Topic 1: Notes
Overview

Before 1975 most software organizations used no specific techniques, each individual worked on his or her own way. Major developments were made between 1978 and 1985 which led to development of the structured or classical paradigm. Techniques in classical paradigm included structured systems analysis (DFDs, structured programming and structured testing)

With time these techniques proved less successful in several respects:-

As the computing power of computers increased, programmers tried to solve more and more complex problems using computers. Programming languages developed but they still used old methodologies. These methodologies had their shortcomings. For instance writing of large programs using procedural programming is very cumbersome.

This problem and others resulted in what is known as the software crisis. The software crisis manifested itself in the following ways

Programs delivered after schedule

2. Program development running over budget
3. Programs developed did not exactly meet specifications
4. Programs developed were not easy to maintain
5. Programs developed were not easy to modify
6. Programs developed were not scalable

These techniques were unable to deal with large-scale products that required several lines of code and could not scale up to sufficiently handle the development of today's large products. It did not solve the problem of cutting down the budget of post-delivery maintenance to 30%. 70%-80% or more time and effort were spent on post-delivery maintenance. The major reason for the limited success of the classical paradigm is that the classical techniques are either operation oriented or attribute (data) oriented but not both. Some classical techniques such as DFDs are operation oriented-i.e. concentrate on the operations of the product others techniques are attribute oriented. In contrast, the OO paradigm considers both attributes and operations to be equally important.

The basic components of a software product are the operations and attributes on which those operations operates. Most programs developed made a distinction between the code and the data the code was operating on. OOP treats data as a critical element in the program development and does not allow it to flow freely around the system. It ties data more closely to the functions that operate on it, and protects it from accidental modification from outside functions. OOP allows decomposition of a problem into a number of entities called objects and then builds data and functions around these objects.

Object-oriented programming itself was created to solve some of the problems inherent in the development of large programs. Certainly OOP helps the design process because objects in the program correspond with objects in the user's world. However, OOP by itself does not tell us what the program should do; it comes into play only after the project's goals have been determined. We need an initial phase that focuses on the program's users and captures their needs. Once this is accomplished we can translate it into an object-oriented program design. But how do we perform the initial step of figuring out what the users really need? These days, new software is usually object-oriented. That is, the software is written using an abstraction called an object. There is, naturally, much more to commercial software development than simply writing lines of code: there is investigation of the business requirements, analysis of the problem, design of the solution, and so on. Objects should be used at every stage of the development because they reduce the amount of information that has to be understood and improve the communication between members of the development team.

OBJECT ORIENTED SYSTEMS ANALYIS AND DESIGN

System analysis and design is a step by step process for developing high quality information systems. An information system combines information technology people and a data to support business requirements –for example business transactions, improve company productivity etc. A system is a set of related components that work together to produce specific results.

Overview of systems development methods (methodologies) (Software development process)

A methodology is a description of the steps a development team should go through in order to produce a high-quality system. A methodology also describes what should be produced (documents, diagrams, code, etc.) and what form the products should take (for example, content, icons, coding style).

Many methodologies exist for developing information systems. The most popular methods are

1) **Structured systems analysis and Design.**

This is a traditional method but is still widely. This method uses a series of phases called the systems development lifecycle to plan, analyze, design, implement and support an information system. It is based on an overall plan similar to blue print for constructing a building and it is a predictive approach. It uses a set of process models to describe a system graphically. Structured analysis uses the system development lifecycle (SDLC) to plan and manage the system development process SDLC describes activities and functions that system developers perform regardless of the the approach the use.

The SDLC model includes five steps

- **System planning:-**Involves describing a problem or desired change to information system or business process. Here you perform feasibility studies and preliminary investigations to evaluate on IT related opportunities. Feasibility studies reviews

anticipated costs and benefits and recommends a course of action based on an operational, technical and economical and time factor.

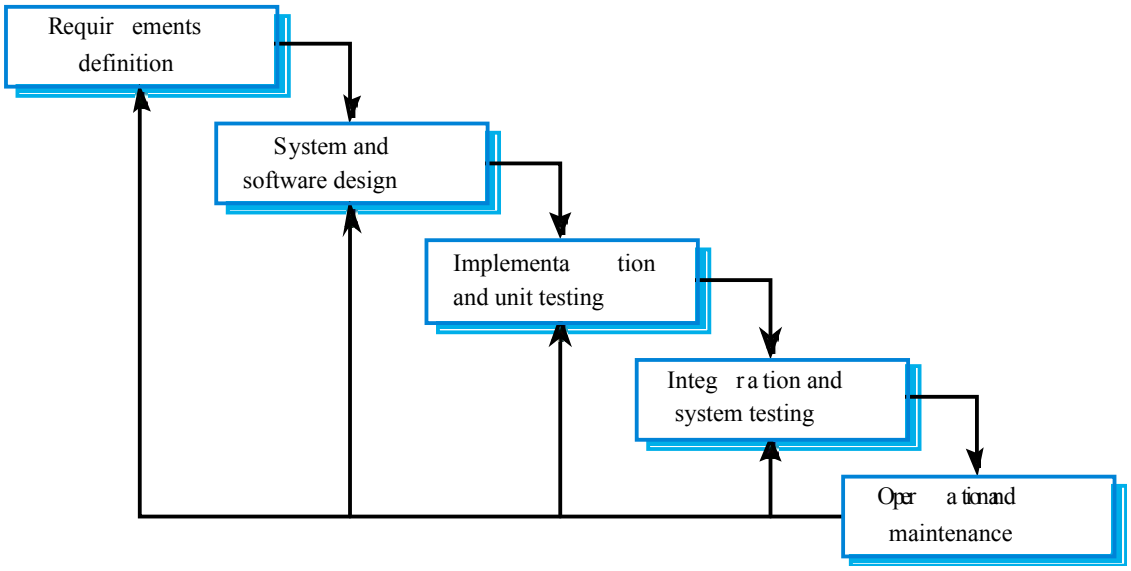
- **Systems analysis**:- The purpose of this phase is to build a logical model of the new system. Here you investigate and document what the new system must do to satisfy the users.
- **System design:-**This phase purpose is to create a physical model that will satisfy all documented requirements for the system. Deliverables for this phase is design specifications.
- **System implementation**-During this phase the new system is constructed- programs are written, tested and documented. A completely functioning and documented program is delivered. This phase also includes a systems evaluation to determine whether the system operates properly and if costs and benefits are within expectations.
- **Systems support and security.:-**IT staff and maintains and enhances and protects the system, maintenance changes, correct errors and adapt to changes in the business environment.

Approaches to system development lifecycle (LifeCycle models)

SDLC describes activities and functions that system developers perform during the system development lifecycle.

i) Waterfall model

Development is supposed to proceed linearly through the phases of requirements analysis, design, implementation, testing (validation), integration and maintenance. Phases of the waterfall and how they are conducted are shown in the diagram below



The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. One phase has to be complete before moving onto the next phase. The standard waterfall model is associated with the failure or cancellation of a number of large

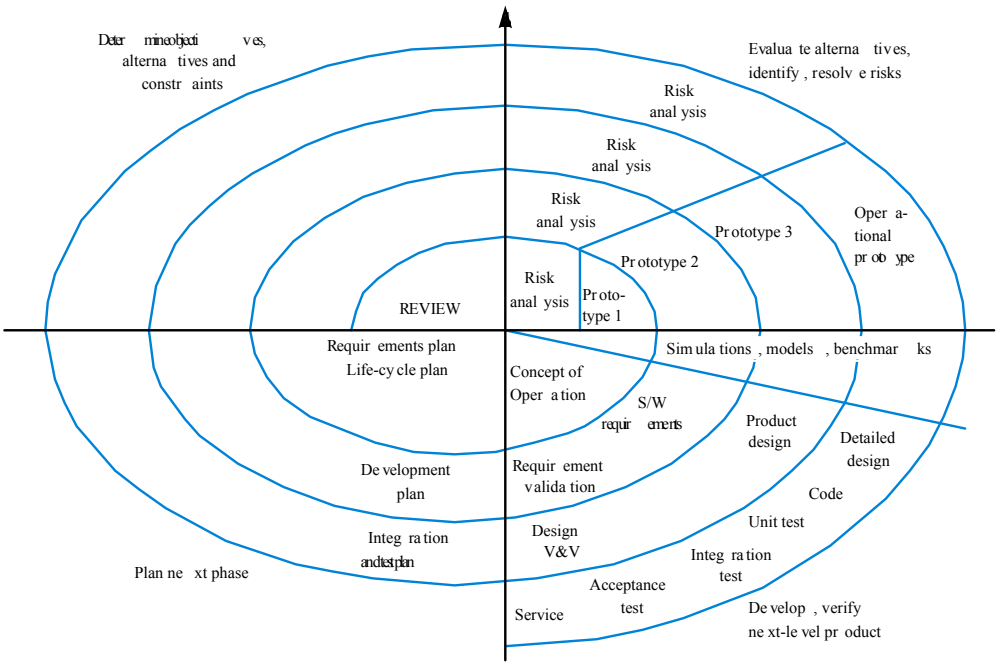
systems. It can also be very expensive. As a result, the software development community has experimented with a number of alternative approaches, including the spiral model and the star model. However it is possible have to paths for feedback loops within waterfall but iterations but can very costly.

ii) Spiral model

The spiral model (Boehm, 1988) uses incremental development, with the aim of managing risk. In the spiral model, developers define and implement features in order of decreasing priority. An initial version of the system is developed, and then repetitively modified based on input received from customer evaluations. The development of each version of the system is carefully designed using the steps involved in the waterfall model. With each iteration around the spiral (beginning at the centre and working outward), progressively more complete versions of the system are built. The Spiral Model is made up of the following steps:

- **Project Objectives.** Similar to the system conception phase of the Waterfall Model. Objectives are determined, possible obstacles are identified and alternative approaches are weighed.
- **Risk Assessment**. Possible alternatives are examined by the developer, and associated risks/problems are identified. Resolutions of the risks are evaluated and weighed in the consideration of project continuation. Sometimes prototyping is used to clarify needs.
- **Engineering & Production.** Detailed requirements are determined and the software piece is developed.
- **Planning and Management**. The customer is given an opportunity to analyze the results of the version created in the Engineering step and to offer feedback to the developer.

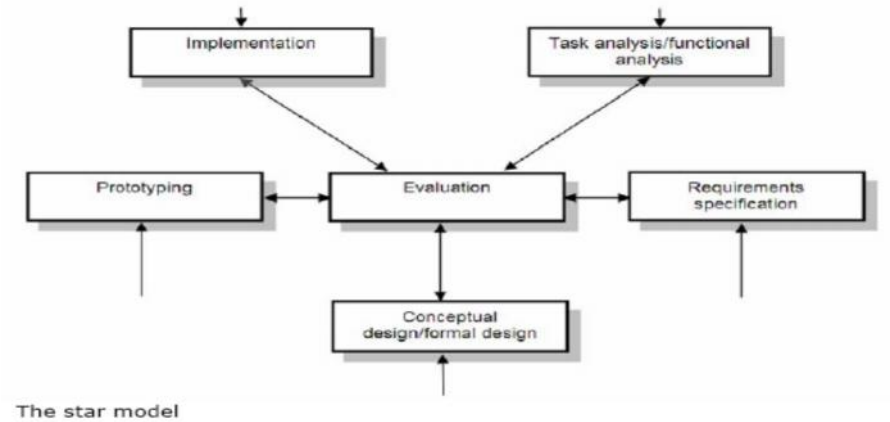
The Spiral model relies heavily on prototyping. When using prototyping, the developer builds a simplified version of the proposed system and presents it to the customer for consideration as part of the development process. The customer in turn provides feedback to the developer, who goes back to refine the system requirements to incorporate the additional information. Often, the prototype code is thrown away and entirely new programs are developed once requirements are identified. We will look at prototyping methods in a later chapter.



iii) The Star Model

The star model (Hartson & Hix, 1989) emphasizes that the design of interactive systems typically does not follow a specific order of steps. Evaluation represents the central phase in the development cycle. Development can start from any point in the star (as shown by the entry arrows) and any stage can be followed by any other stage. Evaluation is always done before moving to a new stage. The requirements, design and product gradually evolve, becoming increasingly well-defined.

The star model can give the user a significant role throughout the project since evaluation is central to the cycle. It is particularly oriented towards the development of interactive systems that will be usable by people. Evaluation can be based on any representation of the system. A variety of representations may be used during the development, including sketches, scenarios, prototypes and formal models.



The star model

Prototyping

Many approaches to systems development incorporates some iterations, for example analysis may involve a series of tasks that are repeated iteratively until the analysis model are deemed complete. A prototype is a system or partially complete system that is built quickly to explore some aspects of the systems requirements and that is not intended as the final working system. A prototype may be constructed with the following objectives in mind to investigate user requirements. To determine whether a particular implementation platform can support certain processing requirements.

Iterative and incremental development.

A common agreement in many current approaches to software development is to adapt an interactive cycle. In such an approach the development project comprises of much iteration, each viewed as a mini program in its own right. Iteration may provide new elements of functionality that improve some aspects of operation for previously delivered functionality. Iterative development results in incremental delivery. Spiral and star models can be viewed as supporting incremental delivery. Modern nature of development approaches are categorized as iterative nature of development process and the incremental nature of the system delivery.

2) Object Oriented System Analysis and Design

The OSSAD methodology uses an object-oriented perspective rather than a functional perspective as in the SSAD methodology. An object is a person, place or thing initially drawn from the problem domain which has three aspects to it: what it knows (its identity and certain attributes), who it knows (relationships to other objects) and what it does (its methods it is responsible for performing on its data).

The object-oriented approach to system development views a system as a collection of interacting objects that work together to accomplish tasks. Conceptually there are no separate processes or programs; there are no separate data entities or files. The system in operation consists of objects. An object is a thing in the computer system that is capable of responding to messages. Consequently, the OOAD methodology can be broken up into two major areas

- **Object-oriented analysis** is the process of developing an object-oriented model of the problem domain where the initial objects represent the entities and methods related to the problem that needs to be resolved. These identified objects represent entities, and possess relationships and methods that are necessary for the problem to be resolved.
- **Object-oriented design** is the process of developing an object-oriented model of the system necessary to meet the specified requirements. So in this methodology we think in terms of things (objects) rather than functions. The analysts and programmers must think in terms of things (objects) rather than processes or functions.

The object model is based on principles including abstraction, encapsulation, modularity, hierarchy, concurrency, and persistence, and follows a repetitive and step by step approach to systems development. The major focus of the object model is object decomposition as opposed to functional decomposition, where a complex system is decomposed into several objects. An object-oriented system will consist of these various objects each of which will collaborate and cooperate with other objects to achieve specified tasks. Consequently, object decomposition allows the analyst to break down the problem into separate and more manageable parts

As opposed to the structured systems development process, object-oriented development follows an iterative and incremental lifecycle. The four phases of the object-oriented lifecycle are inception, elaboration, construction, and transition. Consequently, an information system evolves by going through several iterations, each of which consists of all three primary tasks analysis, design, and construction. Requirements are defined by use cases which are object-oriented tools that describe scenarios of the interactions between the system and its users.

Justifications for Objected-Oriented Programming

Object-oriented development allows you to think in real-world terms. Here are some of the justifications typically given for object orientation

•**Objects are easier for people to understand:** This is because the objects are derived from the business that we're trying to automate, rather than being influenced too early by computer-based procedures or data storage requirements. For example, in a bank system, we program in terms of bank accounts, bank tellers and customers, instead of diving straight into account records, deposit and withdrawal procedures, and loan qualification algorithms.

•**Specialists can communicate better**: Over time, the software industry has constructed career ladders that newcomers are expected to climb gradually as their knowledge and experience increases. Typically, the first rung is programmer: fixing faults (bugs) in the code written by others. The second rung is senior programmer: writing the code itself. The third is designer: deciding what code needs to be written. Finally comes the role of analyst: talking to customers to discover what they need and then writing down a specification of what the finished system must be able to do.

•**Data and processes are not artificially separated:** In traditional methods, the data that needs to be stored is separated early on from the algorithms that operate on that data and they are then developed independently. This can result in the data being in inconvenient formats or inconvenient locations, with respect to the processes that need access. With object oriented development, data and processes are kept together in small, easy-to-manage packages; data is never separated from the algorithms. We also end up with less complex code that is less sensitive to changes in customer requirements.

•**Code can be reused more easily**: With the traditional approach, we start with the problem that needs to be solved and allow that problem to drive the entire development. We end up with a monolithic solution to today's problem. But tomorrow always brings a different problem to solve; no matter how close the new problem is to the last one we dealt with, we're unlikely to be

able to break open our monolithic system and make it fit – we hamper ourselves by allowing a single problem to influence every part of the system.

•**Post-delivery maintenance-** changes need to be effected only within the object. This reduces chances of introducing regression fault- fault introduced in a part as result of changes to another part.

•**Well-designed objects are independent units-objects** consists of both attributes and operations performed on the attributes. If the operations to be performed on the attributes are included in the object, then the object is considered to be conceptually independent.ie

•**OO reduces the level of complexity-** classical paradigm implements products as a set of modules but conceptual it is essentially a single unit. This made the classical paradigm less successful when applied to large products. In contrast, when OO paradigm is used correctly, the resulting product consists of a number of small, large independent units. This reduces the level of complexity of a software product, hence simplifies both development and maintenance.

Benefits Of OOP

OOP offers several benefits to both the program designer and the user. Object-orientation contributes to the solution of many problems associated with the development and quality of software products. The new technology promises greater programmer productivity, better quality of software and lesser maintenance cost. The principal advantages are:

1. Through inheritance, we can eliminate redundant code and extend the use of existing classes.
2. We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch. This leads to saving of development time and higher productivity.
3. The principle of data hiding helps the programmer to build secure programs that cannot be invaded by code in other parts of the program.
4. It is possible to have multiple instances of an object to co-exist without any interference.
5. It is possible to map objects in the problem domain to those in the program.
6. It is easy to partition the work in a project based on objects.
7. The data-centered design approach enables us to capture more details of a model in implement able form.
8. Object-oriented systems can be easily upgraded from small to large systems.
9. Message passing techniques for communication between objects makes the interface descriptions with external systems much simpler.
10. Software complexity can be easily managed.

Applications of OOP

Applications of OOP are beginning to gain importance in many areas. The most popular application of object-oriented programming, up to now, has been in the area of user interface

design such as windows. Hundreds of windowing systems have been developed, using the OOP techniques.

Real-business systems are often much more complex and contain many more objects with complicated attributes and methods. OOP is useful in these types of applications because it can simplify a complex problem. The promising areas for application of OOP include:

1. Real-time systems
2. Simulation and modeling
3. Object-oriented databases
4. Hypertext, hypermedia and expertext
5. AI and expert systems
6. Neural networks and Parallel programming Decision support and office automation systems