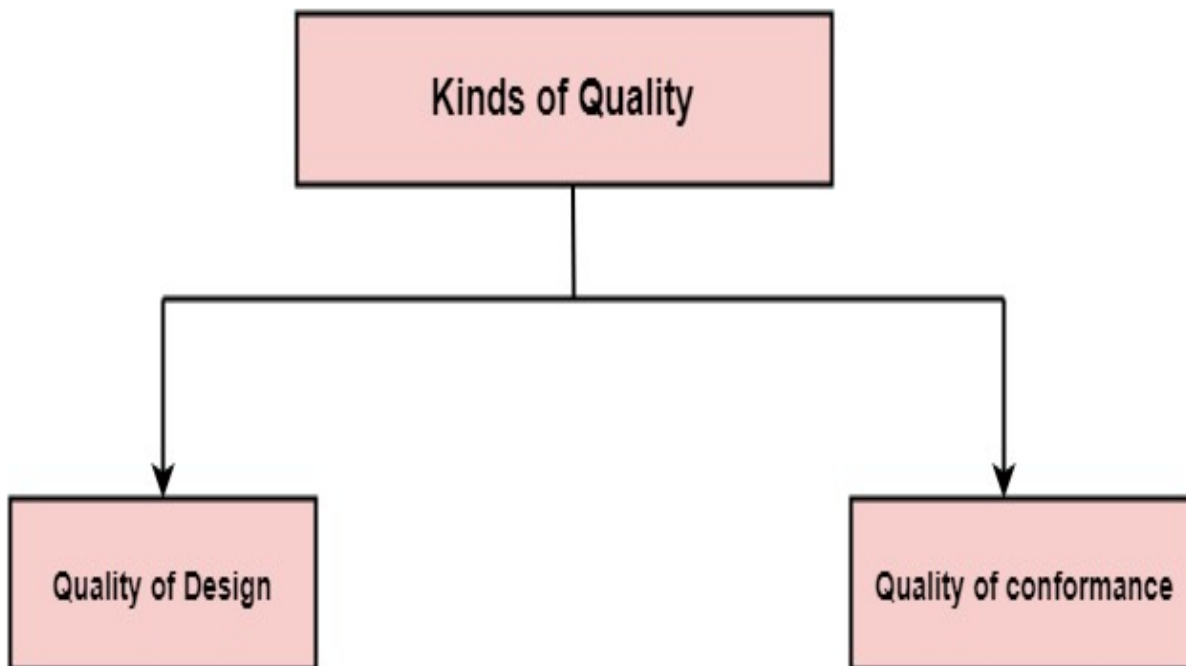
- Define a style sheet: Create different levels and structure the document manually by applying the relevant style to the content.

Note: You can refer standard software style guides like the Microsoft Style Guide or Chicago Style Guide.

Review of conformance to software requirements and quality

Quality defines to any measurable characteristics such as correctness, maintainability, portability, testability, usability, reliability, efficiency, integrity, reusability, and interoperability.

There are two kinds of Quality:



Quality of Design: Quality of Design refers to the characteristics that designers specify for an item. The grade of materials, tolerances, and performance specifications that all contribute to the quality of design.

Quality of conformance: Quality of conformance is the degree to which the design specifications are followed during manufacturing. Greater the degree of conformance, the higher is the level of quality of conformance.

Software Quality: Software Quality is defined as the conformance to explicitly state functional and performance requirements, explicitly documented development standards, and inherent characteristics that are expected of all professionally developed software.

Quality Control: Quality Control involves a series of inspections, reviews, and tests used throughout the software process to ensure each work product meets the requirements placed upon it. Quality control includes a feedback loop to the process that created the work product.

Quality Assurance: Quality Assurance is the preventive set of activities that provide greater confidence that the project will be completed successfully.

Quality Assurance focuses on how the engineering and management activity will be done?

As anyone is interested in the quality of the final product, it should be assured that we are building the right product.

It can be assured only when we do inspection & review of intermediate products, if there are any bugs, then it is debugged. This quality can be enhanced.

Importance of Quality

We would expect the quality to be a concern of all producers of goods and services. However, the distinctive characteristics of software and in particular its intangibility and complexity, make special demands.

Increasing criticality of software: The final customer or user is naturally concerned about the general quality of software, especially its reliability. This is increasing in the case as organizations become more dependent on their computer systems and software is used more and more in safety-critical areas. For example, to control aircraft.

The intangibility of software: This makes it challenging to know that a particular task in a project has been completed satisfactorily. The results of these tasks can be made tangible by demanding that the developers produce 'deliverables' that can be examined for quality.

Accumulating errors during software development: As computer system development is made up of several steps where the output from one level is input to the next, the errors in the earlier 'deliverables' will be added to those in the later stages leading to accumulated determinable effects. In general the later in a project that an error is found, the more expensive it will be to fix. In addition, because the number of errors in the system is unknown, the debugging phases of a project are particularly challenging to control.

Software Quality Assurance

Software quality assurance is a planned and systematic plan of all actions necessary to provide adequate confidence that an item or product conforms to establish technical requirements.

A set of activities designed to calculate the process by which the products are developed or manufactured.

SQA Encompasses

- A quality management approach
- Effective Software engineering technology (methods and tools)
- Formal technical reviews that are tested throughout the software process
- A multitier testing strategy
- Control of software documentation and the changes made to it.
- A procedure to ensure compliances with software development standards
- Measuring and reporting mechanisms.

SQA Activities

Software quality assurance is composed of a variety of functions associated with two different constituencies ? the software engineers who do technical work and an SQA group that has responsibility for quality assurance planning, record keeping, analysis, and reporting.

Following activities are performed by an independent SQA group:

1. **Prepares an SQA plan for a project:** The program is developed during project planning and is reviewed by all stakeholders. The plan governs quality assurance activities performed by the software engineering team and the SQA group. The plan identifies calculation to be performed, audits and reviews to be performed, standards that apply to the project, techniques for error reporting and tracking, documents to be produced by the SQA team, and amount of feedback provided to the software project team.
2. **Participates in the development of the project's software process description:** The software team selects a process for the work to be performed. The SQA group reviews the process description for compliance with organizational policy, internal software standards, externally imposed standards (e.g. ISO-9001), and other parts of the software project plan.

3. **Reviews software engineering activities to verify compliance with the defined software process:** The SQA group identifies, reports, and tracks deviations from the process and verifies that corrections have been made.

4. **Audits designated software work products to verify compliance with those defined as a part of the software process:** The SQA group reviews selected work products, identifies, documents and tracks deviations, verify that corrections have been made, and periodically reports the results of its work to the project manager.

5. **Ensures that deviations in software work and work products are documented and handled according to a documented procedure:** Deviations may be encountered in the project method, process description, applicable standards, or technical work products.

6. **Records any noncompliance and reports to senior management:** Non-compliance items are tracked until they are resolved.

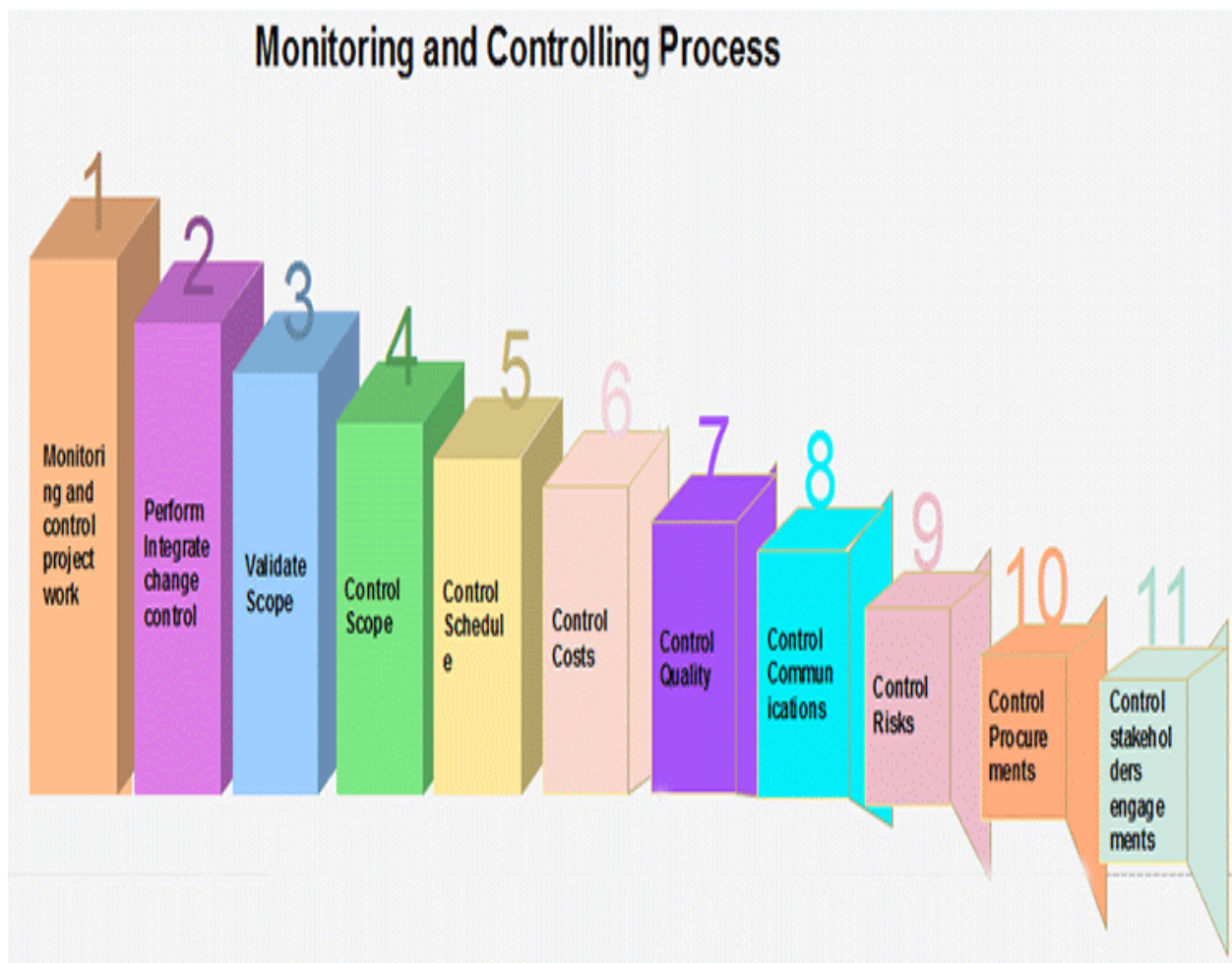
Quality Assurance v/s Quality control

Quality Assurance	Quality Control
Quality Assurance (QA) is the set of actions including facilitation, training, measurement, and analysis needed to provide adequate confidence that processes are established and continuously improved to produce products or services that conform to specifications and are fit for use.	Quality Control (QC) is described as the processes and methods used to compare product quality to requirements and applicable standards, and the actions are taken when a nonconformance is detected.
QA is an activity that establishes and calculates the processes that produce the product. If there is no process, there is no role for QA.	QC is an activity that demonstrates whether or not the product produced met standards.
QA helps establish process	QC relates to a particular product or service
QA sets up a measurement program to	QC verified whether particular

evaluate processes	attributes exist, or do not exist, in a explicit product or service.
QA identifies weakness in processes and improves them	QC identifies defects for the primary goals of correcting errors.
Quality Assurance is a managerial tool.	Quality Control is a corrective tool.
Verification is an example of QA.	Validation is an example of QC.

Monitoring and Controlling are processes needed to track, review, and regulate the progress and performance of the project. It also identifies any areas where changes to the project management method are required and initiates the required changes.

The Monitoring & Controlling process group includes eleven processes, which are:



1. **Monitor and control project work:** The generic step under which all other monitoring and controlling activities fall under.
2. **Perform integrated change control:** The functions involved in making changes to the project plan. When changes to the schedule, cost, or any other area of the project management plan are necessary, the program is changed and re-approved by the project sponsor.
3. **Validate scope:** The activities involved with gaining approval of the project's deliverables.
4. **Control scope:** Ensuring that the scope of the project does not change and that unauthorized activities are not performed as part of the plan (scope creep).
5. **Control schedule:** The functions involved with ensuring the project work is performed according to the schedule, and that project deadlines are met.
6. **Control costs:** The tasks involved with ensuring the project costs stay within the approved budget.
7. **Control quality:** Ensuring that the quality of the project's deliverables is to the standard defined in the project management plan.
8. **Control communications:** Providing for the communication needs of each project stakeholder.
9. **Control Risks:** Safeguarding the project from unexpected events that negatively impact the project's budget, schedule, stakeholder needs, or any other project success criteria.
10. **Control procurements:** Ensuring the project's subcontractors and vendors meet the project goals.
11. **Control stakeholder engagement:** The tasks involved with ensuring that all of the project's stakeholders are left satisfied with the project work.

Unit - IV

Software Implementation:-

Relationship between design and implementation

1. i. Analysis and design for an e-commerce site occur in an iterative fashion through prototyping. Here, the requirements of the organization and users of the system are defined and translated into a design from which the system can be built.
- ii. Analysis and design phase is followed by implementation phase, where writing the program, building the databases, testing the system, etc. occurs.
- iii. Following diagram illustrates the relationship between the three phases:

2. Diagram:

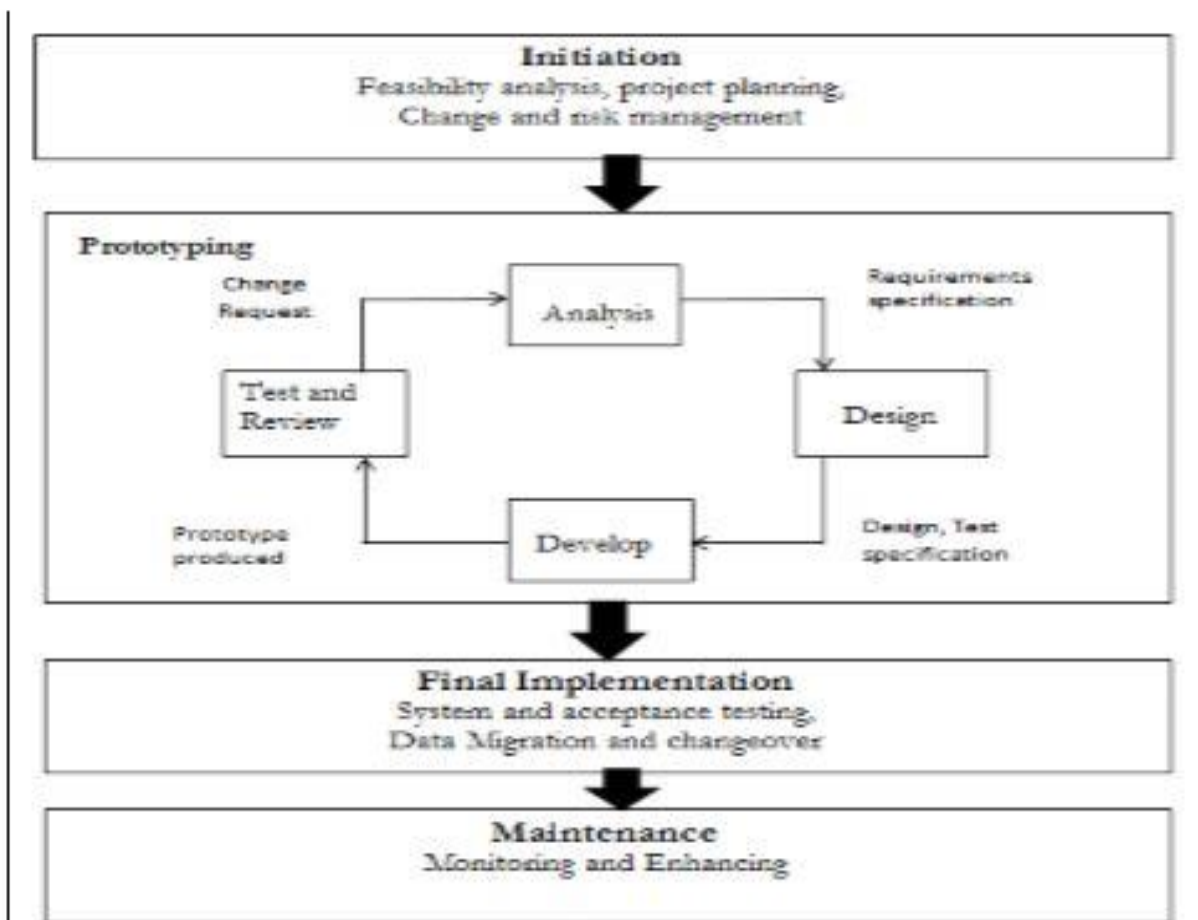


Figure 1: Stages in developing an e-business solution

1. Description:

- i. Analysis for e-business can be defined as “using analytical techniques to capture and summarize business and user requirements”.
- ii. Analysis consists of workflow management which is a key part for managing time based information flows. Workflow helps manage business processes by ensuring that the tasks are prioritized to be performed as soon as possible, by the right people in the right order.
- iii. Process modeling is an important section of analysis, which is used to analyse information flows to optimize business processes. Information storage analysis is done through data modeling.
- iv. Design phase basically specifies how the system should be structured or how an information system will operate.
- v. Various architectural design like Client-server model, three tier client servers, etc. are considered in this phase. The main target in this phase includes developing a user-centred design, i.e. optimizing user experience according to all factors.
- vi. In addition to this, information architecture, page design, security design etc. are considered here.
- vii. Implementation includes activities such as testing and review. These activities occur after the analysis and design during prototyping.
- viii. These activities also occur before the system becomes live in final implementation phase.
- ix. Consider the diagram given below which summarize the sequence of activities in the form of a Gantt chart.

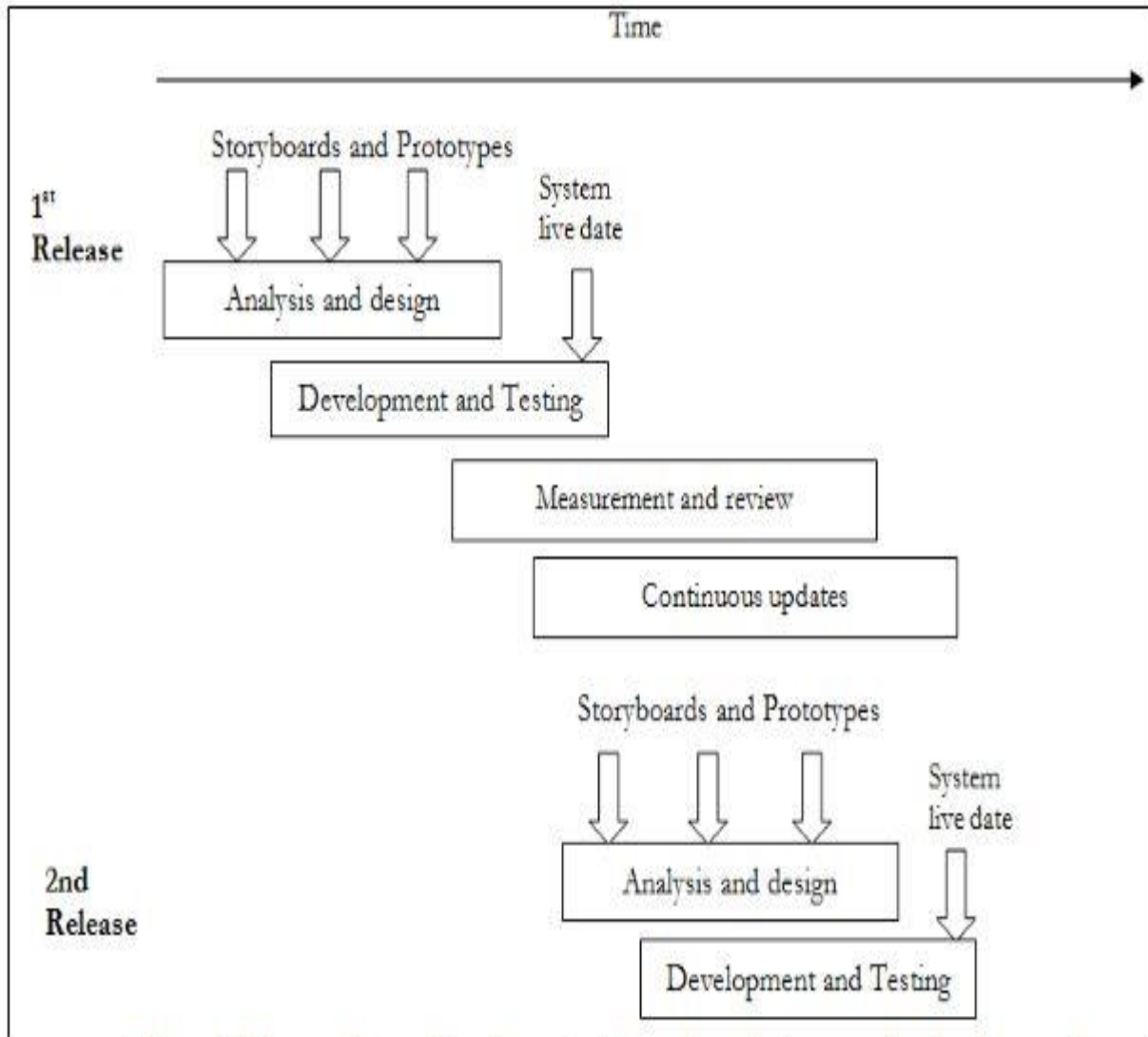


Figure 1: Sequencing of implementation and maintenance for the dynamic e-business application.

1. Although analysis, design, implementation and maintenance activities were considered as separate in systems development life cycle, but there is a great degree of overlap between these phases and all occur simultaneously as a part of evolutionary prototyping approach.
2. While analysis and requirements is in progress, design and implementation will be occurring simultaneously in order to produce storyboards and prototypes.
3. Once the system is live, measurement and review will commence. As soon as the system is live it will be necessary to make minor updates continuously to the content and services.

4. For each update, a small scale prototyping process involving analysis, design and testing will occur.
5. For major updates, another full scale of analysis and design, development and testing will be required.

Implementation issues and programming support environment

IMPLEMENTATION ISSUES

The mere presence of an implementation plan does not guarantee success. Most organizations do not have sufficient staff to cope with the commitment and extra work required when introducing a GIS to existing operations. GIS implementation must also consider all technology transfer processes.

Common Pitfalls

Several pitfalls exist that most often contribute to the failure of a GIS implementation strategy. These are identified below:

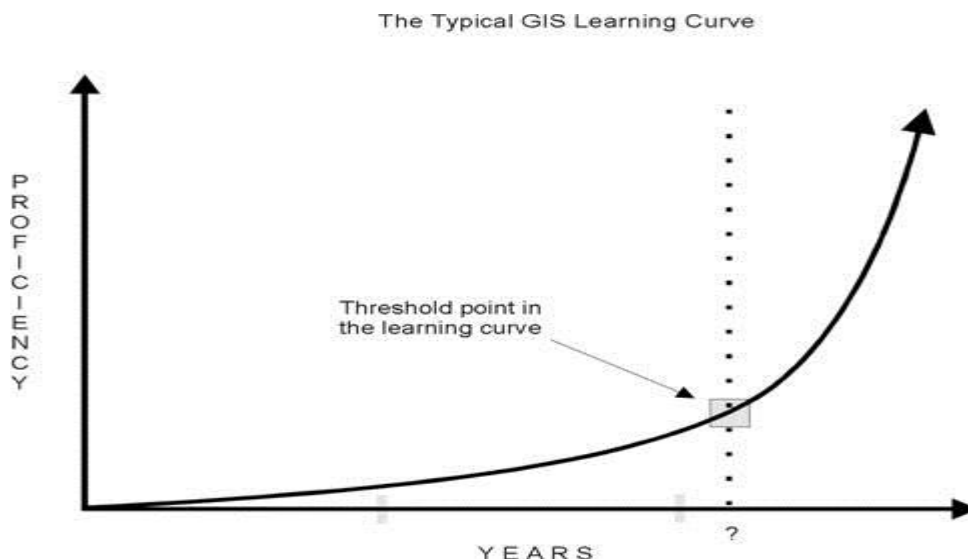
- **Failure to identify and involve all users** Users in an operational GIS environment consist of operations, management, and policy levels of the organization. All three levels should be considered when identifying the needs of your users.
- **Failure to match GIS capability and needs.** A wide spectrum of GIS hardware and software choices currently exist. The buyer is presented with a significant challenge making the right choice. Remember, the right choice will be the GIS that provides the needed performance no more, no less for the minimum investment. The success of a GIS implementation is particularly sensitive to the right hardware and software choices !
- **Failure to identify total costs.** The GIS acquisition cost is relatively easy to identify. However, it will represent a very small fraction of the total cost of implementing a GIS. Ongoing costs are substantial and include hardware and software maintenance, staffing, system administration, initial data loading, data updating, custom programming, and consulting fees.
- **Failure to conduct a pilot study** The GIS implementation plan concerns itself with the many technical and administrative issues and their related cost impacts. Three of the most crucial issues, are database design, data loading

and maintenance, and day-to-day operations. The pilot study will allow you to gather detailed observations, provided it is properly designed, to allow you to effectively estimate the operational requirements.

- ➔ **Giving the GIS implementation responsibility to the EDP Department.** Because of the distinct differences of the GIS from conventional EDP systems, the GIS implementation team is best staffed by non-data processing types. The specialized skills of the 'GIS analyst' are required at this stage. Reliance on conventional EDP personnel who lack these skills will ensure failure.
- ➔ **Failure to consider technology transfer.** Training and support for on-going learning, for in-house staff as well as new personnel, is essential for a successful implementation. Staff at the three levels should be educated with respect to the role of the GIS in the organization. Education and knowledge of the GIS can only be obtained through on-going learning exercises. Nothing can replace the investment of hands on time with a GIS !

The Learning Curve

Contrary to information provided by commercial vendors of GIS software, there is a substantial learning curve associated with GIS. It is normally not a technology that one becomes proficient in overnight. It requires an understanding of geographical relationships accompanied by committed hands-on time to fully apply the technology in a responsible and cost effective manner. Proficiency and productivity are only obtained through applied hands on with the system ! GIS is an applied science. Nothing can replace the investment of hands-on with GIS. The following figure represents the typical learning curve for GIS installations.



The learning curve is dependent on a variety of factors including:

- the amount of time spent by the individual with hands-on access;
- the skills, aptitude and motivation of the individual;
- the commitment and priority attached to GIS technology dictated by the organization and management;
- the availability of data; and
- the choice of software and hardware platforms.

A critical requirement for all GIS implementations is that adequate education and training is provided for operational staff, as well as realistic priorities are defined with which to learn and apply the technology. This is where a formal training curriculum is required to ensure that time is dedicated to learning the technology properly. Adding GIS activities to a staff member's responsibilities without establishing well defined milestones and providing adequate time and training mechanisms is prone to failure. A focused and properly trained operations staff that has consistent training will result in greatly reduced turnaround times for operations, and ensure consistency in quality of product.

The threshold point of the learning curve is typically around the two year time frame. However, this is dependent on the ability of the organization to establish a well defined and structured implementation plan that affords appropriate training and resources for technical staff. The flat part of the learning curve can be shortened if proper training is provided, data is available for use, the right software and hardware is acquired.

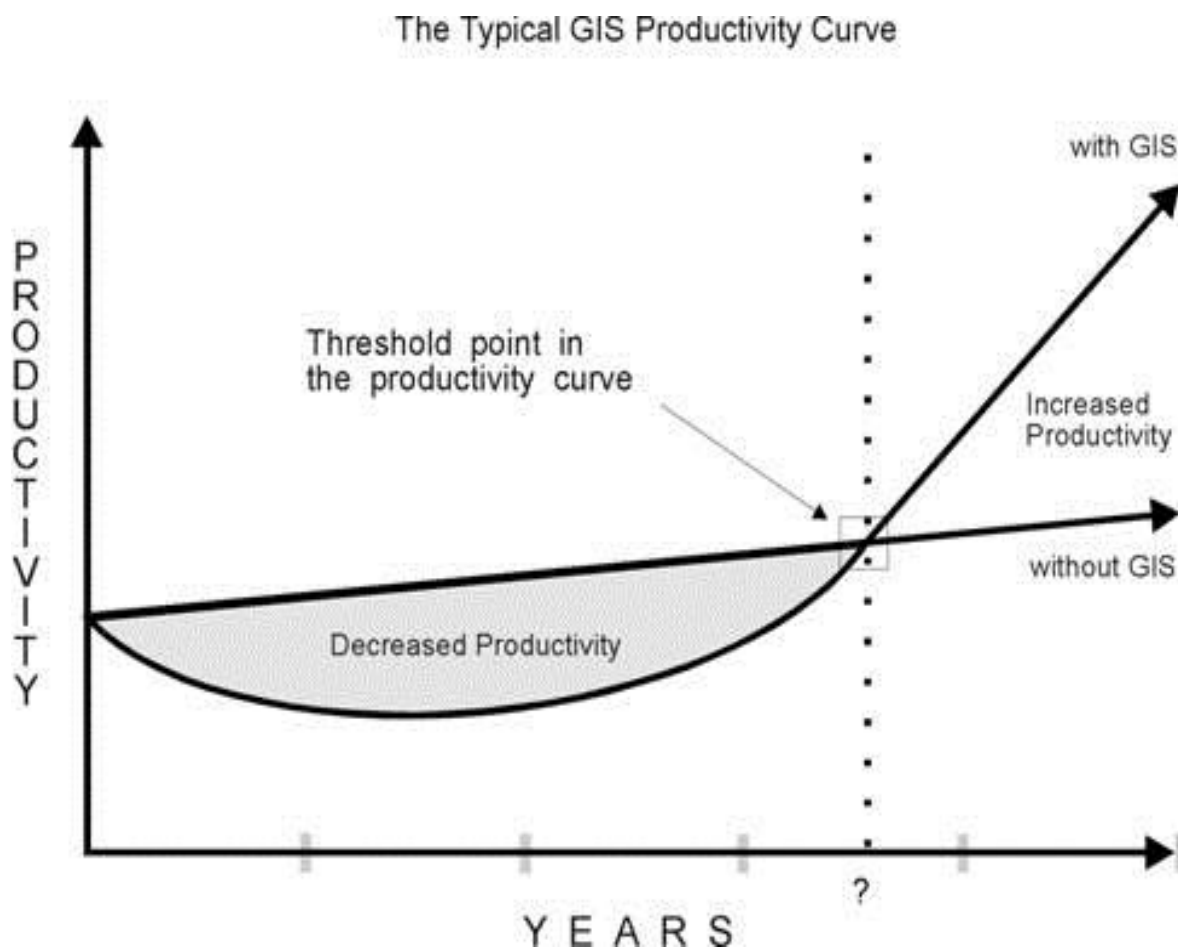
The typical learning curve reflects a long initial period for understanding spatial data compilation requirements and database architecture. However, after data models are well understood and sufficient data compilation has been completed the learning curve accelerates. Once a formal application development environment is established and user needs are well defined an infrastructure exists for effective application of the technology. Building operational applications based on formal functional specifications will result in continued accelerated learning. The data hurdle is often a stumbling block for many GIS users.

The Productivity Curve

GIS is a long term investment that matures over time. The turnaround for results may be longer than initially expected. The establishment of a formal implementation strategy will help to ensure that realistic expectations are met. Data is the framework for successful

application of GIS technology. In this respect, the investment in establishing a solid data platform will reap rewards in a short term timeframe for establishing a cost-effective and productive GIS operation. The availability of quality data supplemented by a planned implementation strategy are the cornerstones of achieving a productive and successful GIS operation. A robust database should be considered an asset !

However, even with a well defined and systematic implementation strategy GIS technology will not provide immediate benefits. Benefits and increased productivity are not achieved overnight. GIS technology is complex in nature, has a generally steep learning curve, and requires a complement of skills for it to be applied successfully. In fact, most organizations realize a loss in overall operational productivity over the short term while the GIS platforms are being installed, staff is trained, the learning curve is initiated, and data is being captured. This is common of all office automation activities. The following figure presents the typical situation that occurs with respect to comparing long term productivity with, and without, GIS technology.



This graph represents the typical situation that occurs with respect to comparing long term productivity with, and without, GIS technology.

Depending on the unique circumstances of the implementation process, the status of data compilation, and the organizational climate, increased productivity is normally reached between the second and fifth year of implementation. This is identified by the threshold point. Again, this is dependent on a variety of factors including:

- the skills and experience of the staff involved;
- the priority and commitment by the organization;
- the implementation strategy; and
- the status of data compilation.

The primary issue with implementing GIS is to achieve the threshold point of increased productivity in the shortest possible time frame. In other words, minimize the time in which a decrease in productivity occurs. Of course, the issue of productivity is typically of greater concern with private industry, e.g. forestry companies. Nonetheless, the significant investment in hardware/software, data, and training necessitates that a structured approach be utilized to achieve the threshold point in the shortest possible time frame.

A GIS acquisition based on well defined user needs and priorities is more likely to succeed than without. A major pitfall of most installations with GIS technology, e.g. particularly forestry companies and government agencies, is the lack of well defined user needs on which to base the GIS acquisition and implementation.

The Implementation Plan

Implementation can be seen as a six phase process. They are:

- **Creating an awareness** GIS needs to be sold within an organization. The education of staff is very important. Depending on the way in which GIS technology is being introduced to the organization the process for creating an awareness may differ. Technical workshops are often appropriate when a top-down approach exists, while management workshops are often more relevant when a bottoms-up approach exists. Education of the new technology should focus on identifying existing problems within an organization. These often help justify a GIS acquisition. They include :
 - spatial information is poorly maintained or out of date;
 - spatial data is not recorded or stored in a standard way;

- spatial data may not be defined in a consistent manner, e.g. different classifications for timber information;
 - data is not shared between departments within an organization;
 - data retrieval and manipulation capabilities are inadequate to meet existing needs; and
 - new demands are made on the organization that cannot be met with existing information systems.
- **Identifying System Requirements**

The definition of system requirements is usually done in a user needs analysis. A user needs analysis identifies users of a system and all information products required by those users. Often a prioritization of the information products and the data requirements of those products is also undertaken. A proper user needs analysis is crucial to the successful evaluation of GIS software alternatives.

After user needs have been identified and prioritized they must be translated into functional requirements. Ideally, the functional requirements definition will result in a set of processing functions, system capabilities, and hardware requirements, e.g. data storage, performance. Experienced GIS consultants often play a major role in this phase.

➤ **System Evaluations**

Evaluating alternative hardware and software solutions is normally conducted in several stages. Initially a number of candidate systems are identified. Information to support this process is acquired through demonstrations, vendor literature, etc. A short listing of candidates normally occurs based on a low level assessment. This followed by a high level assessment based on the functional requirements identified in the previous phase. This often results in a rating matrix or template. The assessment should take into account production priorities and their appropriate functional translation. After systems have been evaluated based on functional requirements a short list is prepared for those vendors deemed suitable. A standard benchmark, as discussed earlier, is then used to determine the system of choice.

➤ **Justifying the System Acquisition**

The proper justification of the chosen system requires consideration of several factors. Typically a cost-benefit analysis is undertaken to analyze the expected costs and benefits of acquiring a system. To proceed further with acquisition the GIS should provide considerable benefits over expected costs. It is important that the identification of intangible benefits also be considered.

The justification process should also include an evaluation of other requirements. These include data base development requirements, e.g. existing data versus new data needs and associated costs; technological needs, e.g. maintenance, training, and organizational requirements, e.g. new staff, reclassification of existing job descriptions for those staff who will use the GIS.

➔ **System Acquisition and Start Up**

After the system, e.g. hardware, software, and data, is acquired the start up phase begins. This phase should include pilot projects. Pilot projects are a valuable means of assessing progress and identifying problems early, before significant resources have been wasted. Also, because of the costs associated with implementing a GIS it is often appropriate to generate some results quickly to appease management. First impressions are often long remembered.

➔ **Operational Phase**

The operational phase of a GIS implementation involves the on-going maintenance, application, and development of the GIS. The issue of responsibility for the system and liability is critical. It is important that appropriate security and transaction control mechanisms be in place to support the system. A systematic approach to system management, e.g. hardware, software, and data, is essential.

Coding the procedural design

The coding is the process of transforming the design of a system into a computer language format. This coding phase of software development is concerned with software translating design specification into the source code. It is necessary to write source code & internal documentation so that conformance of the code to its specification can be easily verified.

Coding is done by the coder or programmers who are independent people than the designer. The goal is not to reduce the effort and cost of the coding phase, but to cut to the cost of a later stage. The cost of testing and maintenance can be significantly reduced with efficient coding.

Goals of Coding

1. **To translate the design of system into a computer language format:** The coding is the process of transforming the design of a system into a computer language format, which can be executed by a computer and that perform tasks as specified by the design of operation during the design phase.

2. **To reduce the cost of later phases:** The cost of testing and maintenance can be significantly reduced with efficient coding.
3. **Making the program more readable:** Program should be easy to read and understand. It increases code understanding having readability and understandability as a clear objective of the coding activity can itself help in producing more maintainable software.

For implementing our design into code, we require a high-level functional language. A programming language should have the following characteristics:

Characteristics of Programming Language

Following are the characteristics of Programming Language:

Characteristics of Programming Language



Readability: A good high-level language will allow programs to be written in some methods that resemble a quite-English description of the underlying functions. The coding may be done in an essentially self-documenting way.

Portability: High-level languages, being virtually machine-independent, should be easy to develop portable software.

Generality: Most high-level languages allow the writing of a vast collection of programs, thus relieving the programmer of the need to develop into an expert in many diverse languages.

Brevity: Language should have the ability to implement the algorithm with less amount of code. Programs mean in high-level languages are often significantly shorter than their low-level equivalents.

Error checking: A programmer is likely to make many errors in the development of a computer program. Many high-level languages invoke a lot of bugs checking both at compile-time and run-time.

Cost: The ultimate cost of a programming language is a task of many of its characteristics.

Quick translation: It should permit quick translation.

Efficiency: It should authorize the creation of an efficient object code.

Modularity: It is desirable that programs can be developed in the language as several separately compiled modules, with the appropriate structure for ensuring self-consistency among these modules.

Widely available: Language should be widely available, and it should be feasible to provide translators for all the major machines and all the primary operating systems.

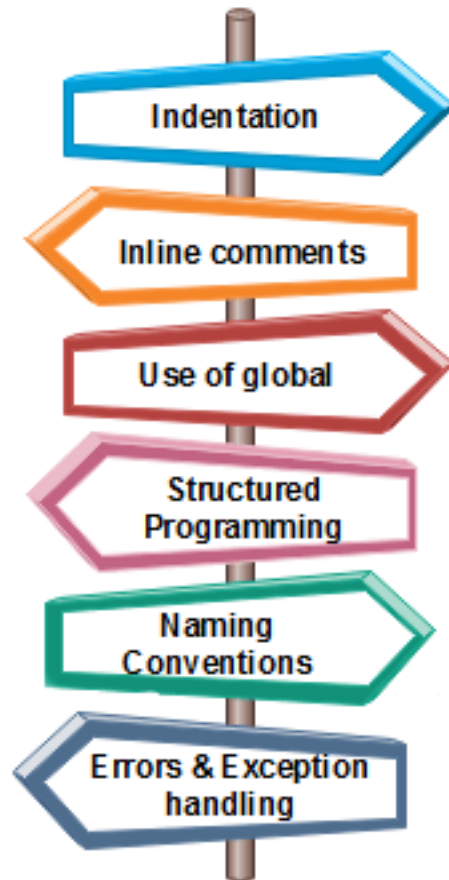
A coding standard lists several rules to be followed during coding, such as the way variables are to be named, the way the code is to be laid out, error return conventions, etc.

Coding Standards

General coding standards refers to how the developer writes code, so here we will discuss some essential standards regardless of the programming language being used.

The following are some representative coding standards:

Coding Standards



1. **Indentation:** Proper and consistent indentation is essential in producing easy to read and maintainable programs.

Indentation should be used to:

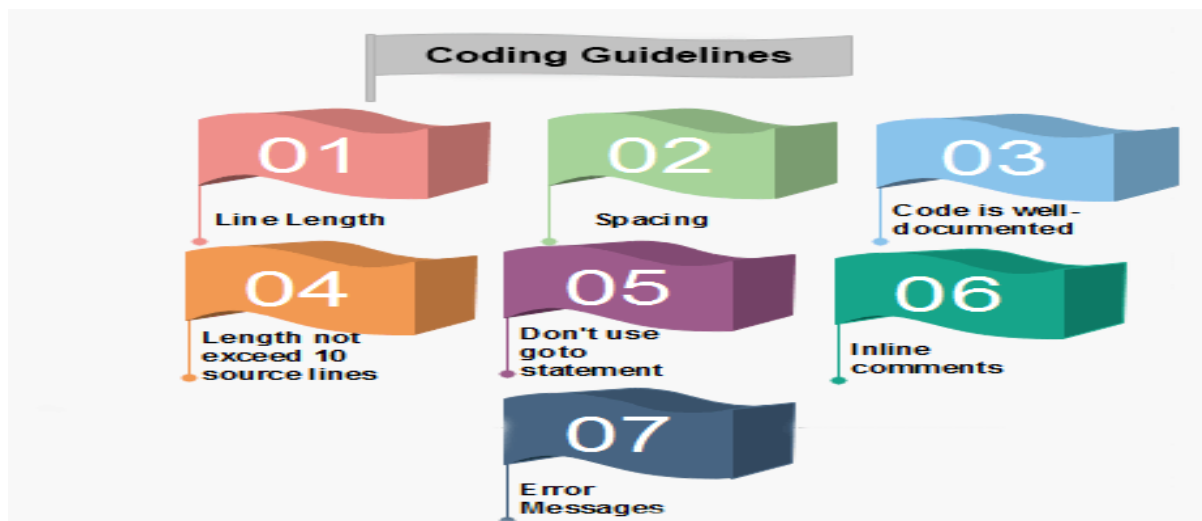
- Emphasize the body of a control structure such as a loop or a select statement.
- Emphasize the body of a conditional statement
- Emphasize a new scope block

2. **Inline comments:** Inline comments analyze the functioning of the subroutine, or key aspects of the algorithm shall be frequently used.
3. **Rules for limiting the use of global:** These rules file what types of data can be declared global and what cannot.
4. **Structured Programming:** Structured (or Modular) Programming methods shall be used. "GOTO" statements shall not be used as they lead to "spaghetti" code, which is hard to read and maintain, except as outlined line in the FORTRAN Standards and Guidelines.
5. **Naming conventions for global variables, local variables, and constant identifiers:** A possible naming convention can be that global variable names always begin with a capital letter, local variable names are made of small letters, and constant names are always capital letters.
6. **Error return conventions and exception handling system:** Different functions in a program report the way error conditions are handled should be standard within an organization. For example, different tasks while encountering an error condition should either return a 0 or 1 consistently.

Coding Guidelines

General coding guidelines provide the programmer with a set of the best methods which can be used to make programs more comfortable to read and maintain. Most of the examples use the C language syntax, but the guidelines can be tested to all languages.

The following are some representative coding guidelines recommended by many software development organizations.



1. Line Length: It is considered a good practice to keep the length of source code lines at or below 80 characters. Lines longer than this may not be visible properly on some terminals and tools. Some printers will truncate lines longer than 80 columns.

2. Spacing: The appropriate use of spaces within a line of code can improve readability.

Example:

Bad: `cost=price+(price*sales_tax)`
 `fprintf(stdout,"The total cost is %5.2f\n",cost);`

Better: `cost = price + (price * sales_tax)`
 `fprintf (stdout,"The total cost is %5.2f\n",cost);`

3. The code should be well-documented: As a rule of thumb, there must be at least one comment line on the average for every three-source line.

4. The length of any function should not exceed 10 source lines: A very lengthy function is generally very difficult to understand as it possibly carries out many various functions. For the same reason, lengthy functions are possible to have a disproportionately larger number of bugs.

5. Do not use goto statements: Use of goto statements makes a program unstructured and very tough to understand.

6. Inline Comments: Inline comments promote readability.

7. Error Messages: Error handling is an essential aspect of computer programming. This does not only include adding the necessary logic to test for and handle errors but also involves making error messages meaningful.

Good coding style & review of correctness and readability

Writing an efficient software code requires a thorough knowledge of programming. This knowledge can be implemented by following a coding style which comprises several guidelines that help in writing the software code efficiently and with minimum errors. These guidelines, known as **coding guidelines**, are used to implement individual programming language constructs, comments, formatting, and so on. These guidelines, if followed, help in preventing errors, controlling the complexity of the program, and increasing the readability and understandability of the program.

A set of comprehensive coding guidelines encompasses all aspects of code development. To ensure that all developers work in a harmonized manner (the source code should reflect a harmonized style as a single developer had written the entire

code in one session), the developers should be aware of the coding guidelines before starting a software project. Moreover, coding guidelines should state how to deal with the existing code when the software incorporates it or when maintenance is performed.

Since there are numerous programming languages for writing software codes, each having different features and capabilities, coding style guidelines differ from one language to another. However, there are some basic guidelines which are followed in all programming languages. These include naming conventions, commenting conventions, and formatting conventions.

There are certain rules for naming variables, functions and methods in the software code. These naming conventions help software; developers in understanding the use of a particular variable or function. The guidelines used to assign a name to any variable, function, and method are listed below.

- All the variables, functions, and methods should be assigned names that make the code more understandable to the reader. By using meaningful names, the code can be self-explanatory, thus, minimizing the effort of writing comments for variables. For example, if two variables are required to refer to 'sales tax' and 'income tax', they should be assigned names such as 'sales Tax' and 'income Tax'.
- For names, a full description in a commonly spoken language (for example, English) should be used. In addition, the use of abbreviations should be avoided. For example, variable names like 'contact Number' and 'address' should be used instead of 'cno' and 'add'.
- Short and clear names should be assigned in place of long names. For 'example, 'multiply The Two Numbers' can be shortened to 'multiply Numbers' as it is clear and short enough to be expressed in reasonable length.

In every programming language, there is a different naming convention for variables and constants in the software code. The commonly used conventions for naming variables and constants are listed in Table.

Table Naming Conventions for Variables and Constants

Variable Naming Conventions	<u>Constant Naming Conventions</u>
<u>The variable names should be in camel case letters starting with a lower case letter. For example, use 'total Amount' instead of 'Total Amount'.</u>	<u>All the names of constants should be in upper case. In case the name of constant is too long, it should be separated by an underscore. For example, sales tax rate should be written as 'SALES TAX RATE'.</u>
<u>The temporary storage variables that</u>	<u>The use of literal should be avoided.</u>

<p><u>are restricted to a segment of code should be short. For example, the variable 'temp' can be used for a temporary variable. It is important to note that a single temporary variable should not be reused in the same program. For example, variables 'i', 'j', or 'k' are declared while using loops.</u></p>	<p><u>Literal numbers such as '15' used in the software code confuses the reader. These numbers are counted as integers and result in wrong output of the program. However, the numbers '0' and '1' can be used as constants.</u></p>
<p><u>The use of numbers in naming variables should be avoided. For example, 'first Number' should be used instead of 'number1'.</u></p>	<p>-</p>

As with variables and constants, there are some guidelines that should be followed while naming functions in the software code. These conventions are listed below.

- The names of functions should be meaningful and should describe the purpose of the function with clarity and brevity. Like variables, the names should be self-explanatory so that no additional description about the task of that function is required.
- The function name should begin with a verb. For example, the verb 'display' can be used for the function that displays the output on the screen. In case the verb itself is not descriptive, an additional noun or adjective can be used with the verb. For example, the function name 'add Marks' should be used to clarify the function and its purpose.
- In case the function returns a Boolean value, the helping verbs 'is' and 'has' should be used as prefixes for the function name. For example, the function name 'is Deposited' or 'has Deposited' should be used for functions that return true or false values.
- Comments are helpful in proper understanding of the code segment used in program. Commenting conventions should be used efficiently to make the code easy to grasp. Generally, two types of commenting conventions are used: file header comments and trailing comments.
- **File header comments** are useful in providing information related to a file as a whole and comprise identification information such as date of creation, Name of the creator, and a brief description of the software code.

trailing comments are used to provide explanation of a single line of code. These comments are used to clarify the complex code. These also specify the function of the abbreviated variable names that are not clear. In some languages, trailing comments

are used with the help of a double slash (//). The commenting conventions that are commonly followed in the software code are listed below.

- Comments should not be used to include information that is clearly understandable from the software.
- Comments should be used with important segments of code and code segments that are difficult to understand.
- Comments should be separated from the code to enhance readability of the software code.
- Formatting (way of arranging a program in order to enhance readability) consists of indentation, alignment, and use of white spaces in the program. Consistency plays an important role while formatting a program in an organized way. A program with consistent formatting makes the code easier to read and understand. The commonly used formatting conventions are listed below. **Indentation:** This refers to one or more spaces left at the beginning of statements in the program. Indentation is useful in making the code easily readable. However, the spaces used for indentation should be followed in the entire program. The guidelines that are commonly followed while indenting a program are listed below. **White spaces:** These improve readability by minimizing the compactness of the code. Some of the guidelines for proper usage of spaces within the code are listed below.
 - Indentation should be used to highlight a nested block. Some nested blocks can be made with the help of 'if-else' and 'do-while' loops.
 - Indentation is required if the statement is large enough to fit in a single line.
 - Indentation should be consistent at the beginning and at the end of the braces in the program.
- There should be a space after placing a comma between two function arguments.
- There should be no space between a function name and parenthesis.
- There should be spaces to align the operators vertically to emphasize program structure and semantics.

We'll be covering the following topics in this tutorial:

- Implementing Coding Guidelines
- Advantages of Coding Guidelines

Implementing Coding Guidelines

If coding guidelines are used in a proper manner, errors can be detected at the time of writing the software code. Such detection in early stages helps in increasing the performance of the software as well as reducing the additional and unplanned costs of correcting and removing errors. Moreover, if a well-defined coding guideline is applied, the program yields a software system that is easy to comprehend and maintain. Some of the coding guidelines that are followed in a programming language are listed below.

- All the codes should be properly commented before being submitted to the review team.
- All curly braces should start from a new line.
- All class names should start with the abbreviation of each group. For example, AA and CM can be used instead of academic administration and course management, respectively.
- Errors should be mentioned in the following format: [error code]: [explanation]. For example, 0102: null pointer exception, where 0102 indicates the error code and null pointer exception is the name of the error.
- Every 'if' statement should be followed by a curly braces even if there exists only a single statement.
- Every file should contain information about the author of the file, modification date, and version information.

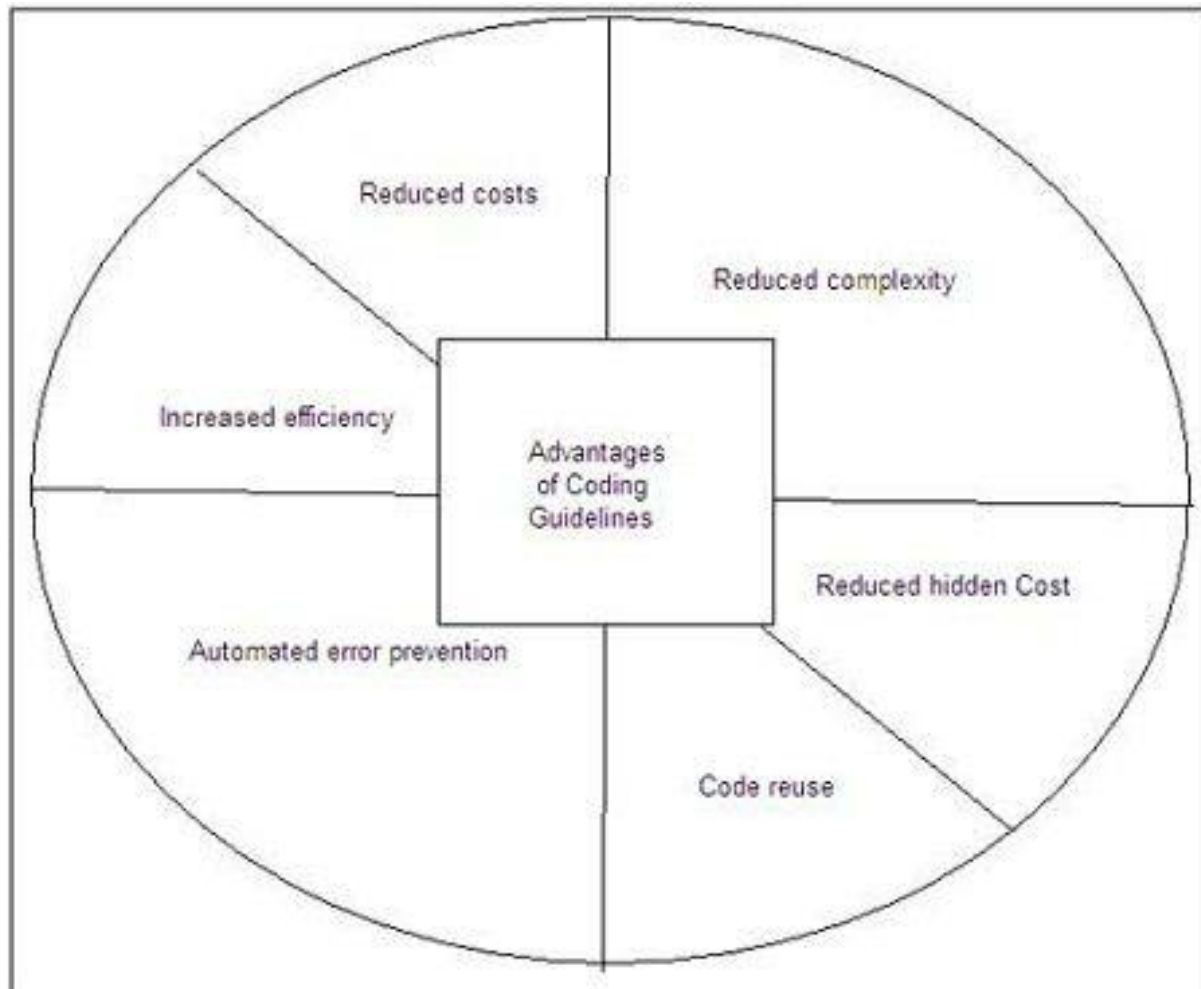
Similarly, some of the commonly used coding guidelines in a database (organized collection of information that is systematically organized for easy access and analysis) are listed below.

- Table names should start with TBL. For example, TBL_STUDENT.
- If table names contain one word, field names should start with the first three characters of the name of the table. For example, STU_FIRSTNAME.
- Every table should have a primary key.
- Long data type (or database equivalent) should be used for the primary key.

Advantages of Coding Guidelines

Coding guidelines supplement the language standard by defining acceptable and unacceptable usage of the programming language used. Acceptable usage avoids troublesome situations while unacceptable usage is conducive to errors or leads to misunderstanding of the written code. Properly implemented coding guidelines help the

developer to limit program complexity, establish the basis for code review, and guard against compiler and common programming errors. Other advantages associated with coding guidelines are listed below and depicted.



- **Increased efficiency:** Coding guidelines can be used effectively to save time spent on gathering unnecessary details. These guidelines increase the efficiency of the software team while the software development phase is carried out. An efficient software code is fast and economical. Software coding guidelines are used to increase efficiency by making the team productive, thus, ensuring that the software is delivered to the user on time.
- **Reduced costs:** Coding guidelines are beneficial in reducing the cost incurred on the software project. This is possible since coding guidelines help in detecting errors in the early stages of the software development. Note that if errors are discovered after the software is delivered to the user, the process of rectifying them becomes expensive as additional costs are incurred on late detection, rework, and retesting of the entire software code.

- **Reduced complexity:** The written software code can be either simple or complex. Generally, it is observed that a complex segment of software code is more susceptible to errors than a segment containing a simple software code. This is because a complex software code reduces readability as well as understandability. In addition, the complex software code may be inefficient in functioning and use of resources. However, if the code is written using the given coding guidelines, the problem of complexity can be significantly avoided as the probability of error occurrence reduces substantially.
- **Reduced hidden costs:** Coding guidelines, if adhered to in a proper manner, help to achieve a high-quality software code. The software quality determines the efficiency of the software. Software quality is the degree to which user requirements are accomplished in the software along with conformity to standards. Note that if quality is not considered while developing the software, the cost for activities such as fixing errors, redesigning the software, and providing technical support increases considerably.
- **Code reuse:** Using coding guidelines, software developers are able to write a code that is more robust and create individual modules of the software code. The reason for making separate code segment is to enable reusability of the modules used in the software. A reusable module can be used a number of times in different modules in one or more software.
- **Automated error prevention:** The coding guidelines enable Automated Error Prevention (AEP). This assures that each time error occurs in software, the software development activity is improved to prevent similar errors in future. AEP begins with detecting errors in the software, isolating its cause, and then searching the cause of error generation. Coding guidelines are useful in preventing errors as they allow implementation of requirements that prevent the most common and damaging errors in the software code.

In addition to the above mentioned advantages, coding guidelines define appropriate metric thresholds. These thresholds help in reducing complexity, thus, minimizing the occurrence of errors. Software developers face increasing demands to demonstrate that development practices meet the accepted coding guidelines. This is essential for companies developing safety-critical software as well as those seeking CMM and ISO certification.

Unit - V

Software Maintenance:-

Maintenance as part of software evaluation

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 for modification.
- **Client Requirements** - Over the time, customer may ask for new features or functions in the software.
- **Host Modifications** - If any of the hardware and/or platform (such as operating system) of the target host changes, software changes are needed to keep adaptability.
- **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 may arise.

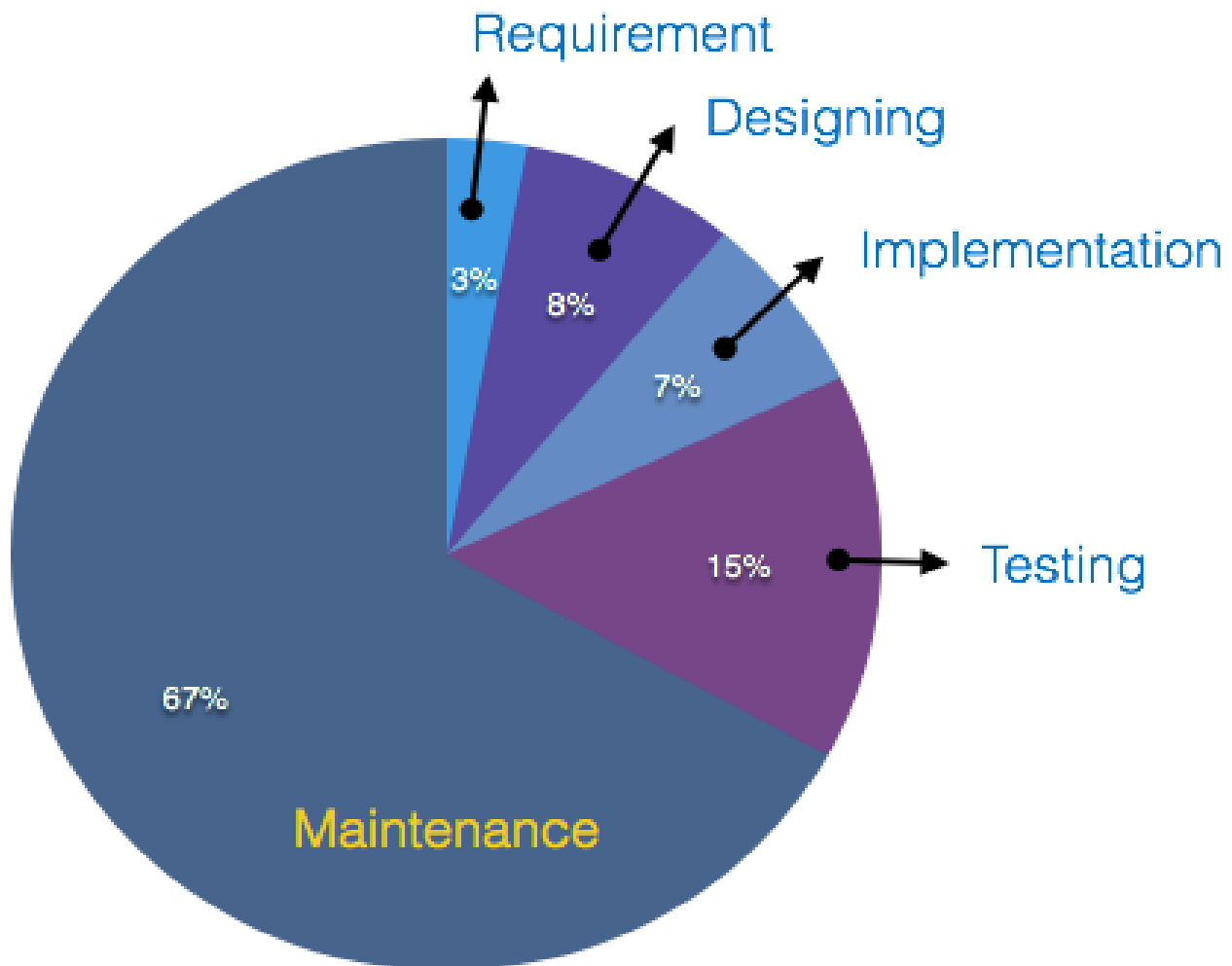
Types of maintenance

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. Following are some types of maintenance based on their characteristics:

- **Corrective Maintenance** - This includes modifications and updations done in order to correct or fix problems, which are either discovered by user or concluded by user error reports.
- **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 business environment.
- **Perfective Maintenance** - This includes modifications and updates done in order to keep the software usable over long period of time. It includes new features, new user requirements for refining the software and improve its reliability and performance.
- **Preventive Maintenance** - This includes modifications and updations to prevent future problems of the software. It aims to attend problems, which are not significant at this moment but may cause serious issues in future.

Cost of Maintenance

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 15 years.
- 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 modern hardware.

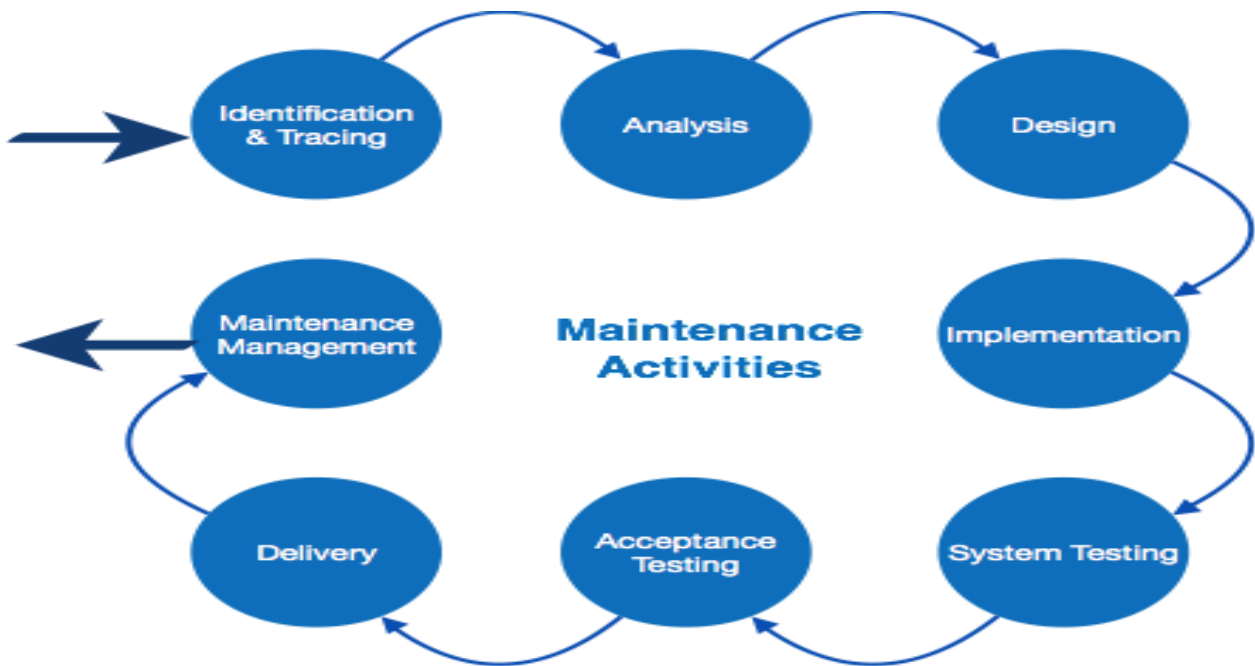
- As technology advances, it becomes costly to maintain old software.
- Most maintenance engineers are newbie and use trial and error method to rectify problem.
- Often, changes made can easily hurt the original structure of the software, making it hard for any subsequent changes.
- Changes are often left undocumented which may cause more conflicts in future.

Software-end factors affecting Maintenance Cost

- Structure of Software Program
- Programming Language
- Dependence on external environment
- Staff reliability and availability

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** - It involves activities pertaining to identification of requirement of modification or maintenance. It is generated by user or system may itself report via logs or error messages. Here, the maintenance type is classified also.
- **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 is concluded.
- **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 and verification.
- **Implementation** - The new modules are coded with the help of structured design created in the design step. Every programmer is expected to do unit 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 testing procedures.
- **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 next iteration.
- **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 part of system maintenance. It is aided with version control tools to control versions, semi-version or patch management.

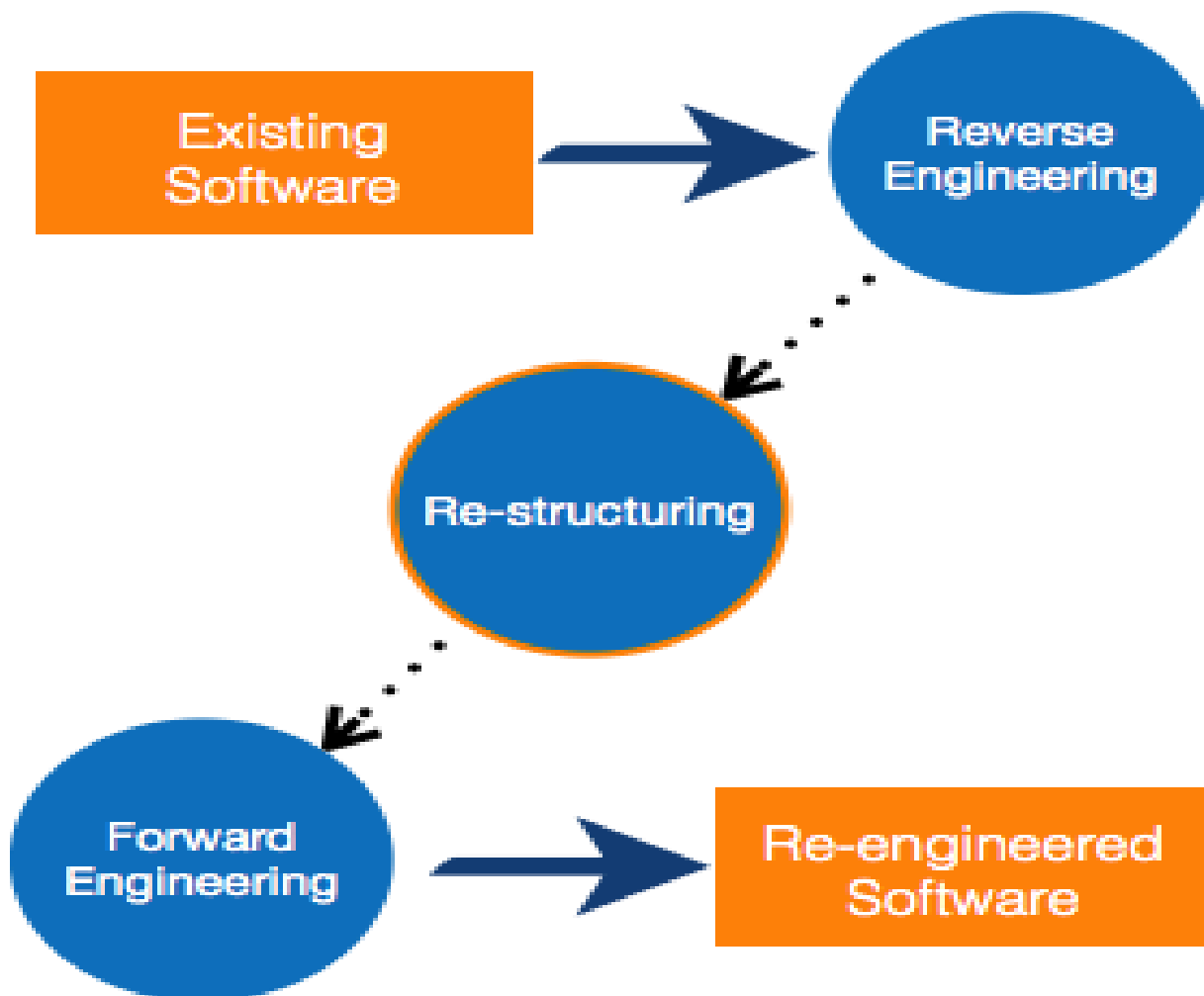
Software Re-engineering

When we need to update the software to keep it to the current market, without impacting its functionality, it is called software re-engineering. It is a thorough process where the design of software is changed and programs are re-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 C came into existence, Unix was re-engineered in C, because working in assembly language was difficult.

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

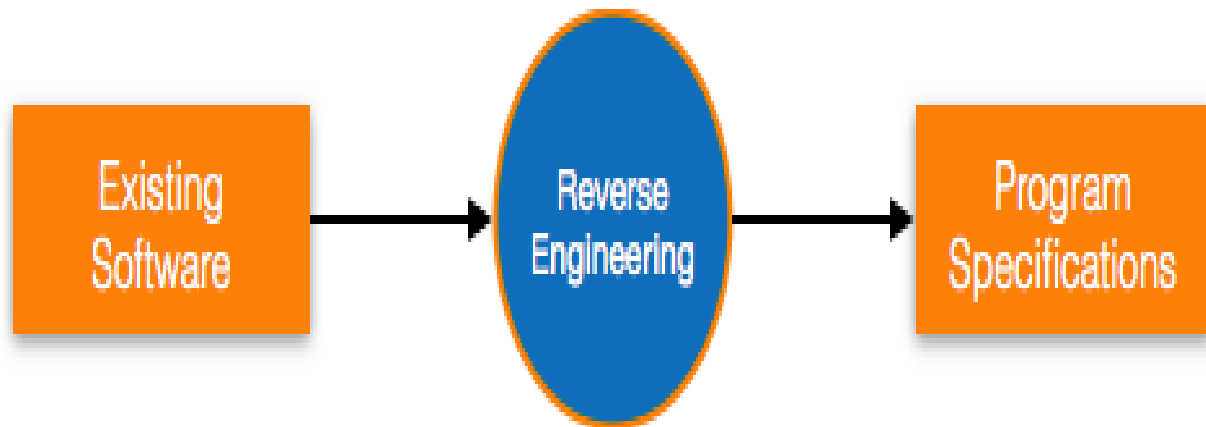
- **Decide** what to re-engineer. Is it whole software or a part of it?
- **Perform** Reverse Engineering, in order to obtain specifications of existing software.
- **Restructure Program** if required. For example, changing function-oriented programs into object-oriented programs.
- **Re-structure data** as required.
- **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 get the design. With design in hand, they try to conclude the specifications. Thus, going in reverse from code to system specification.



Program Restructuring

It is a process to re-structure and re-construct the existing software. It is all about 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 or both.

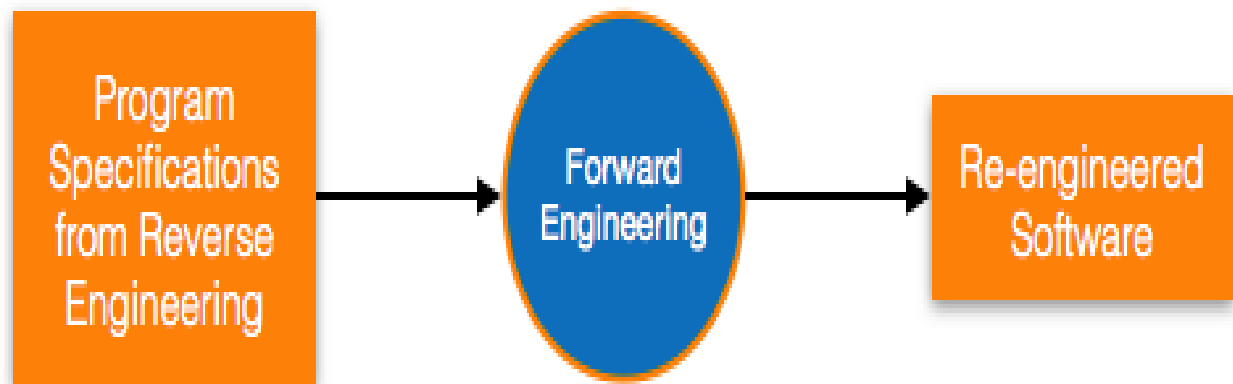
Re-structuring does not impact the functionality of the software but enhance reliability and maintainability. Program components, which cause errors very frequently can be changed, or updated with re-structuring.

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 specifications in hand which were brought down by means of reverse engineering. It assumes that there was some software engineering already done in the past.

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

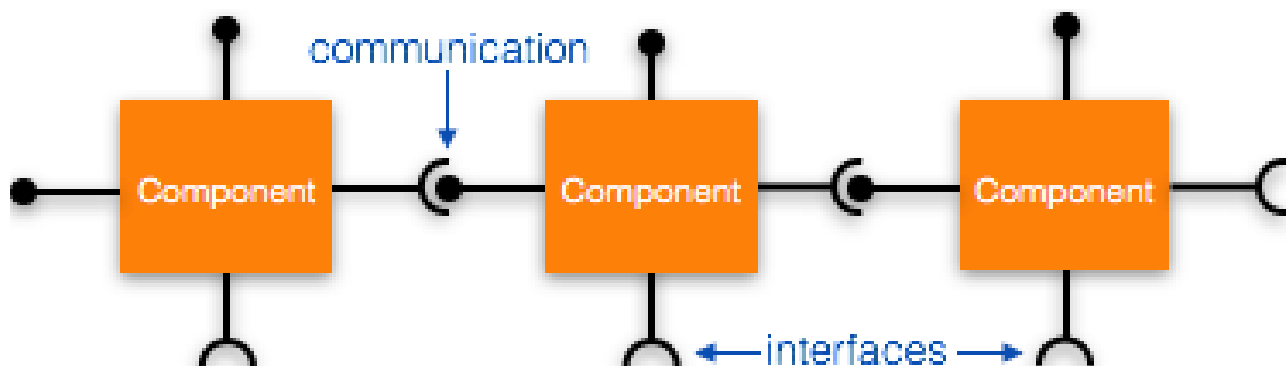
The login procedures used on the web can be considered as components, printing system in software can be seen as a component of the software.

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.

In OOP, the objects are designed are very specific to their concern and have fewer chances to be used in some other software.

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).



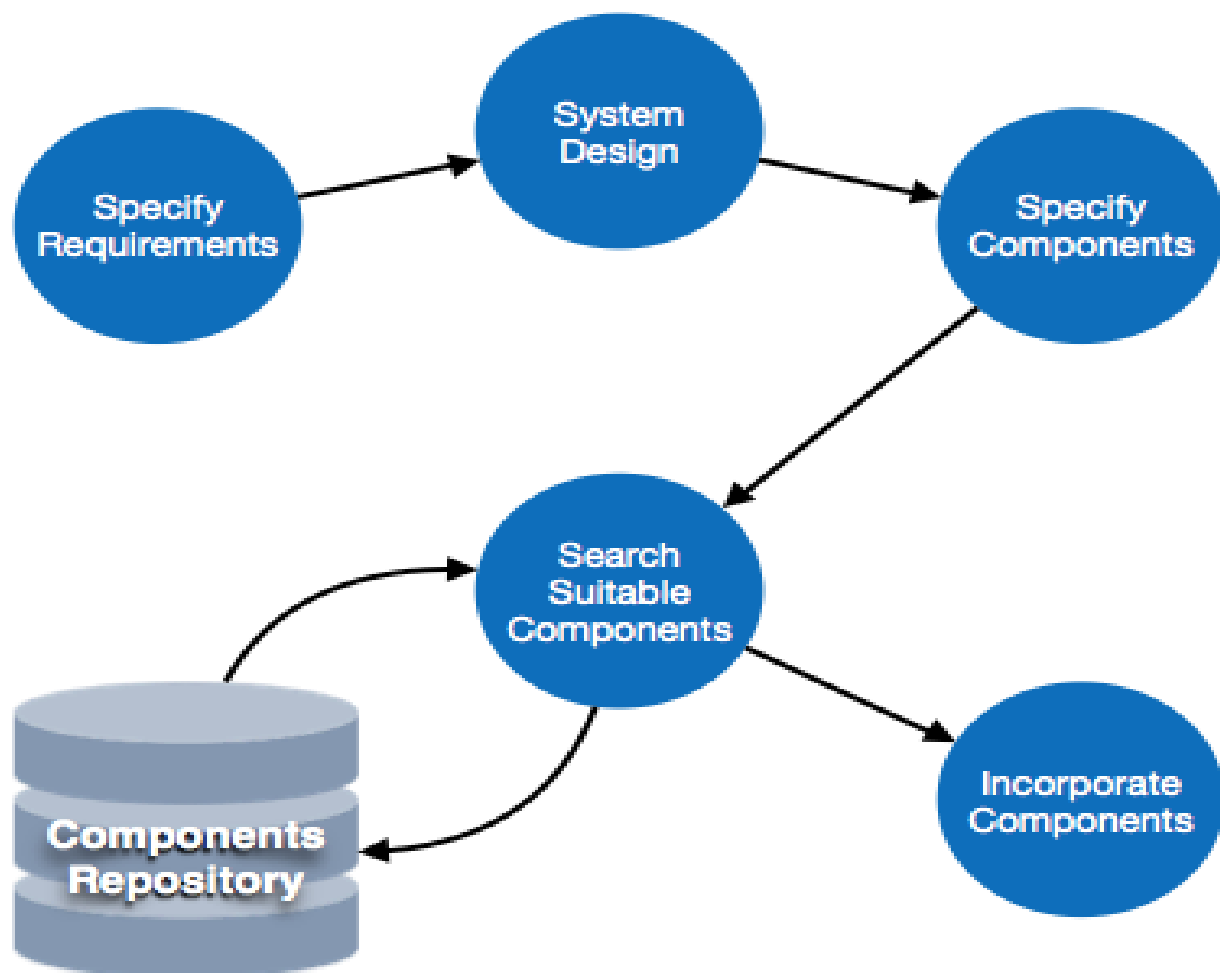
Re-use can be done at various levels

- **Application level** - Where an entire application is used as sub-system of new software.
- **Component level** - Where sub-system of an application is used.
- **Modules level** - Where functional modules are re-used.

Software components provide interfaces, which can be used to establish communication among different components.

Reuse Process

Two kinds of method 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 or both.
- **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 are created.
- **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 working together.
- **Search Suitable Components** - The software component repository is referred by designers to search for the matching component, on the basis of functionality and intended software requirements..
- **Incorporate Components** - All matched components are packed together to shape them as complete software.

reasons for maintenance

Maintaining a system is equally important as Web Application Development. It keeps solutions healthy to deal with changing technical and business environment. Generally, IT service providers suggest their clients to go for software maintenance services for the consistent and enhanced performance of the system. As per Mr. Robert Glass, writer of the 'Facts and Fallacies of Software Engineering', when it comes to software, 60% costing is for maintenance. Even from total maintenance costing, 60% is for solution enhancement.

IT is one of the most updated industry domains. It introduces technical advancements almost every day that improve solution efficiency to streamline business operations. Under the maintenance management, system is updated with latest technologies on the regular basis and run seamlessly with high-end efficiency. Sometimes system maintenance involves improvements in the existing solution and at times there are requirements of new development as per the changing market needs.

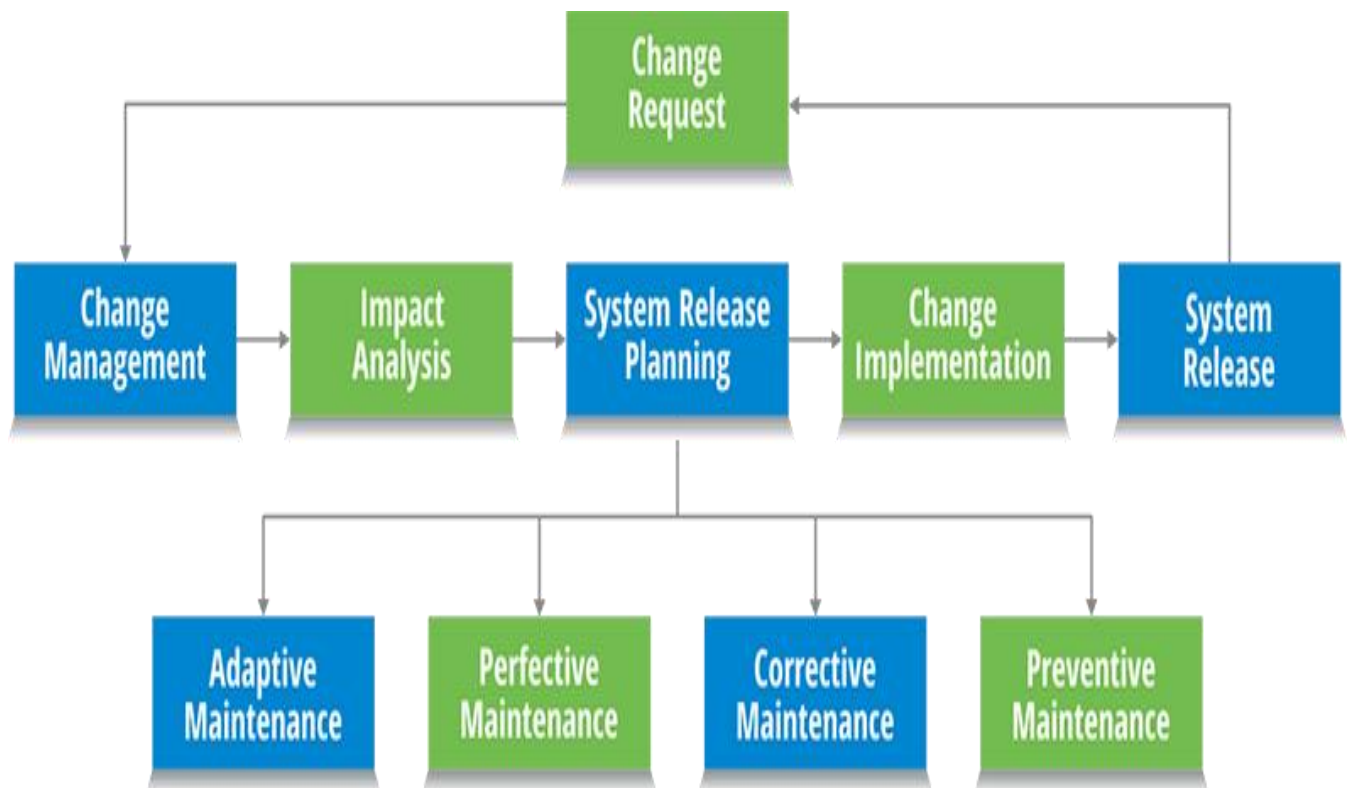
What is Software Maintenance?

It is a very broad activity that takes place once the operation is done. It optimizes the solution performance by reducing errors, eliminating useless development and applying advanced development. Solution development can take 1-2 years to build a system while software maintenance management can be an ongoing activity for 15-20 years.

Software Maintenance Services Categories:

1. Adaptive – Modifications in system to keep it compatible with changing business and technical environment.
2. Perfective – Fine tuning of all elements, functionalities and abilities to improve system operations and perfectness.
3. Corrective – Detecting errors in the existing solution and correcting them to make it works more efficiently.
4. Preventive – Preventive software maintenance services help in preventing the system from any upcoming vulnerabilities.

Maintenance Process



Why Software Requires Maintenance?

1. Bug Fixing

In maintenance management, bug fixing comes at priority to run the software seamlessly. This process contains search out for errors in code and correct them. The

issues can be occurred in hardware, operating systems or any part of software. This must be done without hurting rest of the functionalities of existing software.

2. Capability Enhancement

This comprises improvement in features and functions to make solution compatible with varying market environment. It enhances software platforms, work pattern, hardware upgrade, compilers and all other aspects that affect system workflow. Boost your business using a technically updated solution applying software maintenance services regularly.

3. Removal of Outdated Functions

The unwanted functionalities are useless. Moreover, by occupying space in solution, they hurt efficiency of the solution. Using software maintenance procedure, such elements of UI and coding are removed and replaced with new development using the latest tools and technologies. This elimination makes the system adaptive to cope with changing circumstances.

4. Performance Improvement

To improve system performance, developers detect issues through testing and resolve them. Data and coding restricting as well as reengineering are the part of software maintenance. It prevents the solution from vulnerabilities. This is not any functionality that performs in operations, but it develops to stop harmful activities like hacking.

Thus, software maintenance services keep the solution hale and hearty. The experienced developers offer reliable and authenticated maintenance management applying modern technologies. Contact us today to know more about our offerings.

types of maintenance (Perceptive, adoptive, corrective)

The life of your software does not end when it finally launches. In reality, the software maintenance has just begun.

Software is always evolving and it is never finished as long as it is used; partly to accommodate for the ever changing world we live in. The evolution of your software might be motivated by a variety of reasons; to keep the software up and running, upgrade to the latest release, enhance features or to rework the system for future maintainability. No matter the motivation, software maintenance is vital for the evolution

and success of it. Investing in any project should encompass a reassuring process from the software development company that is inclusive of feedback and modifications every step of way. This process should help clients understand it will have to undergo changes if proper functionality is desired. Understanding the different types of changes your software can go through is important to realize that software maintenance is more than just bug fixing. In fact, a study suggests that over 80% of software change is attributed to non bug related changes.

There are four categories of software change:

- Corrective
- Adaptive
- Perfective
- Preventive

Corrective Change

Corrective change, most commonly referred to as “bugs,” is the most typical change associated with maintenance work. Corrective changes address errors and faults in your software that could affect various areas of your software; design, logic or code. Most commonly, these changes are sprung by bug reports created by users. It is important to note that sometimes problem reports submitted by users are actually enhancements of the system not bugs.

Adaptive Change

Adaptive change is triggered by changes in the environment your software lives in. An adaptive change can be triggered by changes to the operating system, hardware, software dependencies and even organizational business rules and policies. These modifications to the environment can trigger changes within other parts of your software. For example, updating the server, compilers, etc or modifications to shipping carriers and payment processors can affect functionality in your software.

Perfective Change

Perfective changes refers to the evolution of requirements and features in your existing system. As your software gets exposed to users they will think of different ways to expand the system or suggest new features that they would like to see as part of the software, which in turn can become future enhancements to the system. Perfective changes also includes removing features from a system that are not effective and functional to the end goal of the system. Surprisingly, 50-55% of most maintenance work is attributed to perfective changes.

Preventive Change

Preventive changes refer to changes made to increase the understanding and maintainability of your software in the long run. Preventive changes are focused in decreasing the deterioration of your software in the long run. Restructuring, optimizing code and updating documentation are common preventive changes. Executing preventive changes reduces the amount of unpredictable effects a software can have in the long term and helps it become scalable, stable, understandable and maintainable.

Conclusion

Software Maintenance is an essential part of the software development life cycle; it is necessary for the success and evolution of your system. Maintenance on software goes beyond fixing “bugs”, which is one of the four types of software change. Updating the software environment, reducing its deterioration over time, and enhancing features to satisfy user needs are all examples of maintenance work. Next time you think about maintenance and software change keep in mind that it is much more than “bug” fixing.

designing for maintainability

The first time I changed the oil filter of my car (first car in high school) I smashed my knuckles against a grimy block of metal. More than once. It may have been my lack of experience or improper tools. Or, it may have been a rather poor design.

Watching the Indy 500, especially the pit stops, I quickly realized there has to be something different about those cars that lets them change tires, add oil & fuel, and clean the windshield in less time than it took me to get under the hood and find the oil filter. The cars were designed differently and that is what permitted the difference in service time.

In order to increase availability and minimize the cost of maintenance, we have to deliberately design the system to accommodate the needs of maintenance.

Here are 8 factors to consider when designing a system that will require maintenance.

1. Standardization

Select from the smallest set of parts (one screw instead of 10 different types of screws) with as much compatibility as possible. Minimize spare parts inventory is just one benefit.

Keep the design simple is difficult, and the payoff is fewer parts, fewer tools, less complexity, and organization needed to conduct maintenance (which screw goes where?).

2. Modularization

Create a set of standard sizes, shapes, modular units. Lego bricks come to mind.

If we expect to different models with different features, using a standard structure allows the interchange of compatible parts to alter functionally without changing the majority of the product. A good example is light bulbs. You can select the functional bulbs (brightness, intensity, color, etc.) and they will fit in the same socket.

3. Functional packaging

Gather all the required elements to complete a maintenance task in one kit. If I need washers, o-rings, and pumper's grease to complete a faucet repair, having all the items in one package helps me complete the task quickly (without the need to run to the store to pick up the forgotten item.)

4. Interchangeability

If you have to create a custom fit for a part, consider the ramifications. Single source, lack of compatibility with other similar functioning parts, another spare part in inventory, and limitations on future design changes if you want to stay in that custom form factor.

Select parts that are useful for a range of products or applications. Manage and control the dimensional and functional design tolerances.

5. Accessibility

Bruised knuckles are one risk of getting this wrong.

If an item requires replacement or adjustment as part of the expected maintenance, then it should permit access. Consider tools, lighting, environment, and experience of a maintenance crew. Providing access panels is one factor, safety another.

6. Malfunction annunciation

A key step in performing maintenance is to know what caused the problem or which parts are damaged and require replacement.

A bicycle flat tire is obvious to visual inspection or you may notice a change in the sound and feel of the ride. On complex systems which circuit board requires replacement may not be obvious. Minimizing the need for inspection tools and diagnostic tasks minimizes the time/cost of the corrective maintenance tasks. Let the system inform the technician what requires attention.

7. Fault isolation

There are two parts to this factor. One, make the system as informative as possible such that it not only signals a failure mode, it also narrows down the possible failure mechanisms. Replacing a blown fuse doesn't fix the problem and just finding the problem may take significant time.

Second, a failure in one part of a system can cause failure of other elements in the system.

When possible, contain the damage to minimize the amount of damage caused by a failure of one item.

8. Identification

Name the parts with unique identifiers. This streamlines documentation, procedures, and maintenance tasks.

Be consistent and provide meaningful or memorable naming conventions to avoid confusion.

Summary

There are always the considerations of time, complexity, cost and functionality, in a design. Considering these factors during the design process provides a meaningful basis to balance the needs of maintenance as we attempt to restore a system to service.

The cost of ownership is a function of main tenability and during the design process, you have the ability to minimize the number of bruised knuckles that occur.

techniques for maintenance

Software maintenance is a part of Software Development Life Cycle. Its main purpose is to modify and update software application after delivery to correct faults and to improve performance. Software is a model of the real world. When the real world changes, the software requires alteration wherever possible.

Description: Software maintenance is a vast activity which includes optimization, error correction, deletion of discarded features and enhancement of existing features. Since these changes are necessary, a mechanism must be created for estimation, controlling and making modifications. The essential part of software maintenance requires preparation of an accurate plan during the development cycle. Typically, maintenance takes up about 40-80% of the project cost, usually closer to the higher pole. Hence, a focus on maintenance definitely helps keep costs down.

Software Maintenance Processes are:

- The SM process includes a maintenance plan which contains software preparation, problem identification and find out about product configuration management.
- The problem analysis process includes checking validity, examining it and coming up with a solution and finally getting all the required support to apply for modification.
- The process acceptance by confirming the changes with the individual who raised the request.
- The platform migration process, which is used if software is needed to be ported to another platform without any change in functionality.

Some software points that affect maintenance cost include:

- Structure of Software Program
- Programming Language

Comprehensive examples using available software platforms/case tools

he term CASE was originally coined by software company, Nastec Corporation of Southfield, Michigan in 1982 with their original integrated graphics and text editor GraphiText, which also was the first microcomputer-based system to use hyperlinks to

cross-reference text strings in documents — an early forerunner of today's web page link. GraphiText's successor product, DesignAid was the first microprocessor-based tool to logically and semantically evaluate software and system design diagrams and build a data dictionary. The next entrant into the market was Excelerator from Index Technology in Cambridge, Mass. While DesignAid ran on Convergent Technologies and later Burroughs Ngen networked microcomputers, Index launched Excelerator on the IBM PC/AT platform. While, at the time of launch, and for several years, the IBM platform did not support networking or a centralized database as did the Convergent Technologies or Burroughs machines, the allure of IBM was strong, and Excelerator came to prominence. Hot on the heels of Excelerator were a rash of offerings from companies such as Knowledgeware, Texas Instrument's IEF and Accenture's FOUNDATION toolset (METHOD/1, DESIGN/1, INSTALL/1, FCP)

Introduction

Computer-Aided Software Engineering (CASE) technologies are tools that provide automated assistance for software development . The goal of introducing CASE tools is the reduction of the time and cost of software development and the enhancement of the quality of the systems developed. The interest in CASE tools and environments is based on expectations about increasing productivity, improving product quality, facilitating maintenance, and making software engineers' task less odious and more enjoyable. A survey of the CASE tool market showed that the annual worldwide market for CASE tools was \$4.8 billion in 1990 and grew to \$12.11 billion in 1995. Behind such a prosperous CASE market, however, another result gained from the real investigation about the use of CASE tools revealed that CASE tools seem to be sparsely used after being bought in many enterprises.

CASE is the use of computer-based support in the software development process; a CASE tool is a computer-based product aimed at supporting one or more software engineering activities within a software development process; a CASE environment is a collection of CASE tools and other components together with an integration approach that supports most or all of the interactions that occur among the environment components, and between the users of the environment and the environment itself.

Are CASE Tools being used?

Many prior studies have reported limited use of CASE tools. In a survey of 53 companies, found that 39 (73.5%) had never used CASE. Of the 14 companies who had tried CASE, five had subsequently abandoned use of the tools. People within these fourteen companies believed that use of CASE tools improved documentation quality, improved analysis, and resulted in systems that were easier to test and maintain. However, they also found use of CASE tools difficult and time consuming. In another cross organization survey, found that only 24% of companies were using CASE tools. In a follow-up survey of thirteen managers who had been using CASE tools two years earlier, reported that continued CASE use could only be verified for four managers. The

reasons for abandonment included cost, lack of measurable turns, and unrealistic expectations. Looking within organizations that used CASE tools it was found that large numbers of their systems developers were not using CASE tools.

Popular features of CASE tools

The term Computer-Aided Software Engineering (CASE) encompasses many different products with different functionalities. In the International Workshop on Computer-Aided Software Engineering (IWCASE) definition of CASE very broad terms are used: "...tools and methods to support engineering approach to systems development at all stages of the process". When the term CASE is used, it is important to clarify what is being discussed. Most classifications of CASE tools start by considering whether the tool is upper CASE, lower CASE, or integrated CASE [3]. An upper CASE tool (front end CASE) provides support for the early stages in the systems development life cycle such as requirements analysis and design. A lower CASE tool (back end CASE) provides support for the later stages in the life cycle such as code generation and testing. Integrated CASE tools support both the early and later stages. Further classifications usually list which functionalities are supported by the tool, such as data flow diagrams, entity relationships data models, etc. provides a different type of model of CASE functionality which helps organize CASE tools.

Automated Diagram Support

CASE Tools offer an excellent array of features that support the development and business community through its Automated Diagram Support feature. The various popular features that aid the development community are listed below:

- Checks for syntactic correctness
- Data dictionary support
- Checks for consistency and completeness
- Navigation to linked diagrams
- Layering
- Requirements traceability
- Automatic report generation
- System simulation
- Performance analysis

CASE Tools and its scope

CASE technology is the automation of step by step methodologies for software and system development. CASE tools are characterized by the stage or stages of software development life cycle on which they focus. Since different tools covering different stages share common information, it is required that they integrate through some central repository system (data dictionary) to have a consistent view of such information. In phases of software development life cycle integrated through a central data dictionary. Case Tools are used in many ways in our organizations. Case tools can be broadly classed into these broader areas:

- Requirement Analysis Tool
- Structure Analysis Tool
- Software Design Tool
- Code Generation Tool
- Test Case Generation Tool
- Document Production Tool
- Reverse Engineering Tool

While many organizations still use the SDLC methodology, it is often supplemented with other methods. Many systems developers use the CASE tools in various stages of the Software Development Life Cycle. They mainly use it while developing the following methodologies:

- Life Cycle
- Object-oriented Approach
- Rapid Applications Development (RAD)
- Prototyping
- Joint Applications Development (JAD)

The job of a systems developer may contain requirements analysis, process design, data design, and programming among other activities. But, not all systems developers do the same activities. One may spend most of his or her time on analysis; another, on design. The various activities that the system developers involve include Systems

Analysis (including feasibility studies and requirements definition), Systems Design (including user interface, data, and process design), Programming (or generating code), Testing, Supervisory or other management tasks and Maintenance. CASE tools play an important role in helping the system developers to perform the task efficiently.

CASE Tools in future...

Horizontal and Vertical Division of AO-groups

The whole system is composed of many active objects. Active objects (AO) are categorized into several groups. Active objects in the same group are responsible for the same type of tasks. For example, active objects to support SA/SD methodology and object-oriented methodology. This is what we call horizontal division. All AO-groups are organized hierarchically. For example, the group on top level is for users; the group on second level is for domains; the group on third level is for development knowledge; the group on fourth level is for tools; the group on fifth level is for technique support. This is what we call the vertical division. For each group, there is a delegate which is a local server of that group. The communication among different groups is through delegates at the corresponding levels.

Knowledge Representation and Processing

The system maintains multiple layers of knowledge and has the ability of self-learning and self-improvement. Especially, the system can do reasoning on incomplete information. Otherwise, the system cannot 'figure out' users' intentions and the interaction between a user and a tool will not be able to continue in some cases. Each node can be bound to a set of rules. Mixed knowledge representation is good for reducing the size of the network, and thus speeding up the learning process. In neural network, an output can always be derived from any input, even if the input is incomplete. Self learning is a natural and standard process in a neural network.

Visual Integration of CASE Tools

Although a CASE shell is very useful for the creation and integration of CASE tools, we believe it still is not simple and intuitive enough for CASE users. We suggest to provide CASE users with more intuitive means to describe the integration of CASE tools. A diagramming tool, similar to DFD diagramming tool, might be useful for this purpose. Pipe-filter, event-trigger and message passing models are all useful means to realize the integration of CASE tools.

Work Flow Model

Basically, software project development is a team based activity. A CASE tool should be able to support this feature. For this purpose, it is necessary to model the work process, and the collaboration and coordination among team members. We hope to represent all these information through a work flow model. Currently, there are many ways to model a work flow for team based collaborative software development. However, most of them are too strict to change dynamically (on the fly). We suggest using decentralized process models, such as 'ViewPoint' (**E**), which can be described visually and is also possible to cope with deviations during process enactment.

Java Technologies

We are sure that the next generation CASE tools will be able to operate in heterogeneous and distributed environments. JAVA is such a specification which allows for transparent access to applications operating on multiple computing platforms. JAVA is endorsed by the OMG, an organization that includes major computer manufacturers such as Digital, Sun, HP, IBM as well as software providers such as Microsoft, Sunsoft, and Object Design among its members. JAVA is possibly to become a de facto standard in the future. Based on this observation, we suggest that the next generation CASE tools are established on CORBA standard. To develop platform independent CASE tools, some platform independent programming languages, such as Java programming language will be used.

Making a case for and against and CASE Tools

For	Against
Helps standardization of notations and diagrams	Limitations in the flexibility of documentation
Help communication between development team members	May lead to restriction to the tool's capabilities
Automatically check the quality of the models	Major danger: completeness and syntactic correctness does NOT mean compliance with requirements
Reduction of time and effort	Costs associated with the use of the tool: purchase + training
Enhance reuse of models or models' components	Staff resistance to CASE tools

Web References

1. http://en.wikipedia.org/wiki/Computer_aided_software_engineering
2. <http://www.clariondeveloper.net/modules/weblinks/viewcat.php?cid=32>
3. <http://www.openeden.com/opensource/viewcat/id/5752850>
4. www.infoweblinks.com/content/casetools.htm
5. www.downloadthat.com/pda/catalog/Case_Tools
6. www.itmweb.com/case.htm
7. www.visual-paradigm.com/product/vpuml/
8. www.compinfo.co.uk/apps/case_tools.htm
9. www.objectsbydesign.com/tools/umltools_byCompany.html
10. www.sparxsystems.com.au/products/ea/index.html

Paper References

1. Norman, R.J. and Forte, G. "Automating the Software Development Process: CASE in the '90s," *Communications of the ACM* (35:4), 1992, p. 27
2. Brown, et al., *Principles of CASE Tool Integrations*, Oxford University Press, New York
3. Elshazly, H. and Grover, V. "A Study on the Evaluation of CASE Technology," *Journal of Information Technology Management* (4:1), 1993.
4. Forte, G. and Norman, R.J. "A Self-Assessment by the Software Engineering Community," *Communications of the ACM* (35:4), 1992, 28-32.
5. P. Jallaart et al., "A Reflective Approach to Process Model Customization, Enactment and Evolution", *Proc of the 3rd International Conference on the Software Process*, 1994, pp.21-32.
6. F. Russell, "The case for Case", *Software Engineering : A European Perspective*, edited by Richard H. Thayer and Andrew D. McGettrick, IEEE Computer Society Press, Los Alamitos, California, pp.531-547.
7. I. Aaen, "CASE Tool Bootstrapping--how little strokes fell great oaks", *Next Generation CASE Tools*, edited by K. Lyytinen, V.-P. Tahvanainen, IOS, Netherlands, 1992, pp.8-17.
8. A. Brown, et al., *Principles of CASE Tool Integrations*, Oxford University Press, New York, 1994.
9. P. Jallaart et al., "A Reflective Approach to Process Model Customization, Enactment and Evolution", *Proc of the 3rd International Conference on the Software Process*, 1994, pp.21-32.

10. U. Leonhardt et al., " Decentralized Process Enactment in a Multi-Perspective Development Environment ", Proc of the 17th International Conference on Software Engineering, 1995, pp.255-264.

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.

In this software engineering tutorial, you will learn-

- Why do we need Configuration management?
- Tasks in SCM process
- Configuration Identification:
- Baseline
- Change Control
- Configuration Status Accounting
- Configuration Audits and Reviews
- Participant of SCM process
- Software Configuration Management Plan
- Software Configuration Management Tools

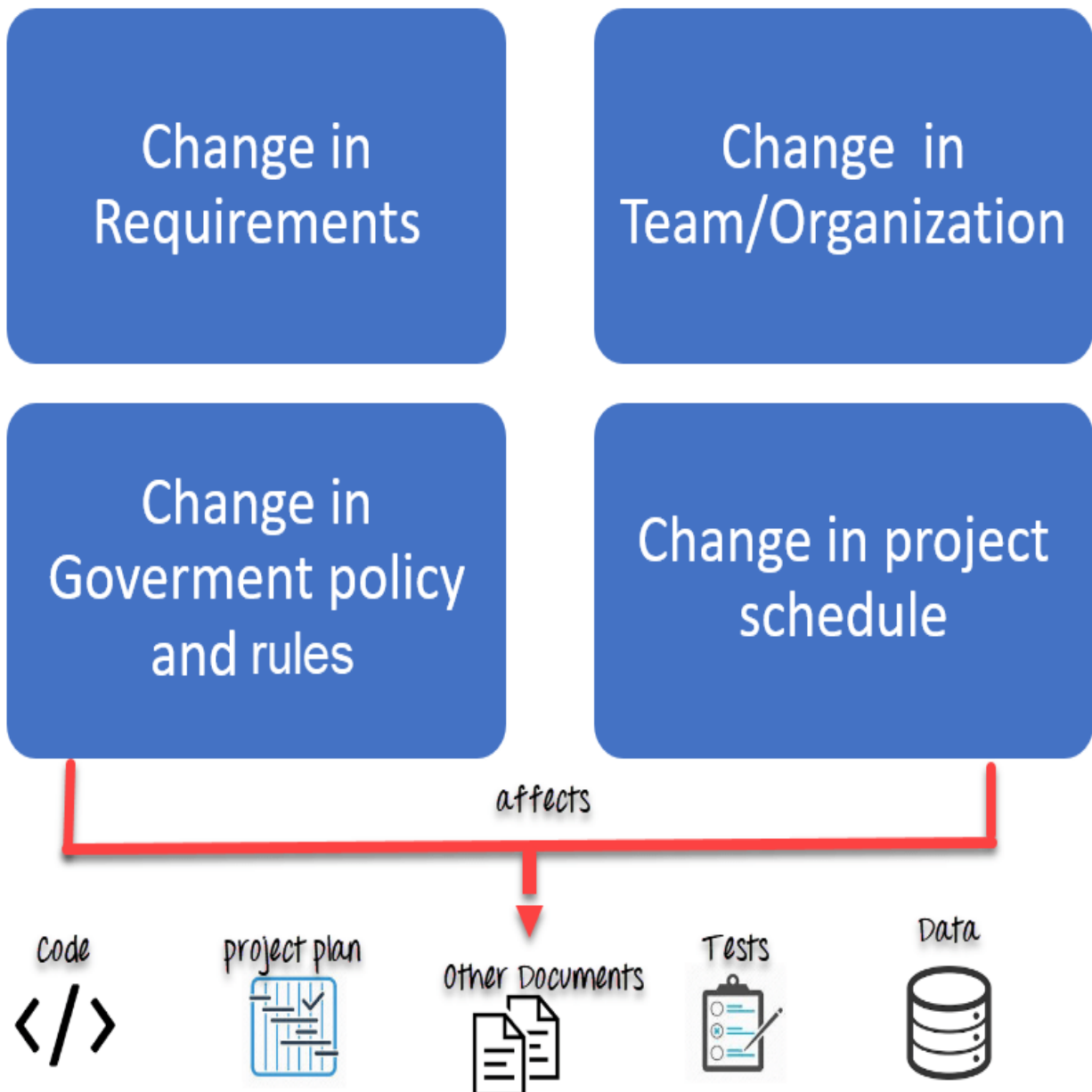
Why do we need Configuration management?

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 be 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.php its 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:

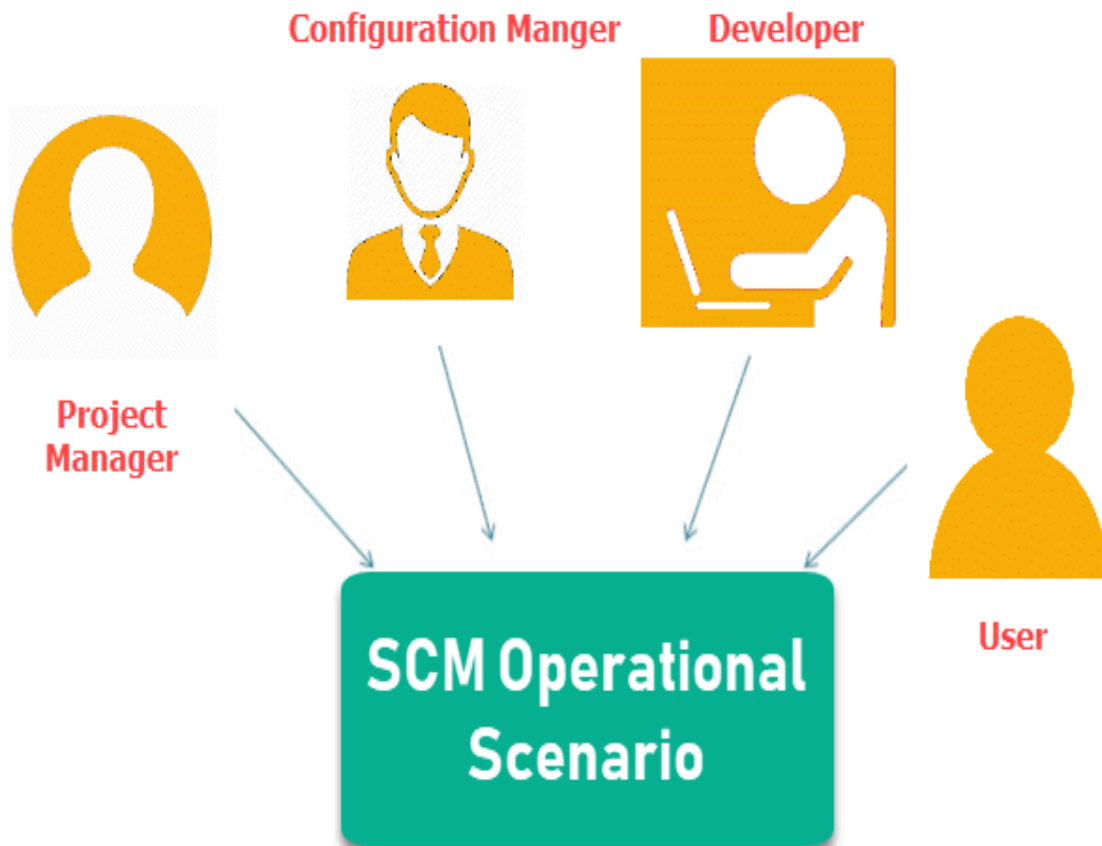
Software Configuration audits verify that all the software product satisfies the baseline needs. It ensures that what is built is what is delivered.

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.

Following are popular tools

1. Git: Git is a free and open source tool which helps version control. It is designed to handle all types of projects with speed and efficiency.

Download link: <https://git-scm.com/>

2. Team Foundation Server: Team Foundation is a group of tools and technologies that enable the team to collaborate and coordinate for building a product.

Download link: <https://www.visualstudio.com/tfs/>

3. Ansible: It is an open source Software configuration management tool. Apart from configuration management it also offers application deployment & task automation.

Conclusion:

- Configuration Management best practices helps organizations to systematically manage, organize, and control the changes in the documents, codes, and other entities during the Software Development Life Cycle.
- The primary goal of the SCM process is to increase productivity with minimal mistakes

- The main reason behind configuration management process is that there are multiple people working on software which is continually updating. SCM helps establish concurrency, synchronization, and version control.
- A baseline is a formally accepted version of a software configuration item
- Change control is a procedural method which ensures quality and consistency when changes are made in the configuration object.
- Configuration status accounting tracks each release during the SCM process
- Software Configuration audits verify that all the software product satisfies the baseline needs
- Project manager, Configuration manager, Developer, Auditor, and user are participants in SCM process
- The SCM process planning begins at the early phases of a project.
- Git, Team foundation Sever and Ansible are few popular SCM tools.

When we develop software, the product (software) undergoes many changes in their maintenance phase; we need to handle these changes effectively.

Several individuals (programs) works together to achieve these common goals. This individual produces several work product (SC Items) e.g., Intermediate version of modules or test data used during debugging, parts of the final product.

The elements that comprise all information produced as a part of the software process are collectively called a software configuration.

As software development progresses, the number of Software Configuration elements (SCI's) grow rapidly.

These are handled and controlled by SCM. This is where we require software configuration management.

A configuration of the product refers not only to the product's constituent but also to a particular version of the component.

Therefore, SCM is the discipline which

- Identify change

- Monitor and control change
- Ensure the proper implementation of change made to the item.
- Auditing and reporting on the change made.

Configuration Management (CM) is a technic of identifying, organizing, and controlling modification to software being built by a programming team.

The objective is to maximize productivity by minimizing mistakes (errors).

CM is used to essential due to the inventory management, library management, and updation management of the items essential for the project.

Why do we need Configuration Management?

Multiple people are working on software which is consistently updating. It may be a method where multiple version, branches, authors are involved in a software project, and the team is geographically distributed and works concurrently. It changes in user requirements, and policy, budget, schedules need to be accommodated.

Importance of SCM

It is practical in controlling and managing the access to various SCIs e.g., by preventing the two members of a team for checking out the same component for modification at the same time.

It provides the tool to ensure that changes are being properly implemented.

It has the capability of describing and storing the various constituent of software.

SCM is used in keeping a system in a consistent state by automatically producing derived version upon modification of the same component.