SOFTWARE ENGINEERING AND TOOLS

MODULE 1

📄 Overview of system analysis and design

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

There are four main steps involved in systems design and analysis.

1. Identifying Business Needs

It is important for a company to weigh the pros and cons of changing their currents systems already in place. Identifying the value of the new system is incredibly important for this process to be successful. Just like building your dream home, you want to make sure it's an improvement from your current home.

Businesses must first identify what their needs are, and they can do this by asking the following questions:

• What value do we want to bring to the organization?

• Is it to improve efficiency in the workplace?

• How do we want to change our current processes?

• What results are we trying to achieve?

• What are the budgetary concerns?

This stage involves all levels of the organization, to collaborate the business needs and the overall ability to develop the new information system.

2. Planning and Analyzing

Once the business needs are set, planning and analysis begins. This process should begin by identifying the extent to which they are trying to use this new system. This process is similar to a blueprint one might have for a home, where you lay out the big picture. Part of determining this is identifying who the users of the new system will be and what the system will be used for as well as addressing any privacy concerns. This initial part is important to make sure that the business objectives are being met, and it lays the foundation for the 'big picture'.

By looking into the needs of the users of the system, this helps identify the reach or span in which the new system will be used. It determines the cross functionality of the system and if it will be used for one or many departments.

Some questions one might ask include the following:

• What data should we store, and how does that, in turn, provide information to the company?

• Do we want the marketing department information to be integrated with the sales departments?

Asking questions like this will help to identify the 'who' and 'what' the system will be used for. Part of this stage should also address any privacy concerns the company may have as far as who should or should not have access to the information. Certain information is more sensitive than others and only certain users should be given specific access.

3. Designing a System

This phase will start by researching what necessary hardware and/or software is needed to use the system. At this point, the company will look at their physical structure and find ways to mimic this and any new changes in the new information system.

Questions asked during this phase include the following:

• How do we want to set up and store the data we need?

• How should we capture the data?

• How can we ensure that the information is accurate or complete?

Once this is answered, logical design is also incredibly important to make the system logical to daily users. This system should be used to improve organization systems, and consideration for ease of use and user trainability are essential in this phase.

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 −

o Inputs that enter the system

o The processor through which transformation takes place

o The program(s) required for processing

o 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.

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).

📄Business system concept

The system helps the business organisations to achieve their goals.

A business system is a combination of policies, personnel, equipment and computer facilities to co-ordinate the activities of a business organisation.

It establishes the rules and procedures of that organisation, which are to be governed.

Business system decides how data must be handled and is methodically processed. It also controls the procedures of the processed data and the results to be displayed. For e.g. a system may automatically order parts for an inventory, monitor future corporate profits or post credit card sales to the on line customer accounts. The overall nature of the business system will reflect the efficiency of its designers.

Objectives of Business System:

The objectives of business system are:

1. To meet the user and customer needs.

2. To cut down the operating costs and increase savings.

3. To smooth the flow data through various levels of the organisation.

4. To speed up the execution of results with the reliable data available in a system.

5. To handle data efficiently and provide timely information to the management.

6. To establish the most desirable distribution of data, services and equipment’s throughout the organisation.

7. To define a proper method of handling business activities.

8. To eliminate duplicated, conflicting and unnecessary services.

Types of Business Systems:

There are five major types of business systems (Fig. 13.6).

1. Payroll business system

2. Personnel business system

3. Accounts receivable system

4. Accounts payable system

5. Inventory system.

1. Payroll Business System:

A payroll system consists of all forms, procedures, files, equipment’s, personnel, and computer support necessary to completely process the payment of employees. A payroll system fully handles all tax deductions, personal deductions, and the update of payroll data related to each employee.

It provides for the actual payment of employees, a record of that payment, the modification of all payroll records, and the preparation of payroll reports. The payroll system must also generate all tax documents to include pay-cheques, W-2 statements, 941 quarterly reports, and a wide range of state and municipal employment tax filings.

Another payroll responsibility is the accurate reporting of all personal deductions to include bonds, medical and life insurance, profit sharing plans, stock options, credit union deductions, and the garnishing of an employee’s salary by a creditor.

These accumulated totals must be reported accurately to both the recipient of these movies and the individuals from whose salaries these amounts were deducted. The computer’s support makes it possible to accurately and promptly process a payroll, providing the input data are properly handled on a timely basis.

2. Personnel Business System:

Personnel system describes varied aspects of an organisation’s work force. The outputs generated by personnel systems are frequently used in compiling central & state labor power reports. Retail organisations are major users of accounts receivable systems, since these systems detail monies that are owed to an organisation.

Conversely, accounts payable systems focus on the monies that are owed to an organisation. These two systems parallel to each other, requiring the continued maintenance of files, their update reporting on movies due and owed, providing customer statements and invoices, and recording payments made.

3. Accounts Receivable System:

An account receivable systems are monitors the flow of money. An accounts receivable system monitors the people who owe money to a business. It provides the means to process all data for credit cards and other kinds of charge accounts.

The files contain the individual customer data, including names, addresses, financial charges like, payments received and current charges. The information is issued as monthly statements of each customer and also provides useful information for management’s use.

4. Accounts Payable System:

Accounts payable system monitors the organisation to which money is owed. The file structures and input/output (I/O) formats are similar as the accounts receivable system. It contains the accounts of vendors to whom money is owed. Input will have goods and services received by the company while outputs include issue of payments and management reports.

5. Inventory System:

Inventory system monitors the status of items held in an inventory. These systems report on the quantities of goods on hand, as well as when items should be purchased to replenish stock and what critical items are needed. Inventory systems are crucial to organisations that maintain large and costly inventories.

📄 System Development Life Cycle (SDLC)

The systems development life cycle (SDLC) is a conceptual model used in project management that describes the stages involved in an information system development project, from an initial feasibility study through maintenance of the completed application.

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 –

SDLC Phases

The entire SDLC process divided into the following stages:

 Phase 1: Requirement collection and analysis

 Phase 2: Feasibility study:

 Phase 3: Design:

 Phase 4: Coding:

 Phase 5: Testing:

 Phase 6: Installation/Deployment:

 Phase 7: Maintenance:

In this tutorial, I have explained all these phases

Phase 1: Requirement collection and analysis:

The requirement is the first stage in the SDLC process. It is conducted by the senior team members with inputs from all the stakeholders and domain experts in the industry. Planning for the quality assurance requirements and recognization of the risks involved is also done at this stage.

This stage gives a clearer picture of the scope of the entire project and the anticipated issues, opportunities, and directives which triggered the project.

Requirements Gathering stage need teams to get detailed and precise requirements. This helps companies to finalize the necessary timeline to finish the work of that system.

Phase 2: Feasibility study:

Once the requirement analysis phase is completed the next step is to define and document software needs. This process conducted with the help of 'Software Requirement Specification' document also known as 'SRS' document. It includes everything which should be designed and developed during the project life cycle.

There are mainly five types of feasibilities checks:

• Economic: Can we complete the project within the budget or not?

• Legal: Can we handle this project as cyber law and other regulatory framework/compliances.

• Operation feasibility: Can we create operations which is expected by the client?

• Technical: Need to check whether the current computer system can support the software

• Schedule: Decide that the project can be completed within the given schedule or not.

Phase 3: Design:

In this third phase, the system and software design documents are prepared as per the requirement specification document. This helps define overall system architecture.

This design phase serves as input for the next phase of the model.

There are two kinds of design documents developed in this phase:

High-Level Design (HLD)

• Brief description and name of each module

• An outline about the functionality of every module

• Interface relationship and dependencies between modules

• Database tables identified along with their key elements

• Complete architecture diagrams along with technology details

Low-Level Design(LLD)

• Functional logic of the modules

• Database tables, which include type and size

• Complete detail of the interface

• Addresses all types of dependency issues

• Listing of error messages

• Complete input and outputs for every module

Phase 4: Coding:

Once the system design phase is over, the next phase is coding. In this phase, developers start build the entire system by writing code using the chosen programming language. In the coding phase, tasks are divided into units or modules and assigned to the various developers. It is the longest phase of the Software Development Life Cycle process.

In this phase, Developer needs to follow certain predefined coding guidelines. They also need to use programming tools like compiler, interpreters, debugger to generate and implement the code.

Phase 5: Testing:

Once the software is complete, and it is deployed in the testing environment. The testing team starts testing the functionality of the entire system. This is done to verify that the entire application works according to the customer requirement.

During this phase, QA and testing team may find some bugs/defects which they communicate to developers. The development team fixes the bug and send back to QA for a re-test. This process continues until the software is bug-free, stable, and working according to the business needs of that system.

Phase 6: Installation/Deployment:

Once the software testing phase is over and no bugs or errors left in the system then the final deployment process starts. Based on the feedback given by the project manager, the final software is released and checked for deployment issues if any.

Phase 7: Maintenance:

Once the system is deployed, and customers start using the developed system, following 3 activities occur

• Bug fixing - bugs are reported because of some scenarios which are not tested at all

• Upgrade - Upgrading the application to the newer versions of the Software

• Enhancement - Adding some new features into the existing software

The main focus of this SDLC phase is to ensure that needs continue to be met and that the system continues to perform as per the specification mentioned in the first phase.

Popular SDLC models

Here, are some most important phases of SDLC life cycle:

Waterfall model

The waterfall is a widely accepted SDLC model. In this approach, the whole process of the software development is divided into various phases. In this SDLC model, the outcome of one phase acts as the input for the next phase.

This SDLC model is documentation-intensive, with earlier phases documenting what need be performed in the subsequent phases.

Incremental Approach

The incremental model is not a separate model. It is essentially a series of waterfall cycles. The requirements are divided into groups at the start of the project. For each group, the SDLC model is followed to develop software. The SDLC process is repeated, with each release adding more functionality until all requirements are met. In this method, every cycle act as the maintenance phase for the previous software release. Modification to the incremental model allows development cycles to overlap. After that subsequent cycle may begin before the previous cycle is complete.

V-Model

In this type of SDLC model testing and the development, the phase is planned in parallel. So, there are verification phases on the side and the validation phase on the other side. V-Model joins by Coding phase.

Agile Model

Agile methodology is a practice which promotes continue interaction of development and testing during the SDLC process of any project. In the Agile method, the entire project is divided into small incremental builds. All of these builds are provided in iterations, and each iteration lasts from one to three weeks.

Spiral Model

The spiral model is a risk-driven process model. This SDLC model helps the team to adopt elements of one or more process models like a waterfall, incremental, waterfall, etc.

This model adopts the best features of the prototyping model and the waterfall model. The spiral methodology is a combination of rapid prototyping and concurrency in design and development activities.

Big bang model

Big bang model is focusing on all types of resources in software development and coding, with no or very little planning. The requirements are understood and implemented when they come.

This model works best for small projects with smaller size development team which are working together. It is also useful for academic software development projects. It is an ideal model where requirements is either unknown or final release date is not given.

Conclusion

• The SDLC is a systematic process for building software that ensures the quality and correctness of the software built

• The full form SDLC is Software Development Lifecycle.

• SDLC process provides a framework for a standard set of activities and deliverables

• Seven different SDLC stages are 1) Requirement collection and analysis 2) Feasibility study: 3) Design 4) Coding 5) Testing: 6) Installation/Deployment and 7) Maintenance

• The senior team members conduct the requirement analysis phase

• Feasibility Study stage includes everything which should be designed and developed during the project life cycle

• In the Design phase, the system and software design documents are prepared as per the requirement specification document

• In the coding phase, developers start build the entire system by writing code using the chosen programming language

• Testing is the next phase which is conducted to verify that the entire application works according to the customer requirement.

• Installation and deployment face begins when the software testing phase is over, and no bugs or errors left in the system

• Bug fixing, upgrade, and engagement actions covered in the maintenance face

• Waterfall, Incremental, Agile, V model, Spiral, Big Bang are some of the popular SDLC models

• SDLC consists of a detailed plan which explains how to plan, build, and maintain specific software

Advantages and disadvantages of SDLC

Benefits of abiding by a clearly defined SDLC model include:

 Having a clear view of an entire project, workers involved, estimated costs and timelines.

 Gives project managers a projected base cost of the project.

 Goals and standards are clearly defined.

 Developers can move back a step if something does not go as expected.

Disadvantages, however, can include:

• Due to assumptions made at the beginning of a project, if an unexpected circumstance complicates the development of a system, then it may stockpile into more complications down the road. As an example, if newly installed hardware does not work correctly, then it may increase the time a system is in development, increasing the cost.

• Some methods are not flexible.

• It can be complicated to estimate the overall cost at the beginning of a project.

• Testing at the end of development may slow down some development teams.

📄 Waterfall Model

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

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

The waterfall Model illustrates the software development process in a linear sequential flow. This means that any phase in the development process begins only if the previous phase is complete. In this waterfall model, the phases do not overlap. It was introduced in 1970 by Winston Royce.

Waterfall Model - Design

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

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

Let us now learn about each of these phases in brief details:

1. Requirements analysis and specification: The aim of the requirement analysis and specification phase is to understand the exact requirements of the customer and document them properly. This phase consists of two different activities.

o Requirement gathering and analysis: Firstly all the requirements regarding the software are gathered from the customer and then the gathered requirements are analyzed. The goal of the analysis part is to remove incompleteness (an incomplete requirement is one in which some parts of the actual requirements have been omitted) and inconsistencies (inconsistent requirement is one in which some part of the requirement contradicts with some other part).

o Requirement specification: These analyzed requirements are documented in a software requirement specification (SRS) document. SRS document serves as a contract between development team and customers. Any future dispute between the customers and the developers can be settled by examining the SRS document.

2. Design: The aim of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language.

3. Implementation − With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing. In coding phase software design is translated into source code using any suitable programming language. Thus each designed module is coded. The aim of the unit testing phase is to check whether each module is working properly or not.

4. Integration and System testing: Integration of different modules are undertaken soon after they have been coded and unit tested. Integration of various modules is carried out incrementally over a number of steps. During each integration step, previously planned modules are added to the partially integrated system and the resultant system is tested. Finally, after all the modules have been successfully integrated and tested, the full working system is obtained and system testing is carried out on this.

System testing consists three different kinds of testing activities as described below :

o Alpha testing: Alpha testing is the system testing performed by the development team.

o Beta testing: Beta testing is the system testing performed by a friendly set of customers.

o Acceptance testing: After the software has been delivered, the customer performed the acceptance testing to determine whether to accept the delivered software or to reject it.

5. Deployment of system − Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.

6. Maintainence: Maintenance is the most important phase of a software life cycle. The effort spent on maintenance is the 60% of the total effort spent to develop a full software. There are basically three types of maintenance :

o Corrective Maintenance: This type of maintenance is carried out to correct errors that were not discovered during the product development phase.

o Perfective Maintenance: This type of maintenance is carried out to enhance the functionalities of the system based on the customer’s request.

o Adaptive Maintenance: Adaptive maintenance is usually required for porting the software to work in a new environment such as work on a new computer platform or with a new operating system.

Advantages of Classical Waterfall Model

Classical waterfall model is an idealistic model for software development. It is very simple, so it can be considered as the basis for other software development life cycle models. Below are some of the major advantages of this SDLC model:

• This model is very simple and is easy to understand.

• Phases in this model are processed one at a time.

• Each stage in the model is clearly defined.

• This model has very clear and well undestood milestones.

• Process, actions and results are very well documented.

• Reinforces good habits: define-before- design,

design-before-code.

• This model works well for smaller projects and projects where requirements are well

understood.

Drawbacks of Classical Waterfall Model

Classical waterfall model suffers from various shortcomings, basically we can’t use it in real projects, but we use other software development lifecycle models which are based on the classical waterfall model. Below are some major drawbacks of this model:

• No feedback path: In classical waterfall model evolution of a software from one phase to another phase is like a waterfall. It assumes that no error is ever committed by developers during any phases. Therefore, it does not incorporate any mechanism for error correction.

• Difficult to accommodate change requests: This model assumes that all the customer requirements can be completely and correctly defined at the beginning of the project, but actually customers’ requirements keep on changing with time. It is difficult to accommodate any change requests after the requirements specification phase is complete.

• No overlapping of phases: This model recommends that new phase can start only after the completion of the previous phase. But in real projects, this can’t be maintained. To increase the efficiency and reduce the cost, phases may overlap.

No working software is produced until late during the life cycle.

High amounts of risk and uncertainty.

Not a good model for complex and object-oriented projects.

Poor model for long and ongoing projects.

Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.

It is difficult to measure progress within stages.

Cannot accommodate changing requirements.

Adjusting scope during the life cycle can end a project.

Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

Waterfall Model - Application

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

• Requirements are very well documented, clear and fixed.

• Product definition is stable.

• Technology is understood and is not dynamic.

• There are no ambiguous requirements.

• Ample resources with required expertise are available to support the product.

• The project is short.

📄 Spiral Model

Spiral Model is a combination of a waterfall model and iterative model. Each phase in spiral model begins with a design goal and ends with the client reviewing the progress. The spiral model was first mentioned by Barry Boehm in his 1986 paper.

The development team in Spiral-SDLC model starts with a small set of requirement and goes through each development phase for those set of requirements. The software engineering team adds functionality for the additional requirement in every-increasing spirals until the application is ready for the production phase.

To explain in simpler terms, the steps involved in the spiral model are:

A spiral model has 4 phases described below:

1. Planning phase

2. Risk analysis phase

3. Engineering phase

4. Evaluation phase.

Activities which are performed in the spiral model phases are shown below:

Phase Name Activities performed Deliverables / Output

Planning -Requirements are studied and gathered.

- Feasibility study

- Reviews and walkthroughs to streamline the requirements Requirements understanding document

Finalized list of requirements.

Risk Analysis Requirements are studied and brain storming sessions are done to identify the potential risks

Once the risks are identified , risk mitigation strategy is planned and finalized Document which highlights all the risks and its mitigation plans.

Engineering Actual development and testing if the software takes place in this phase Code

Test cases and test results

Test summary report and defect report.

Evaluation Customers evaluate the software and provide their feedback and approval Features implemented document

Pictorial representation of SDLC Spiral model

Different colors represent different spiral or iteration. For first iteration, represented in brown color , all the 4 activities (Planning, risk analysis, engineering and evaluation) are performed. After the evaluation phase is over for the first iteration (spiral), second iteration (spiral) starts The second iteration, which is represented in orange color, here again all the 4 activities (Planning, risk analysis, engineering and evaluation) are performed. In a similar way, third iteration is done shown in blue color and so on the process continues.

When to Use Spiral model?

Spiral model is used in the following scenarios:

• When the project is large.

• Where the software needs continuous risk evaluation.

• Requirements are a bit complicated and require continuous clarification.

• Software requires significant changes.

• Where enough time frame is their to get end user feedback.

• Where releases are required to be frequent.

Advantages of using Spiral Model:

Advantages of using Spiral model are as follows:

• Development is fast

• Larger projects / software are created and handled in a strategic way

• Risk evaluation is proper.

• Control towards all the phases of development.

• More and more features are added in a systematic way.

• Software is produced early.

• Has room for customer feedback and the changes are implemented faster.

Disadvantages of using Spiral model:

Disadvantages of Spiral model are as follows:

• Risk analysis is important phase so requires expert people.

• Is not beneficial for smaller projects.

• Spiral may go infinitely.

• Documentation is more as it has intermediate phases.

• It is costly for smaller projects.

Conclusion:

Each spiral can be termed as a loop and each loop is a separate development process in a spiral model. The four activities (Planning, Risk analysis, engineering and evaluation) form the intermediary phases of a spiral model and is repeated again for each loop.

This model is very good to use for larger projects where you can develop and deliver smaller prototypes and can enhance it to make the larger software. The implementation of this model requires experienced resources as risk analysis is a very integral part of this model and risk analysis requires expertise and as a result this model becomes costly.

📄 COCOMO Model

COCOMO is one of the most widely used software estimation models in the world. This model is developed in 1981 by Barry Boehm to give estimation of number of man-months it will take to develop a software product.

COCOMO predicts the efforts and schedule of software product based on size of software.

In COCOMO, projects are categorized into three types:

1. Organic

2. Semidetached

3. Embedded

1.Organic: A development project can be treated of the organic type, if the project deals with developing a well-understood application program, the size of the development team is reasonably small, and the team members are experienced in developing similar methods of projects. Examples of this type of projects are simple business systems, simple inventory management systems, and data processing systems.

2. Semidetached: A development project can be treated with semidetached type if the development consists of a mixture of experienced and inexperienced staff. Team members may have finite experience in related systems but may be unfamiliar with some aspects of the order being developed. Example of Semidetached system includes developing a new operating system (OS), a Database Management System (DBMS), and complex inventory management system.

3. Embedded: A development project is treated to be of an embedded type, if the software being developed is strongly coupled to complex hardware, or if the stringent regulations on the operational method exist. For Example: ATM, Air Traffic control.

For three product categories, Bohem provides a different set of expression to predict effort (in a unit of person month)and development time from the size of estimation in KLOC(Kilo Line of code) efforts estimation takes into account the productivity loss due to holidays, weekly off, coffee breaks, etc.

Types of Models:

COCOMO consists of a hierarchy of three increasingly detailed and accurate forms. Any of the three forms can be adopted according to our requirements. These are types of COCOMO model:

1. Basic COCOMO Model

2. Intermediate COCOMO Model

3. Detailed COCOMO Model

The first level, Basic COCOMO can be used for quick and slightly rough calculations of Software Costs. Its accuracy is somewhat restricted due to the absence of sufficient factor considerations.

Intermediate COCOMO takes these Cost Drivers into account and Detailed COCOMO additionally accounts for the influence of individual project phases, i.e in case of Detailed it accounts for both these cost drivers and also calculations are performed phase wise henceforth producing a more accurate result. These two models are further discussed below.

Estimation of Effort: Calculations –

Basic Model –

The above formula is used for the cost estimation of for the basic COCOMO model, and also is used in the subsequent models. The constant values a and b for the Basic Model for the different categories of system:

Software Projects a b

Organic 2.4 1.05

Semi Detached 3.0 1.12

Embedded 3.6 1.20

The effort is measured in Person-Months and as evident from the formula is dependent on Kilo-Lines of code. These formulas are used as such in the Basic Model calculations, as not much consideration of different factors such as reliability, expertise is taken into account, henceforth the estimate is rough.

Intermediate Model –

The basic Cocomo model assumes that the effort is only a function of the number of lines of code and some constants evaluated according to the different software system. However, in reality, no system’s effort and schedule can be solely calculated on the basis of Lines of Code. For that, various other factors such as reliability, experience, Capability. These factors are known as Cost Drivers and the Intermediate Model utilizes 15 such drivers for cost estimation.

Classification of Cost Drivers and their attributes:

(i) Product attributes –

 Required software reliability extent

 Size of the application database

 The complexity of the product

(ii) Hardware attributes –

 Run-time performance constraints

 Memory constraints

 The volatility of the virtual machine environment

 Required turnabout time

(iii) Personnel attributes –

 Analyst capability

 Software engineering capability

 Applications experience

 Virtual machine experience

 Programming language experience

(iv) Project attributes –

 Use of software tools

 Application of software engineering methods

 Required development schedule

Cost Drivers Very Low Low Nominal High Very High

Product Attributes

Required Software Reliability 0.75 0.88 1.00 1.15 1.40

Size of Application Database 0.94 1.00 1.08 1.16

Complexity of The Product 0.70 0.85 1.00 1.15 1.30

Hardware Attributes

Runtime Performance Constraints 1.00 1.11 1.30

Memory Constraints 1.00 1.06 1.21

Volatility of the virtual machine environment 0.87 1.00 1.15 1.30

Required turnabout time 0.94 1.00 1.07 1.15

Personnel attributes

Analyst capability 1.46 1.19 1.00 0.86 0.71

Applications experience 1.29 1.13 1.00 0.91 0.82

Software engineer capability 1.42 1.17 1.00 0.86 0.70

Virtual machine experience 1.21 1.10 1.00 0.90

Programming language experience 1.14 1.07 1.00 0.95

Project Attributes

Application of software engineering methods 1.24 1.10 1.00 0.91 0.82

Use of software tools 1.24 1.10 1.00 0.91 0.83

Required development schedule 1.23 1.08 1.00 1.04 1.10

The project manager is to rate these 15 different parameters for a particular project on a scale of one to three. Then, depending on these ratings, appropriate cost driver values are taken from the above table. These 15 values are then multiplied to calculate the EAF (Effort Adjustment Factor). The Intermediate COCOMO formula now takes the form:

The values of a and b in case of the intermediate model are as follows:

Software Projects a b

Organic 3.2 1.05

Semi Detached 3.0 1.12

Embeddedc 2.8 1.20

Detailed Model –

Detailed COCOMO incorporates all characteristics of the intermediate version with an assessment of the cost driver’s impact on each step of the software engineering process. The detailed model uses different effort multipliers for each cost driver attribute. In detailed cocomo, the whole software is divided into different modules and then we apply COCOMO in different modules to estimate effort and then sum the effort.

The Six phases of detailed COCOMO are:

 Planning and requirements

 System design

 Detailed design

 Module code and test

 Integration and test

 Cost Constructive model

The effort is calculated as a function of program size and a set of cost drivers are given according to each phase of the software lifecycle.

MODULE-2

📄 Software Requirement Specification

A software requirements specification (SRS) is a detailed description of a software system to be developed with its functional and non-functional requirements. The SRS is developed based the agreement between customer and contractors. It may include the use cases of how user is going to interact with software system. The software requirement specification document consistent of all necessary requirements required for project development. To develop the software system we should have clear understanding of Software system. To achieve this we need to continuous communication with customers to gather all requirements.

A good SRS defines the how Software System will interact with all internal modules, hardware, communication with other programs and human user interactions with wide range of real life scenarios. Using the Software requirements specification (SRS) document on QA lead, managers creates test plan. It is very important that testers must be cleared with every detail specified in this document in order to avoid faults in test cases and its expected results.

It is highly recommended to review or test SRS documents before start writing test cases and making any plan for testing. Let’s see how to test SRS and the important point to keep in mind while testing it.

1. Correctness of SRS should be checked. Since the whole testing phase is dependent on SRS, it is very important to check its correctness. There are some standards with which we can compare and verify.

2. Ambiguity should be avoided. Sometimes in SRS, some words have more than one meaning and this might confused testers making it difficult to get the exact reference. It is advisable to check for such ambiguous words and make the meaning clear for better understanding.

3. Requirements should be complete. When tester writes test cases, what exactly is required from the application, is the first thing which needs to be clear. For e.g. if application needs to send the specific data of some specific size then it should be clearly mentioned in SRS that how much data and what is the size limit to send.

4. Consistent requirements.The SRS should be consistent within itself and consistent to its reference documents. If you call an input “Start and Stop” in one place, don’t call it “Start/Stop” in another. This sets the standard and should be followed throughout the testing phase.

5. Verification of expected result: SRS should not have statements like “Work as expected”, it should be clearly stated that what is expected since different testers would have different thinking aspects and may draw different results from this statement.

6. Testing environment: some applications need specific conditions to test and also a particular environment for accurate result. SRS should have clear documentation on what type of environment is needed to set up.

7. Pre-conditions defined clearly: one of the most important part of test cases is pre-conditions. If they are not met properly then actual result will always be different expected result. Verify that in SRS, all the pre-conditions are mentioned clearly.

8. Requirements ID: these are the base of test case template. Based on requirement Ids, test case ids are written. Also, requirements ids make it easy to categorize modules so just by looking at them, tester will know which module to refer. SRS must have them such as id defines a particular module.

9. Security and Performance criteria: security is priority when a software is tested especially when it is built in such a way that it contains some crucial information when leaked can cause harm to business. Tester should check that all the security related requirements are properly defined and are clear to him. Also, when we talk about performance of a software, it plays a very important role in business so all the requirements related to performance must be clear to the tester and he must also know when and how much stress or load testing should be done to test the performance.

10. Assumption should be avoided: sometimes when requirement is not cleared to tester, he tends to make some assumptions related to it, which is not a right way to do testing as assumptions could go wrong and hence, test results may vary. It is better to avoid assumptions and ask clients about all the “missing requirements” to have a better understanding of expected results.

11. Deletion of irrelevant requirements: there are more than one team who work on SRS so it might be possible that some irrelevant requirements are included in SRS. Based on the understanding of the software, tester can find out which are these requirements and remove them to avoid confusions and reduce work load.

12. Freeze requirements: when an ambiguous or incomplete requirement is sent to client to analyze and tester gets a reply, that requirement result will be updated in the next SRS version and client will freeze that requirement. Freezing here means that result will not change again until and unless some major addition or modification is introduced in the software.

Characteristics of good SRS

Following are the features of a good SRS document:

1. Correctness: User review is used to provide the accuracy of requirements stated in the SRS. SRS is said to be perfect if it covers all the needs that are truly expected from the system.

2. Completeness: The SRS is complete if, and only if, it includes the following elements:

(1). All essential requirements, whether relating to functionality, performance, design, constraints, attributes, or external interfaces.

(2). Definition of their responses of the software to all realizable classes of input data in all available categories of situations.

Note: It is essential to specify the responses to both valid and invalid values.

(3). Full labels and references to all figures, tables, and diagrams in the SRS and definitions of all terms and units of measure.

3. Consistency: The SRS is consistent if, and only if, no subset of individual requirements described in its conflict. There are three types of possible conflict in the SRS:

(1). The specified characteristics of real-world objects may conflicts. For example,

(a) The format of an output report may be described in one requirement as tabular but in another as textual.

(b) One condition may state that all lights shall be green while another states that all lights shall be blue.

(2). There may be a reasonable or temporal conflict between the two specified actions. For example,

(a) One requirement may determine that the program will add two inputs, and another may determine that the program will multiply them.

(b) One condition may state that "A" must always follow "B," while other requires that "A and B" co-occurs.

(3). Two or more requirements may define the same real-world object but use different terms for that object. For example, a program's request for user input may be called a "prompt" in one requirement's and a "cue" in another. The use of standard terminology and descriptions promotes consistency.

4. Unambiguousness: SRS is unambiguous when every fixed requirement has only one interpretation. This suggests that each element is uniquely interpreted. In case there is a method used with multiple definitions, the requirements report should determine the implications in the SRS so that it is clear and simple to understand.

5. Ranking for importance and stability: The SRS is ranked for importance and stability if each requirement in it has an identifier to indicate either the significance or stability of that particular requirement.

Typically, all requirements are not equally important. Some prerequisites may be essential, especially for life-critical applications, while others may be desirable. Each element should be identified to make these differences clear and explicit. Another way to rank requirements is to distinguish classes of items as essential, conditional, and optional.

6. Modifiability: SRS should be made as modifiable as likely and should be capable of quickly obtain changes to the system to some extent. Modifications should be perfectly indexed and cross-referenced.

7. Verifiability: SRS is correct when the specified requirements can be verified with a cost-effective system to check whether the final software meets those requirements. The requirements are verified with the help of reviews.

8. Traceability: The SRS is traceable if the origin of each of the requirements is clear and if it facilitates the referencing of each condition in future development or enhancement documentation.

There are two types of Traceability:

1. Backward Traceability: This depends upon each requirement explicitly referencing its source in earlier documents.

2. Forward Traceability: This depends upon each element in the SRS having a unique name or reference number.

The forward traceability of the SRS is especially crucial when the software product enters the operation and maintenance phase. As code and design document is modified, it is necessary to be able to ascertain the complete set of requirements that may be concerned by those modifications.

9. Design Independence: There should be an option to select from multiple design alternatives for the final system. More specifically, the SRS should not contain any implementation details.

10. Testability: An SRS should be written in such a method that it is simple to generate test cases and test plans from the report.

11. Understandable by the customer: An end user may be an expert in his/her explicit domain but might not be trained in computer science. Hence, the purpose of formal notations and symbols should be avoided too as much extent as possible. The language should be kept simple and clear.

12. The right level of abstraction: If the SRS is written for the requirements stage, the details should be explained explicitly. Whereas,for a feasibility study, fewer analysis can be used. Hence, the level of abstraction modifies according to the objective of the SRS.

Properties of a good SRS document

The essential properties of a good SRS document are the following:

Concise: The SRS report should be concise and at the same time, unambiguous, consistent, and complete. Verbose and irrelevant descriptions decrease readability and also increase error possibilities.

Structured: It should be well-structured. A well-structured document is simple to understand and modify. In practice, the SRS document undergoes several revisions to cope up with the user requirements. Often, user requirements evolve over a period of time. Therefore, to make the modifications to the SRS document easy, it is vital to make the report well-structured.

Black-box view: It should only define what the system should do and refrain from stating how to do these. This means that the SRS document should define the external behavior of the system and not discuss the implementation issues. The SRS report should view the system to be developed as a black box and should define the externally visible behavior of the system. For this reason, the SRS report is also known as the black-box specification of a system.

Conceptual integrity: It should show conceptual integrity so that the reader can merely understand it. Response to undesired events: It should characterize acceptable responses to unwanted events. These are called system response to exceptional conditions.

Verifiable: All requirements of the system, as documented in the SRS document, should be correct. This means that it should be possible to decide whether or not requirements have been met in an implementation.

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

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It can be manual, automated, or a combination of both.

It shows how data enters and leaves the system, what changes the information, and where data is stored.

The objective of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communication tool between a system analyst and any person who plays a part in the order that acts as a starting point for redesigning a system. The DFD is also called as a data flow graph or bubble chart.

There is a prominent difference between DFD and Flowchart. The flowchart depicts flow of control in program modules. DFDs depict flow of data in the system at various levels. DFD does not contain any control or branch elements.

Types of DFD

Data Flow Diagrams are either Logical or Physical.

 Logical DFD - This type of DFD concentrates on the system process, and flow of data in the system.For example in a Banking software system, how data is moved between different entities.

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

DFD Components

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

• Entities - Entities are source and destination of information data. Entities are represented by a rectangles with their respective names.

• Process - Activities and action taken on the data are represented by Circle or Round-edged rectangles.

• Data Storage - There are two variants of data storage - it can either be represented as a rectangle with absence of both smaller sides or as an open-sided rectangle with only one side missing.

• Data Flow - Movement of data is shown by pointed arrows. Data movement is shown from the base of arrow as its source towards head of the arrow as destination.

The following observations about DFDs are essential:

 All names should be unique. This makes it easier to refer to elements in the DFD.

 Remember that DFD is not a flow chart. Arrows is a flow chart that represents the order of events; arrows in DFD represents flowing data. A DFD does not involve any order of events.

 Suppress logical decisions. If we ever have the urge to draw a diamond-shaped box in a DFD, suppress that urge! A diamond-shaped box is used in flow charts to represents decision points with multiple exists paths of which the only one is taken. This implies an ordering of events, which makes no sense in a DFD.

 Do not become bogged down with details. Defer error conditions and error handling until the end of the analysis.

Standard symbols for DFDs are derived from the electric circuit diagram analysis and are shown in fig:

Circle: A circle (bubble) shows a process that transforms data inputs into data outputs.

Data Flow: A curved line shows the flow of data into or out of a process or data store.

Data Store: A set of parallel lines shows a place for the collection of data items. A data store indicates that the data is stored which can be used at a later stage or by the other processes in a different order. The data store can have an element or group of elements.

Source or Sink: Source or Sink is an external entity and acts as a source of system inputs or sink of system outputs.

Levels in Data Flow Diagrams (DFD)

The DFD may be used to perform a system or software at any level of abstraction. Infact, DFDs may be partitioned into levels that represent increasing information flow and functional detail. Levels in DFD are numbered 0, 1, 2 or beyond. Here, we will see primarily three levels in the data flow diagram, which are: 0-level DFD, 1-level DFD, and 2-level DFD.

0-level DFDM

It is also known as fundamental system model, or context diagram represents the entire software requirement as a single bubble with input and output data denoted by incoming and outgoing arrows. Then the system is decomposed and described as a DFD with multiple bubbles. Parts of the system represented by each of these bubbles are then decomposed and documented as more and more detailed DFDs. This process may be repeated at as many levels as necessary until the program at hand is well understood. It is essential to preserve the number of inputs and outputs between levels, this concept is called leveling by DeMacro. Thus, if bubble "A" has two inputs x1 and x2 and one output y, then the expanded DFD, that represents "A" should have exactly two external inputs and one external output as shown in fig:

The Level-0 DFD, also called context diagram of the result management system is shown in fig. As the bubbles are decomposed into less and less abstract bubbles, the corresponding data flow may also be needed to be decomposed.

1-level DFD

In 1-level DFD, a context diagram is decomposed into multiple bubbles/processes. In this level, we highlight the main objectives of the system and breakdown the high-level process of 0-level DFD into subprocesses.

2-Level DFD

2-level DFD goes one process deeper into parts of 1-level DFD. It can be used to project or record the specific/necessary detail about the system's functioning.

📄 Data Dictionary

A data dictionary is a file or a set of files that includes a database's metadata. The data dictionary hold records about other objects in the database, such as data ownership, data relationships to other objects, and other data. The data dictionary is an essential component of any relational database. Ironically, because of its importance, it is invisible to most database users. Typically, only database administrators interact with the data dictionary.

The data dictionary, in general, includes information about the following:

o Name of the data item

o Aliases

o Description/purpose

o Related data items

o Range of values

o Data structure definition/Forms

The name of the data item is self-explanatory.

Aliases include other names by which this data item is called DEO for Data Entry Operator and DR for Deputy Registrar.

Description/purpose is a textual description of what the data item is used for or why it exists.

Related data items capture relationships between data items e.g., total\_marks must always equal to internal\_marks plus external\_marks.

Range of values records all possible values, e.g. total marks must be positive and between 0 to 100.

Data structure Forms: Data flows capture the name of processes that generate or receive the data items. If the data item is primitive, then data structure form captures the physical structures of the data item. If the data is itself a data aggregate, then data structure form capture the composition of the data items in terms of other data items.

The mathematical operators used within the data dictionary are defined in the table:

Notations Meaning

x=a+b x includes of data elements a and b.

x=[a/b] x includes of either data elements a or b.

x=a x includes of optimal data elements a.

x=y[a] x includes of y or more occurrences of data element a

x=[a]z x includes of z or fewer occurrences of data element a

x=y[a]z x includes of some occurrences of data element a which are between y and z.

The different types of data dictionary are −

Active Data Dictionary

If the structure of the database or its specifications change at any point of time, it should be reflected in the data dictionary. This is the responsibility of the database management system in which the data dictionary resides.

So, the data dictionary is automatically updated by the database management system when any changes are made in the database. This is known as an active data dictionary as it is self updating.

Passive Data Dictionary

This is not as useful or easy to handle as an active data dictionary. A passive data dictionary is maintained separately to the database whose contents are stored in the dictionary. That means that if the database is modified the database dictionary is not automatically updated as in the case of Active Data Dictionary.

So, the passive data dictionary has to be manually updated to match the database. This needs careful handling or else the database and data dictionary are out of sync

📄 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.

Elements of a System

• Architecture - This is the conceptual model that defines the structure, behavior and more views of a system. We can use flowcharts to represent and illustrate the architecture.

• Modules - This are components that handle one specific tasks in a system. A combination of the modules make up the system.

• Components - This provides a particular function or group of related functions. They are made up of modules.

• Interfaces - This is the shared boundary across which the components of a the system exchange information and relate.

• Data - This the management of the information and data flow.

Major Tasks Performed During the System Design Process

1. Initialize design definition

• Plan for and Identify the technologies that will compose and implement the systems elements and their physical interfaces.

• Determine which technologies and system elements have a risk to become obsolete, or evolve during the operation stage of the system. Plan for their potential replacement.

• Document the design definition strategy, including the need for and requirements of any enabling systems, products, or services to perform the design.

2. Establish design characteristics

• Define the design characteristics relating to the architectural characteristics and check that they are implementable.

• Define the interfaces that were not defined by the System Architecture process or that need to be refined as the design details evolve.

• Define and document the design characteristics of each system element2.

3. Assess alternatives for obtaining system elements

• Assess the design options

• Select the most appropriate alternatives.

• If the decision is made to develop the system element, rest of the design definition process and the implementation process are used. If the decision is to buy or reuse a system element, the acquisition process may be used to obtain the system element.

4. Manage the design

• Capture and maintain the rationale for all selections among alternatives and decisions for the design, architecture characteristics.

• Assess and control the evolution of the design characteristics.

Factors that Affect Technology Trade-offs during System Design

Scale of Product

• For example, enterprise software companies that are building system-level software prioritize reliability because customers need to use them. Each change needs to be rigorously tested, and often approved before it can be released.

• Meanwhile, consumer internet companies spend time and money on making their UX delightful so that people want to use them. Reliability is something they’re willing to sacrifice. Since many are web-based applications, they can iterate quickly and release changes frequently.

Time

• Learning new technologies sometimes often takes time. The trade-offs in this instance will be made according to which stack/technology will be in time with the set delivery dates. If switching to a new stack/technology will result in a major shift on the delivery dates and major inconveniences to the stakeholders then the switch can be held off until an appropriate time.

Cost

• On a larger scale Technology decisions are made based on which is more cost effective, where a comparison can be done on which will be more effective between buying an off the shelf system and customizing it or building a new system.

Efficiency

• Technology trade offs are also done based on which technology is more efficient for example choosing between ReactJs or AngularJs for a front end application.

User Experience and Support

• The amount of support and documentation available on a given technology can also be a determining factor on the decisions. Working with Technologies that have a large support base, comprehensive documentation and A good user experience is much easier and take a very short time to ramp up on due to the large amount of resources available to support it.

Maintainability

• maintainability in this case is the ease with which a product can be maintained in order to correct errors, fix bugs and add additional features. Trade-offs decisions will be made based on the maintainability of the Technology

Reliability

• In this case the trade offs are made based on the Technology that performs consistently well and consistently upgrading to more efficient versions.

Scalability

• Technology trade offs are also made based on the technologies that are more scalable and able to handle increase loads efficiently without a break in the system efficiency.

MVC Design pattern

The Model View Controller (MVC) design pattern specifies that an application consist of a data model, presentation information, and control information.

MVC mostly relates to the user Interface/interaction layer of an application.

In the MVC pattern the user sees the View which is updated by the model which is turn manipulated by the Controller.

MVC Pattern

• The Model contains only the pure application data, it contains no logic describing how to present the data to a user. They are the parts of the application that implement the logic for the application’s data domain. They retrieve and store model state in a database.

• The View presents the model’s data to the user. The view can only be used to access the model’s data. They are the components that display the application’s user interface (UI).

• The Controller exists between the view and the model. It listens to events triggered by the view and executes the appropriate commands. They are the components that handle user interaction, work with the model, and ultimately select a view to render that displays UI.

Advantages of the MVC design pattern

• Multiple developers can work simultaneously on the model, controller and views.

• MVC enables logical grouping of related actions on a controller together. The views for a specific model are also grouped together.

• Low coupling — The very nature of the MVC framework is such that there is low coupling among models, views or controllers.

• Models can have multiple views.

• Ease of modification — Because of the separation of responsibilities, future development or modification is easier

Disadvantages

• Knowledge on multiple technologies becomes the norm. Developers using MVC need to be skilled in multiple technologies.

Below is an example of a System Design

📄 Problem Partitioning

 When solving a small problem, the entire problem can be tackled at once. The complexity of large problems and the limitations of human minds do not allow large problems to be treated as huge monoliths.

 As basic aim of problem analysis is to obtain a clear understanding of the needs of the clients and the users.

 Frequently the client and the users do not understand or know all their needs, because the potential of the new system is often not fully appreciated.

 The analysts have to ensure that the real needs of the clients and the users are uncovered, even if they don't know them clearly.

 That is, the analysts are not just collecting and organizing information about the client's organization and its processes, but they also act as consultants who play an active role of helping the clients and users identify their needs.

 For solving larger problems, the basic principle is the time-tested principle of "divide and conquer.“

"divide into smaller pieces, so that each piece can be conquered separately.“

For software design, partition the problem into sub problems and then try to understand each sub problem and its relationship to other sub problems in an effort to understand the total problem.

That is goal is to divide the problem into manageably small pieces that can be solved separately, because the cost of solving the entire problem is more than the sum of the cost of solving all the pieces.

The different pieces cannot be entirely independent of each other, as they together form the system. The different pieces have to cooperate and communicate to solve the larger problem.

Problem partitioning also aids design verification.

The concepts of state and projection can sometimes also be used effectively in the partitioning process.

A state of a system represents some conditions about the system. This approach is sometimes used in real-time software or process-control software.

Projection, different viewpoints of the system are defined and the system is then analyzed from these different perspectives. The different "projections" obtained are combined to form the analysis for the complete system. Analyzing the system from the different perspectives is often easier, as it limits and focuses the scope of the study.

Software design principles are concerned with providing means to handle the complexity of the design process effectively. Effectively managing the complexity will not only reduce the effort needed for design but can also reduce the scope of introducing errors during design.

Following are the principles of Software Design

Problem Partitioning

For small problem, we can handle the entire problem at once but for the significant problem, divide the problems and conquer the problem it means to divide the problem into smaller pieces so that each piece can be captured separately.

For software design, the goal is to divide the problem into manageable pieces.

Benefits of Problem Partitioning

 Software is easy to understand

 Software becomes simple

 Software is easy to test

 Software is easy to modify

 Software is easy to maintain

 Software is easy to expand

These pieces cannot be entirely independent of each other as they together form the system. They have to cooperate and communicate to solve the problem. This communication adds complexity.

Note: As the number of partition increases = Cost of partition and complexity increases

Abstraction

An abstraction is a tool that enables a designer to consider a component at an abstract level without bothering about the internal details of the implementation. Abstraction can be used for existing element as well as the component being designed.

Here, there are two common abstraction mechanisms

• Functional Abstraction

• Data Abstraction

Functional Abstraction

• A module is specified by the method it performs.

• The details of the algorithm to accomplish the functions are not visible to the user of the function.

Functional abstraction forms the basis for Function oriented design approaches.

Data Abstraction

Details of the data elements are not visible to the users of data. Data Abstraction forms the basis for Object Oriented design approaches.

Modularity

Modularity specifies to the division of software into separate modules which are differently named and addressed and are integrated later on in to obtain the completely functional software. It is the only property that allows a program to be intellectually manageable. Single large programs are difficult to understand and read due to a large number of reference variables, control paths, global variables, etc.

The desirable properties of a modular system are:

• Each module is a well-defined system that can be used with other applications.

• Each module has single specified objectives.

• Modules can be separately compiled and saved in the library.

• Modules should be easier to use than to build.

• Modules are simpler from outside than inside.

Advantages and Disadvantages of Modularity

In this topic, we will discuss various advantage and disadvantage of Modularity.

Advantages of Modularity

There are several advantages of Modularity

• It allows large programs to be written by several or different people

• It encourages the creation of commonly used routines to be placed in the library and used by other programs.

• It simplifies the overlay procedure of loading a large program into main storage.

• It provides more checkpoints to measure progress.

• It provides a framework for complete testing, more accessible to test

• It produced the well designed and more readable program.

Disadvantages of Modularity

There are several disadvantages of Modularity

• Execution time maybe, but not certainly, longer

• Storage size perhaps, but is not certainly, increased

• Compilation and loading time may be longer

• Inter-module communication problems may be increased

• More linkage required, run-time may be longer, more source lines must be written, and more documentation has to be done

Modular Design

Modular design reduces the design complexity and results in easier and faster implementation by allowing parallel development of various parts of a system. We discuss a different section of modular design in detail in this section:

1. Functional Independence: Functional independence is achieved by developing functions that perform only one kind of task and do not excessively interact with other modules. Independence is important because it makes implementation more accessible and faster. The independent modules are easier to maintain, test, and reduce error propagation and can be reused in other programs as well. Thus, functional independence is a good design feature which ensures software quality.

It is measured using two criteria:

• Cohesion: It measures the relative function strength of a module.

• Coupling: It measures the relative interdependence among modules.

2. Information hiding: The fundamental of Information hiding suggests that modules can be characterized by the design decisions that protect from the others, i.e., In other words, modules should be specified that data include within a module is inaccessible to other modules that do not need for such information.

The use of information hiding as design criteria for modular system provides the most significant benefits when modifications are required during testing's and later during software maintenance. This is because as most data and procedures are hidden from other parts of the software, inadvertent errors introduced during modifications are less likely to propagate to different locations within the software.

Strategy of Design

A good system design strategy is to organize the program modules in such a method that are easy to develop and latter too, change. Structured design methods help developers to deal with the size and complexity of programs. Analysts generate instructions for the developers about how code should be composed and how pieces of code should fit together to form a program.

Importance :

 If any pre-existing code needs to be understood, organised and pieced together.

 It is common for the project team to have to write some code and produce original programs that support the application logic of the system.

To design a system, there are two possible approaches:

• Top-down Approach

• Bottom-up Approach

Top-down Approach: This approach starts with the identification of the main components and then decomposing them into their more detailed sub-components.

Each system is divided into several subsystems and components. Each of the subsystem is further divided into set of subsystems and components. This process of division facilitates in forming a system hierarchy structure. The complete software system is considered as a single entity and in relation to the characteristics, the system is split into sub-system and component. The same is done with each of the sub-system.

This process is continued until the lowest level of the system is reached. The design is started initially by defining the system as a whole and then keeps on adding definitions of the subsystems and components. When all the definitions are combined together, it turns out to be a complete system.

For the solutions of the software need to be developed from the ground level, top-down design best suits the purpose.

Advantages:

 The main advantage of top down approach is that its strong focus on requirements helps to make a design responsive according to its requirements.

Disadvantages:

• Project and system boundries tends to be application specification oriented. Thus it is more likely that advantages of component reuse will be missed.

• The system is likely to miss, the benefits of a well-structured, simple architecture.

Bottom-up Approach:The design starts with the lowest level components and subsystems. By using these components, the next immediate higher level components and subsystems are created or composed. The process is continued till all the components and subsystems are composed into a single component, which is considered as the complete system. The amount of abstraction grows high as the design moves to more high levels.

By using the basic information existing system, when a new system needs to be created, the bottom up strategy suits the purpose.

Advantages:

 The economics can result when general solutions can be reused.

 It can be used to hide the low-level details of implementation and be merged with top-down technique.

Disadvantages:

• It is not so closely related to the structure of the problem.

• High quality bottom-up solutions are very hard to construct.

• It leads to proliferation of ‘potentially useful’ functions rather than most approprite .

📄 ER Diagram

ER-modeling is a data modeling method used in software engineering to produce a conceptual data model of an information system. Diagrams created using this ER-modeling method are called Entity-Relationship Diagrams or ER diagrams or ERDs.

Purpose of ERD

o The database analyst gains a better understanding of the data to be contained in the database through the step of constructing the ERD.

o The ERD serves as a documentation tool.

o Finally, the ERD is used to connect the logical structure of the database to users. In particular, the ERD effectively communicates the logic of the database to users.

Components of an ER Diagrams

1. Entity

An entity can be a real-world object, either animate or inanimate, that can be merely identifiable. An entity is denoted as a rectangle in an ER diagram. For example, in a school database, students, teachers, classes, and courses offered can be treated as entities. All these entities have some attributes or properties that give them their identity.

Entity Set

An entity set is a collection of related types of entities. An entity set may include entities with attribute sharing similar values. For example, a Student set may contain all the students of a school; likewise, a Teacher set may include all the teachers of a school from all faculties. Entity set need not be disjoint.

2. Attributes

Entities are denoted utilizing their properties, known as attributes. All attributes have values. For example, a student entity may have name, class, and age as attributes.

There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.

There are four types of Attributes:

1. Key attribute

2. Composite attribute

3. Single-valued attribute

4. Multi-valued attribute

5. Derived attribute

1. Key attribute: Key is an attribute or collection of attributes that uniquely identifies an entity among the entity set. For example, the roll\_number of a student makes him identifiable among students.

There are mainly three types of keys:

1. Super key: A set of attributes that collectively identifies an entity in the entity set.

2. Candidate key: A minimal super key is known as a candidate key. An entity set may have more than one candidate key.

3. Primary key: A primary key is one of the candidate keys chosen by the database designer to uniquely identify the entity set.

2. Composite attribute: An attribute that is a combination of other attributes is called a composite attribute. For example, In student entity, the student address is a composite attribute as an address is composed of other characteristics such as pin code, state, country.

3. Single-valued attribute: Single-valued attribute contain a single value. For example, Social\_Security\_Number.

4. Multi-valued Attribute: If an attribute can have more than one value, it is known as a multi-valued attribute. Multi-valued attributes are depicted by the double ellipse. For example, a person can have more than one phone number, email-address, etc.

5. Derived attribute: Derived attributes are the attribute that does not exist in the physical database, but their values are derived from other attributes present in the database. For example, age can be derived from date\_of\_birth. In the ER diagram, Derived attributes are depicted by the dashed ellipse.

3. Relationships

The association among entities is known as relationship. Relationships are represented by the diamond-shaped box. For example, an employee works\_at a department, a student enrolls in a course. Here, Works\_at and Enrolls are called relationships.

Relationship set

A set of relationships of a similar type is known as a relationship set. Like entities, a relationship too can have attributes. These attributes are called descriptive attributes.

Degree of a relationship set

The number of participating entities in a relationship describes the degree of the relationship. The three most common relationships in E-R models are:

1. Unary (degree1)

2. Binary (degree2)

3. Ternary (degree3)

1. Unary relationship: This is also called recursive relationships. It is a relationship between the instances of one entity type. For example, one person is married to only one person.

2. Binary relationship: It is a relationship between the instances of two entity types. For example, the Teacher teaches the subject.

3. Ternary relationship: It is a relationship amongst instances of three entity types. In fig, the relationships "may have" provide the association of three entities, i.e., TEACHER, STUDENT, and SUBJECT. All three entities are many-to-many participants. There may be one or many participants in a ternary relationship.

In general, "n" entities can be related by the same relationship and is known as n-ary relationship.

Cardinality

Cardinality describes the number of entities in one entity set, which can be associated with the number of entities of other sets via relationship set.

Types of Cardinalities

1. One to One: One entity from entity set A can be contained with at most one entity of entity set B and vice versa. Let us assume that each student has only one student ID, and each student ID is assigned to only one person. So, the relationship will be one to one.

Using Sets, it can be represented as:

2. One to many: When a single instance of an entity is associated with more than one instances of another entity then it is called one to many relationships. For example, a client can place many orders; a order cannot be placed by many customers.

Using Sets, it can be represented as:

3. Many to One: More than one entity from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A. For example - many students can study in a single college, but a student cannot study in many colleges at the same time.

Using Sets, it can be represented as:

4. Many to Many: One entity from A can be associated with more than one entity from B and vice-versa. For example, the student can be assigned to many projects, and a project can be assigned to many students.

Using Sets, it can be represented as:

📄 Decision tree

A decision tree is a graph that uses a branching method to illustrate every possible outcome of a decision. Decision trees can be drawn by hand or created with a graphics program or specialized software.

Decision tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.

A decision tree for the concept PlayTennis.

Construction of Decision Tree :

A tree can be “learned” by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node all has the same value of the target variable, or when splitting no longer adds value to the predictions. The construction of decision tree classifier does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery. Decision trees can handle high dimensional data. In general decision tree classifier has good accuracy. Decision tree induction is a typical inductive approach to learn knowledge on classification.

Decision Tree Representation :

Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. An instance is classified by starting at the root node of the tree,testing the attribute specified by this node,then moving down the tree branch corresponding to the value of the attribute as shown in the above figure.This process is then repeated for the subtree rooted at the new node.

The decision tree in above figure classifies a particular morning according to whether it is suitable for playing tennis and returning the classification associated with the particular leaf.(in this case Yes or No).

For example,the instance

(Outlook = Rain, Temperature = Hot, Humidity = High, Wind = Strong )

would be sorted down the leftmost branch of this decision tree and would therefore be classified as a negative instance.

In other words we can say that decision tree represent a disjunction of conjunctions of constraints on the attribute values of instances.

(Outlook = Sunny ^ Humidity = Normal) v (Outllok = Overcast) v (Outlook = Rain ^ Wind = Weak)

There are three different types of nodes: chance nodes, decision nodes, and end nodes. A chance node, represented by a circle, shows the probabilities of certain results. A decision node, represented by a square, shows a decision to be made, and an end node shows the final outcome of a decision path.Decision trees can also be drawn with flowchart symbols.

How to draw a decision tree

To draw a decision tree, first pick a medium. You can draw it by hand on paper or a whiteboard, or you can use special decision tree software. In either case, here are the steps to follow:

1.Start with the main decision. Draw a small box to represent this point, then draw a line from the box to the right for each possible solution or action. Label them accordingly.

2. Add chance and decision nodes to expand the tree as follows:

If another decision is necessary, draw another box.

 If the outcome is uncertain, draw a circle (circles represent chance nodes).If the problem is solved, leave it blank (for now).

From each decision node, draw possible solutions. From each chance node, draw lines representing possible outcomes. If you intend to analyze your options numerically, include the probability of each outcome and the cost of each action.

3.Continue to expand until every line reaches an endpoint, meaning that there are no more choices to be made or chance outcomes to consider. Then, assign a value to each possible outcome. It could be an abstract score or a financial value. Add triangles to signify endpoints.

Decision tree analysis example

By calculating the expected utility or value of each choice in the tree, you can minimize risk and maximize the likelihood of reaching a desirable outcome.

To calculate the expected utility of a choice, just subtract the cost of that decision from the expected benefits. The expected benefits are equal to the total value of all the outcomes that could result from that choice, with each value multiplied by the likelihood that it’ll occur. Here’s how we’d calculate these values for the example we made above:

When identifying which outcome is the most desirable, it’s important to take the decision maker’s utility preferences into account. For instance, some may prefer low-risk options while others are willing to take risks for a larger benefit.

When you use your decision tree with an accompanying probability model, you can use it to calculate the conditional probability of an event, or the likelihood that it’ll happen, given that another event happens. To do so, simply start with the initial event, then follow the path from that event to the target event, multiplying the probability of each of those events together.

In this way, a decision tree can be used like a traditional tree diagram, which maps out the probabilities of certain events, such as flipping a coin twice.

Strengths and Weakness of Decision Tree approach

The strengths of decision tree methods are:

 Decision trees are able to generate understandable rules.

 Decision trees perform classification without requiring much computation.

 Decision trees are able to handle both continuous and categorical variables.

 Decision trees provide a clear indication of which fields are most important for prediction or classification.

The weaknesses of decision tree methods :

• Decision trees are less appropriate for estimation tasks where the goal is to predict the value of a continuous attribute.

• Decision trees are prone to errors in classification problems with many class and relatively small number of training examples.

• Decision tree can be computationally expensive to train. The process of growing a decision tree is computationally expensive. At each node, each candidate splitting field must be sorted before its best split can be found. In some algorithms, combinations of fields are used and a search must be made for optimal combining weights. Pruning algorithms can also be expensive since many candidate sub-trees must be formed and compared.

📄 Decision Table and Structured English

Decision table is a brief visual representation for specifying which actions to perform depending on given conditions. The information represented in decision tables can also be represented as decision trees or in a programming language using if-then-else and switch-case statements.

A decision table is a good way to settle with different combination inputs with their corresponding outputs and also called cause-effect table. Reason to call cause-effect table is a related logical diagramming technique called cause-effect graphing that is basically used to obtain the decision table.

Importance of Decision Table:

 Decision tables are very much helpful in test design technique.

 It helps testers to search the effects of combinations of different inputs and other software states that must correctly implement business rules.

 It provides a regular way of stating complex business rules, that is helpful for developers as well as for testers.

 It assists in development process with developer to do a better job. Testing with all combination might be impractical.

 A decision table is basically an outstanding technique used in both testing and requirements management.

 It is a structured exercise to prepare requirements when dealing with complex business rules.

 It is also used in model complicated logic.

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.

Decision Table in test designing:

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.

Blank Decision Table

CONDITIONS STEP 1 STEP 2 STEP 3 STEP 4

Condition 1

Condition 2

Condition 3

Condition 4

Decision Table: Combinations

CONDITIONS STEP 1 STEP 2 STEP 3 STEP 4

Condition 1 Y Y N N

Condition 2 Y N Y N

Condition 3 Y N N Y

Condition 4 N Y Y N

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

Advantage of Decision Table:

• Any complex business flow can be easily converted into the test scenarios & test cases using this technique.

• Decision tables work iteratively that means the table created at the first iteration is used as input table for next tables. The iteration is done only if the initial table is not satisfactory.

• Simple to understand and everyone can use this method design the test scenarios & test cases.

• It provide complete coverage of test cases which help to reduce the rework on writing test scenarios & test cases.

• These tables guarantee that we consider every possible combination of condition values. This is known as its completeness property.

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

Structured English is one more tool available to the analyst. It comes as an aid against the problems of ambiguous language in stating condition and actions in decisions and procedures. Here no trees or tables are employed, rather with narrative statements a procedure is described. Thus it does not show but states the decision rules. The analyst is first required to identify the conditions that occur in the process, subsequent decisions, which are to be made and the alternative actions to be taken.

Here the steps are clearly listed in the order in which they should be taken. There are no special symbols or formats involved unlike in the case of decision trees and tables, also the entire procedure can be stated quickly as only English like statements are used.

Structured English borrows heavily from structured programming as it uses logical construction and imperative statements designed to carry out instructions for actions. Using "IF", "THEN", "ELSE" and "So" statement decisions are made. In this structured description terms from the data dictionary are widely used which makes the description compact and straight.

Developing Structured Statements

Three basic types of statements are employed to describe the process.

1. Sequence Structures - A sequence structure is a single step or action included in a process. It is independent of the existence of any condition and when encountered it is always taken. Usually numerous such instructions are used together to describe a process.

2. Decision Structures - Here action sequences described are often included within decision structures that identify conditions. Therefore these structures occur when two or more actions can be taken as per the value of a specific condition. Once the condition is determined the actions are unconditional.

3. Iteration Structures- these are those structures, which are repeated, in routing operations such as DO WHILE statements.

📄 Functional vs Object oriented approach

A programming paradigm is a style, or “way, ” of programming. Programming paradigms differ from one another based on the features and the style they support. There are several features that determine a programming paradigm such as modularity, objects, interrupts or events, control flow etc. Every programming paradigm has its own advantage so, it better to know where to use it before actually using it.

Object-oriented languages are good when you have a fixed set of operations on things, and as your code evolves, you primarily add new things. This can be accomplished by adding new classes which implement existing methods, and the existing classes are left alone.

Functional languages are good when you have a fixed set of things, and as your code evolves, you primarily add new operations on existing things. This can be accomplished by adding new functions which compute with existing data types, and the existing functions are left alone.

It is also possible to use both the programming paradigms according to our own need. As we have got languages like python, java, etc that supports both object oriented concept and are also functional by supporting various inbuilt functions.

Function Oriented Design

In function-oriented design, the system is comprised of many smaller sub-systems known as functions. These functions are capable of performing significant task in the system. The system is considered as top view of all functions.

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

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

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

Design Process

• The whole system is seen as how data flows in the system by means of data flow diagram.

• DFD depicts how functions changes data and state of entire system.

• The entire system is logically broken down into smaller units known as functions on the basis of their operation in the system.

• Each function is then described at large.

Object Oriented Design

Object oriented design works around the entities and their characteristics instead of functions involved in the software system. This design strategies focuses on entities and its characteristics. The whole concept of software solution revolves around the engaged entities.

Let us see the important concepts of Object Oriented Design:

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

• Classes - A class is a generalized description of an object. An object is an instance of a class. Class defines all the attributes, which an object can have and methods, which defines the functionality of the object.

In the solution design, attributes are stored as variables and functionalities are defined by means of methods or procedures.

• Encapsulation - In OOD, the attributes (data variables) and methods (operation on the data) are bundled together is called encapsulation. Encapsulation not only bundles important information of an object together, but also restricts access of the data and methods from the outside world. This is called information hiding.

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

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

➡️ Difference between functional and Object oriented design

Coding

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:

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:

1. Indentation: Proper and consistent indentation is essential in producing easy to read and maintainable programs.

Indentation should be used to:

o Emphasize the body of a control structure such as a loop or a select statement.

o Emphasize the body of a conditional statement

o 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.