UNIT-III

SYSTEM DEVELOPMENT

System Concepts – Types of Systems – Modern Information Systems – System Development Life Cycle – Completing the system development process - Modeling and Design Systems: Structured and Object-Oriented Methodologies – Computer-Aided-Software-Engineering (CASE), Alternative System-Building Approaches: Traditional System life Cycle and Prototyping.

SYSTEM DEVELOPMENT

System Concepts

The term system is derived from the Greek word *systema*, which means an organized relationship among functioning units or components. A system exists because it is designed to achieve one or more objectives.

The study of systems concepts, then, has three basic implications:

1. A system must be designed to achieve a predetermined objective.
2. Interrelationships and interdependence must exist among the components.
3. The objectives of the organization as a whole have a higher priority than the objectives of its subsystems. For example, computerizing personnel applications must conform to the organization’s policy on privacy, confidentiality and security, as will as making selected data (e.g. payroll) available to the accounting division on request.

Characteristics of a System

Organization

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

Interaction

Interaction refers to the manner in which each component functions with other components of the system. In an organization, for example, purchasing must interact with production, advertising with sales and payroll with personnel. In a computer system, the central processing unit must interact with the input device to solve a problem. In turn, the main memory holds programs and data that the arithmetic unit uses for computation. The interrelationship between these components enables the computer to perform.

Interdependence

Interdependence means that parts of the organization or computer system depend on one another. They are coordinated and linked together according to a plan. One subsystem depends on the input of another subsystem for proper functioning: that is, the output of one subsystem is the required input for another subsystem. This interdependence is crucial in systems work.

Integration

Integration refers to the holism of systems. Synthesis follows analysis to achieve the central objective of the organization. Integration is concerned with how a system is tied together. It is more than sharing a physical part or location. It means that parts of the system work together within the system even though each part performs a unique function. Successful integration will typically produce a synergistic effect and greater total impact than if each component works separately.

Central objective

The last characteristic of a system is its central objective. Objectives may be real or stated. Although a stated objective may be the real objective, it is not uncommon for an organization to state one objective and operate to achieve another. The important point is that users must know the central objective of a computer application early in the analysis for a successful design and conversion. Political as well as organizational considerations often cloud the real objective. This means that the analyst must work around such obstacles to identify the real objective of the proposed change.

Elements of a System

In most cases, systems analysts operate in a dynamic environment where change is a way of life. The environment may be a business firm, a business application, or a computer system. To reconstruct a system, the following key elements must be considered:

1. Outputs and inputs.
2. Processor(s).
3. Control.
4. Feedback.
5. Environment.
6. Boundaries and interface

Outputs and Inputs

A major objective of a system is to produce an output that has value to its user. Whatever the nature of the output (goods, services, or information), it must be in line with the expectations of the intended user. Inputs are the elements (material, human resources, and information) that enter the system for processing. Output is the outcome of processing. A system feeds on input to produce output in much the same way that a business brings in human, financial, and material resources to produce goods and services. It is important to point out here that determining the output is a first step in specifying the nature, amount, and regularity of the input needed to operate a system. For example, in systems analysis, the first concern is to determine the user's requirements of a proposed computer system – that is, specification of the output that the computer is expected to provide for meeting user requirements.

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 totally or partially, depending on the specifications of the output. This means that as the output specifications change so does the processing. In some cases, input is also modified to enable the processor to handle 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. In an organizational context, management as a decision – making body controls the inflow, handling and outflow of activities that affect the welfare of the business. In a computer system, the operating system and accompanying software influence the behaviour of the system. Output specifications determine what and how much input is needed to keep the system in balance.

Feedback

Control in a dynamic system is achieved by feedback. Feedback measures output against a standard in some form of cybernetic procedure that includes communication and control. Output information is fed back to the input and / or to management (Controller) for deliberation. After the output is compared against performance standards, changes can result in the input or processing and consequently, the output.

Feedback may be positive or negative, routing or informational. Positive feedback reinforces the performance of the system. It is routine in nature. Negative feedback generally provides the controller with information for action. In systems analysis, feedback is important in different ways. During analysis, the user may be told that the problems in a given application verify the initial concerns and justify the need for change.

Another form of feedback comes after the system is implemented. The user informs the analyst about the performance of the new installation. This feedback often results in enhancements to meet the user’s requirements.

Environment

The environment is the “supra-system” within which an organization operates. It is the source of external elements that impinge on the system. In fact, it often determines how a system must function. For example, the organization’s environment, consisting of vendors, competitors, and others, may provide constraints and, consequently, influence the actual performance of the business.

Boundaries and interface

A system should be defined by its boundaries – the limits that identify its components, processes and interrelationship when it interfaces with another system. For example, a teller system in a commercial bank is restricted to the deposits, withdrawals and related activities of customers checking and savings accounts. It may exclude mortgage foreclosures, trust activities, and the like.

Systems analysts need to know several other important systems concepts such as:

- Decomposition
- Modularity
- Coupling
- Cohesion

Decomposition

Decomposition is the process of breaking down a system into its smaller components. These components may themselves be systems (subsystems) and can be broken down into their components as well. How does decomposition aid understand of a system? It results in smaller and less complex pieces that are easier to understand than larger, complicated pieces. Decomposing a system also allows us to focus on one particular part of a system, making it easier to think of how to modify that one part independently of the entire system. Decomposition is a technique that allows the systems analyst to:

- Break a system into small, manageable, and understandable subsystems
- Focus attention on one area (subsystem) at a time, without interference from other areas
- Concentrate on the part of the system pertinent to a particular group of users, without confusing users with unnecessary details
- Build different parts of the system at independent times and have the help of different analysts

Modularity

Modularity is a direct result of decomposition. It refers to dividing a system into chunks or modules of a relatively uniform size. Modules can represent a system simply, making it easier to understand and easier to redesign and rebuild. For example, each of the separate subsystem modules for the MP3 player in the above figure shows how decomposition makes it easier to understand the overall system.

Coupling

Coupling means that subsystems are dependent on each other. Subsystems should be as independent as possible. If one subsystem fails and other subsystems are highly dependent on it, the others will either fail themselves or have problems functioning. Looking at the above figure, we would say the components of a portable MP3 player are tightly coupled. The best example is the control system, made up of the printed circuit board and its chips. Every function the MP3 player can perform is enabled by the board and the chips. A failure in one part of the circuit board would typically lead to replacing the entire board rather than attempting to isolate the problem on the board and fix it. Even though repairing a circuit board in an MP3 player is certainly possible, it is typically not cost-effective; the cost of the labor expended to diagnose and fix the problem may be worth more than the value of the circuit board itself.

In a home stereo system, the components are loosely coupled because the subsystems, such as the speakers, the amplifier, the receiver, and the CD player, are all physically separate and function independently. If the amplifier in a home stereo system fails, only the amplifier needs to be repaired.

Cohesion

Cohesion is the extent to which a subsystem performs a single function. In the MP3 player example, supplying power is a single function.

TYPES OF SYSTEMS

The frame of reference within which one views a system is related to the use of the systems approach for analysis. Systems have been classified in different ways. Common classifications are:

1. physical or abstract,
2. open or closed, and
3. “man – made” information systems.

Physical or abstract systems

Physical systems are tangible entities that may be static or dynamic in operation. For example, the physical parts of the computer center are the officers, desks, and chairs that facilitate operation of the computer. They can be seen and counted; they are static. In contrast, a programmed computer is a dynamic system. Data, programs, output, and applications change as the user’s demands or the priority of the information requested changes.

Abstract systems are conceptual or non-physical entities. They may be as straightforward as formulas of relationships among sets of variables or models – the abstract conceptualization of physical situations. A model is a representation of a real or a planned system. The use of models makes it easier for the analyst to visualize relationships in the system under study. The objective is to point out the significant elements and the key interrelationships of a complex system.

Open or Closed Systems

Another classification of systems is based on their degree of independence. An open system has many interfaces with its environment. It permits interaction across its boundary; it receives inputs from and delivers outputs to the outside. An information system falls into this category, since it must adapt to the changing demands of the user. In contrast, a closed system is isolated from environmental influences. In reality, a completely closed system is rare. In systems analysis, organizations, applications and computers are invariably open, dynamic systems influenced by their environment.

Man – Made Information Systems

Ideally, information reduces uncertainty about a state or event. For example, information that the wind is calm reduces the uncertainty that the boat trip will be pleasant. An information system is the basis for interaction between the user and the analyst. It provides instruction, commands and feedback. It determines the nature of the relationships among decision-makers. In fact, it may be viewed as a decision center for personnel at all levels. From this basis, an information system may be defined as a set of devices, procedures and operating systems designed around user-based criteria to produce information and communicate it to the user for planning, control and performance. In systems analysis, it is important to keep in mind that considering an alternative system means improving one or more of these criteria.

What is System Analysis and Design?

System analysis and design is a method used by companies ranging from IBM to PepsiCo to Sony to create and maintain information systems that perform basic business functions such as keeping track of customer names and addresses, processing orders, and paying employees.
The main goal of systems analysis and design is to improve organizational systems, typically through applying software that can help employees accomplish key business tasks more easily and efficiently. As a systems analyst, you will be at the center of developing this software. The analysis and design of information systems are based on:

- Your understanding of the organization 's objectives, structure, and processes
- Your knowledge of how to exploit information technology for advantage

System Development Life Cycle

A software life cycle model (also called process model) is a descriptive and diagrammatic representation of the software life cycle. A life cycle model represents all the activities required to make a software product transit through its life cycle phases. It also captures the order in which these activities are to be undertaken. In other words, a life cycle model maps the different activities performed on a software product from its inception to retirement. Different life cycle models may map the basic development activities to phases in different ways.

System Development Life Cycle (SDLC) is a series of six main phases to create a hardware system only, a software system only or a combination of both to meet or exceed customer's expectations.

1- System Planning

The Planning phase is the most crucial step in creating a successful system, during this phase you decide exactly what you want to do and the problems you're trying to solve, by:

- Defining the problems, the objectives and the resources such as personnel and costs.
- Studying the ability of proposing alternative solutions after meeting with clients, suppliers, consultants and employees.
- Studying how to make your product better than your competitors'.

After analyzing this data, you will have three choices: develop a new system, improve the current system or leave the system as it is.

2- System Analysis

The end-user’s requirements should be determined and documented, what their expectations are for the system, and how it will perform. A feasibility study will be made for the project as well, involving determining whether it’s organizationally, economically, socially, technologically feasible. it’s very important to maintain strong communication level with the clients to make sure you have a clear vision of the finished product and its function.

3- System Design

The design phase comes after a good understanding of customer’s requirements, this phase defines the elements of a system, the components, the security level, modules, architecture and the different interfaces and type of data that goes through the system.

A general system design can be done with a pen and a piece of paper to determine how the system will look like and how it will function, and then a detailed and expanded system design is produced, and it will meet all functional and technical requirements, logically and physically.

4- Implementation and Deployment

This phase comes after a complete understanding of system requirements and specifications, it’s the actual construction process after having a complete and illustrated design for the requested system.

In the Software Development Life Cycle, the actual code is written here, and if the system contains hardware, then the implementation phase will contain configuration and fine-tuning for the hardware to meet certain requirements and functions.

In this phase, the system is ready to be deployed and installed in customer’s premises, ready to become running, live and productive, training may be required for end users to make sure they know how to use the system and to get familiar with it, the implementation phase may take a long time and that depends on the complexity of the system and the solution it presents.

5- System Testing and Integration

Bringing different components and subsystems together to create the whole integrated system, and then Introducing the system to different inputs to obtain and analyze its outputs and behavior and the way it functions. Testing is becoming more and more important to ensure customer’s satisfaction, and it requires no knowledge in coding, hardware configuration or design.

Testing can be performed by real users, or by a team of specialized personnel, it can also be systematic and automated to ensure that the actual outcomes are compared and equal to the predicted and desired outcomes.

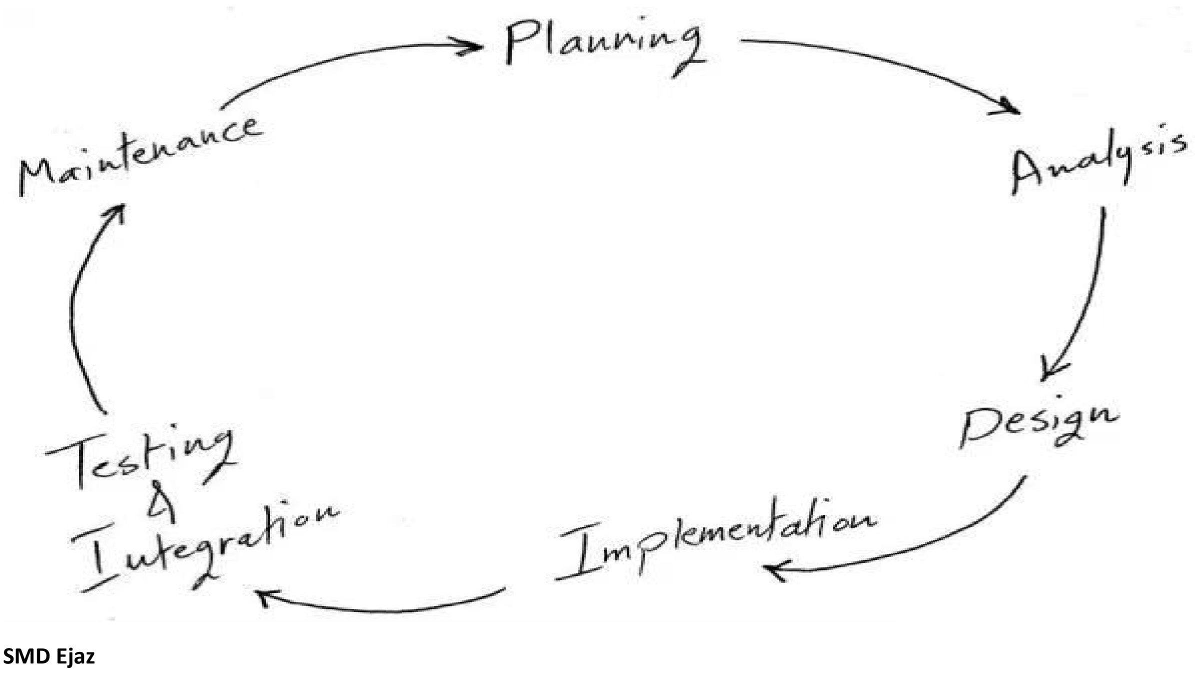
6- System Maintenance

In this phase, periodic maintenance for the system will be carried out to make sure that the system won’t become obsolete, this will include replacing the old hardware and continuously evaluating system’s performance, it also includes providing latest updates for certain components to make sure it meets the right standards and the latest technologies to face current security threats.

These are the main six phases of the System Development Life Cycle, and it’s an iterative process for each project. It’s important to mention that excellent communication level should be maintained with the customer, and Prototypes are very important and helpful when it comes to meeting the requirements. By building the system in short iterations; we can guarantee meeting the customer’s requirements before we build the whole system.

Many models of system development life cycle came up from the idea of saving effort, money and time, in addition to minimizing the risk of not meeting the customer’s requirement at the end of project, some of these models are SDLC Iterative Model, and SDLC Agile Model.

Figure 1 System Development Life Cycle Phases:



THE CORE ACTIVITIES IN THE SYSTEMS DEVELOPMENT PROCESS

Introduction

New information systems are an outgrowth of organizational problem solving. A new information system is built as a solution to some type of problem or set of problems the organization perceives it is facing. The problem may be one in which managers and employees realize that the organization is not performing as well as expected or that the organization should take advantage of new opportunities to perform more successfully.

The activities that go into producing an information system solution to an organizational problem or opportunity are called systems development. Systems development is a structured kind of problem solved with distinct activities**. These activities consist of systems analysis, systems design, programming, testing, conversion, and production and maintenance.**

Figure 2 illustrates the systems development process. The systems development activities depicted usually take place in sequential order. But some of the activities may need to be repeated or some may take place simultaneously depending on the approach to system building that is being employed.

1. Systems Analysis

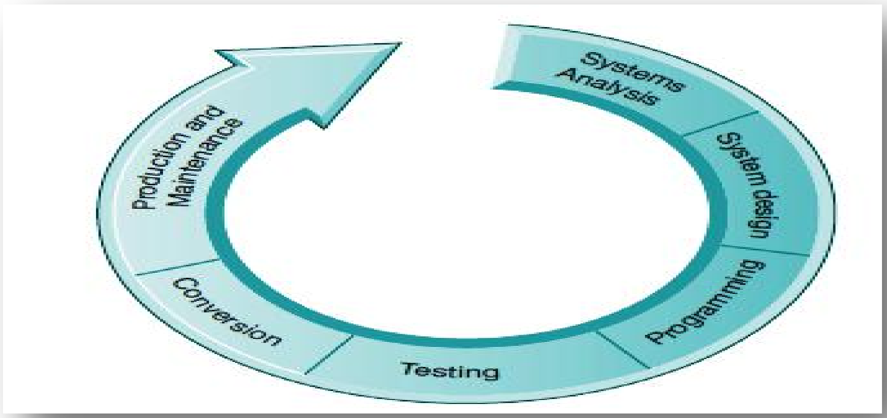
Systems analysis is the analysis of a problem that a firm tries to solve with an information system. It consists of defining the problem, identifying its causes, specifying the solution, and identifying the information requirements that must be met by a system solution.

The systems analyst creates a road map of the existing organization and systems, identifying the primary owners and users of data along with existing hardware and software. The systems analyst then details the problems of existing systems. By examining documents, work papers, and procedures, observing system operations, and interviewing key users of the systems, the analyst can identify the problem areas and objectives a solution would achieve. Often, the solution requires building a new information system or improving an existing one.

Figure 2 The system development process

SMD Ejaz
Assistant Professor
AITS, Rajampeta
WhatsApp: 9133915846

Building a system can be broken down into six core activities



The systems analysis also includes a feasibility study to determine whether that solution is feasible, or achievable, from a financial, technical, and organizational standpoint. The feasibility study determines whether the proposed system is expected to be a good investment, whether the technology needed for the system is available and can be handled by the firm's information systems specialists, and whether the organization can handle the changes introduced by the system.

Normally, the systems analysis process identifies several alternative solutions that the organization can pursue and assess the feasibility of each. A written systems proposal report describes the costs and benefits, and the advantages and disadvantages, of each alternative. It is up to management to determine which mix of costs, benefits, technical features, and organizational impacts represents the most desirable alternative.

Establishing Information Requirements

Perhaps the most challenging task of the systems analyst is to define the specific information requirements that must be met by the chosen system solution. At the most basic level, the information requirements of a new system involve identifying who needs what information, where, when, and how. Requirements analysis carefully defines the objectives of the new or modified system and develops a detailed description of the functions that the new system must perform. Faulty requirements analysis is a leading cause of systems failure and high systems development costs. A system designed around the wrong set of requirements will either have to be discarded because of poor performance or will need to undergo major modifications.

Some problems do not require an information system solution but instead need an adjustment in management, additional training, or refinement of existing organizational procedures. If the problem is information-related, systems analysis still may be required to diagnose the problem and arrive at the proper solution.

2. Systems Design

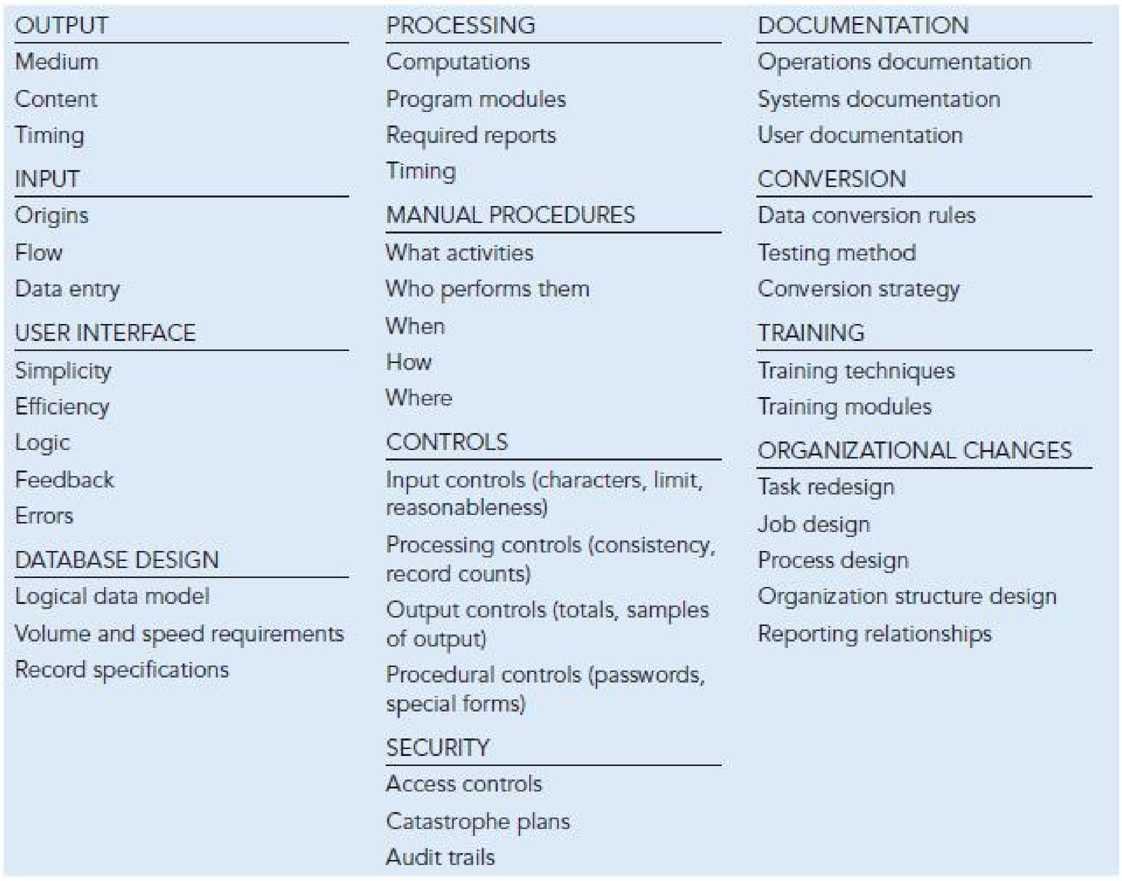
Systems analysis describes what a system should do to meet information requirements, and systems design shows how the system will fulfil this objective. The design of an information system is the overall plan or model for that system. Like the blueprint of a building or house, it consists of all the specifications that give the system its form and structure.

The systems designer details the system specifications that will deliver the functions identified during systems analysis. These specifications should address all of the managerial, organizational, and technological components of the system solution. Table 13.1 lists the types of specifications that would be produced during systems design.

The Role of End Users

User information requirements drive the entire system-building effort. Users must have sufficient control over the design process to ensure that the system reflects their business priorities and information needs, not the biases of the technical staff. Working on design increases users' understanding and acceptance of the system. Insufficient user involvement in the design effort is a major cause of system failure. However, some systems require more user participation in design than others.

Figure 3 System Design Specification



OUTPUT
- Medium
- Content
- Timing

INPUT
- Origins
- Flow
- Data entry

USER INTERFACE
- Simplicity
- Efficiency
- Logic
- Feedback
- Errors

DATABASE DESIGN
- Logical data model
- Volume and speed requirements
- Record specifications

PROCESSING
- Computations
- Program modules
- Required reports
- Timing

MANUAL PROCEDURES
- What activities
- Who performs them
- When
- How
- Where

CONTROLS
- Input controls (characters, limit, reasonableness)
- Processing controls (consistency, record counts)
- Output controls (totals, samples of output)
- Procedural controls (passwords, special forms)

SECURITY
- Access controls
- Catastrophe plans
- Audit trails

DOCUMENTATION
- Operations documentation
- Systems documentation
- User documentation

CONVERSION
- Data conversion rules
- Testing method
- Conversion strategy

TRAINING
- Training techniques
- Training modules

ORGANIZATIONAL CHANGES
- Task redesign
- Job design
- Process design
- Organization structure design
- Reporting relationships

3. Completing the Systems Development Process

The remaining steps in the systems development process translate the solution specifications established during systems analysis and design into a fully operational information system. These concluding steps consist of programming, testing, conversion, production, and maintenance.

3.1. Programming

During the programming stage, system specifications that were prepared during the design stage are translated into software program code. Today, many organizations no longer do their own programming for new systems. Instead, they purchase the software that meets the requirements for a new system from external sources such as software packages from a commercial software vendor, software services from a software service provider, or out-sourcing firms that develop custom application software for their clients.

3.2. Testing

Exhaustive and thorough testing must be conducted to ascertain whether the system produces the right results. Testing answers the question: Will the system produce the desired results under known conditions? Some companies are starting to use cloud computing services for this work.

The amount of time needed to answer this question has been traditionally underrated in systems project planning. Testing is time-consuming: Test data must be carefully prepared, results reviewed, and corrections made in the system. In some instances, parts of the system may have to be redesigned. The risks resulting from glossing over this step are enormous.

Testing an information system can be broken down into three types of activities: unit testing, system testing, and acceptance testing. Unit testing, or program testing, consists of testing each program separately in the system. It is widely believed that the purpose of such testing is to guarantee that programs are error- free, but this goal is realistically impossible. Testing should be viewed instead as a means of locating errors in programs, by focusing on finding all the ways to make a program fail. Once they are pinpointed, problems can be corrected.

System testing tests the functioning of the information system as a whole. It tries to determine whether discrete modules will function together as planned and whether discrepancies exist between the way the system actually works and the way it was conceived. Among the areas examined are performance time, capacity for file storage and handling peak loads, recovery and restart capabilities, and manual procedures.

Acceptance testing provides the final certification that the system is ready to be used in a production setting. Systems tests are evaluated by users and reviewed by management. When all parties are satisfied that the new system meets their standards, the system is formally accepted for installation.

The systems development team works with users to devise a systematic test plan. The test plan includes all of the preparations for the series of tests we have just described.

3.3. Conversion

Conversion is the process of changing from the old system to the new system. Four main conversion strategies can be employed: the parallel strategy, the direct cutover strategy, the pilot study strategy, and the phased approach strategy.

In a parallel strategy, both the old system and its potential replacement are run together for a time until everyone is assured that the new one functions correctly. This is the safest conversion approach because, in the event of errors or processing disruptions, the old system can still be used as a backup. However, this approach is very expensive, and additional staff or resources may be required to run the extra system.

The direct cutover strategy replaces the old system entirely with the new system on an appointed day. It is a very risky approach that can potentially be more costly than running two systems in parallel if serious problems with the new system are found. There is no other system to fall back on. Dislocations, disruptions, and the cost of corrections may be enormous.

The pilot study strategy introduces the new system to only a limited area of the organization, such as a single department or operating unit. When this pilot version is complete and working smoothly, it is installed throughout the rest of the organization, either simultaneously or in stages.

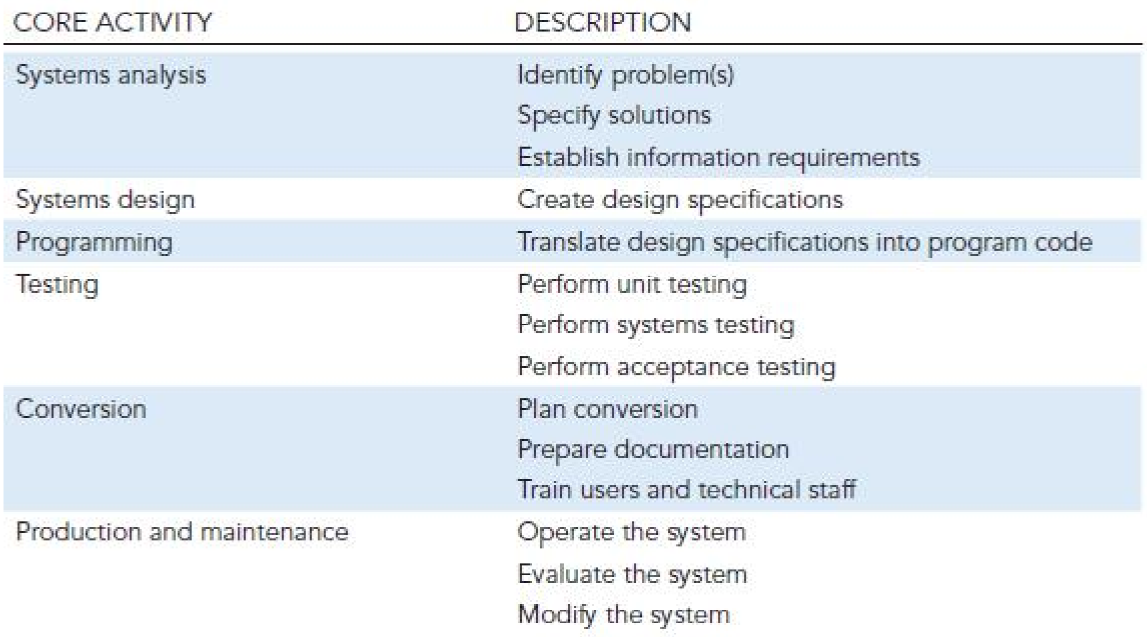
The phased approach strategy introduces the new system in stages, either by functions or by organizational units. If, for example, the system is introduced by function, a new payroll system might begin with hourly workers who are paid weekly, followed six months later by adding salaried employees (who are paid monthly) to the system. If the system is introduced by organizational unit, corporate headquarters might be converted first, followed by outlying operating units four months later.

Moving from an old system to a new one requires that end users be trained to use the new system. Detailed documentation showing how the system works from both a technical and end-user standpoint is finalized during conversion time for use in training and everyday operations. Lack of proper training and documentation contributes to system failure, so this portion of the systems development process is very important.

3.4. Production and Maintenance

After the new system is installed and conversion is complete, the system is said to be in production. During this stage, the system will be reviewed by both users and technical specialists to determine how well it has met its original objectives and to decide whether any revisions or modifications are in order. In some instances, a formal post-implementation audit document is prepared. After the system has been fine-tuned, it must be maintained while it is in production to correct errors, meet requirements, or improve processing efficiency. Changes in hardware, software, documentation, or procedures to a production system to correct errors, meet new requirements, or improve processing efficiency are termed maintenance. Routine maintenance consumes a large percentage of many firms' IT budgets, but could be reduced significantly through more up-to-date systems-building practices and technology. Table 13.2 summarizes the systems development activities.

Figure 4 Core systems development activities.



CORE ACTIVITY	DESCRIPTION
Systems analysis	Identify problem(s) Specify solutions Establish information requirements
Systems design	Create design specifications
Programming	Translate design specifications into program code
Testing	Perform unit testing Perform systems testing Perform acceptance testing
Conversion	Plan conversion Prepare documentation Train users and technical staff
Production and maintenance	Operate the system Evaluate the system Modify the system

SOFTWARE DEVELOPMENT MODELS

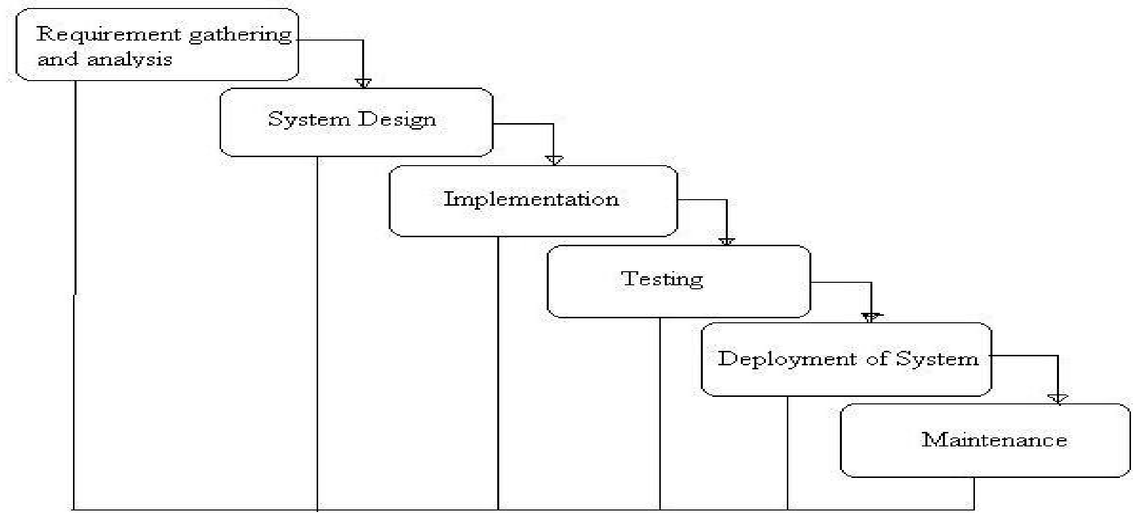
Software development life cycle (**SDLC**) is a series of phases that provide a common understanding of the software building process. How the software will be realized and developed from the business understanding and requirements elicitation phase to convert these business ideas and requirements into functions and features until its usage and operation to achieve the business needs. The good software engineer should have enough knowledge on how to choose the SDLC model based on the project context and the business requirements.

Classical waterfall model

The Waterfall Model was 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 fully before the next phase can begin. This type of software development model is basically used for the for the project which is small and there are no uncertain requirements. At the end of each phase, a review takes place to determine if the project is on the right path and whether or not to continue or discard the project. In this model software testing starts only after the development is complete. In waterfall model phases do not overlap.

Figure 5 Classical Waterfall Model

General Overview of "Waterfall Model"



Advantages of waterfall model:

- This model is simple and easy to understand and use.
- It is easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.
- In this model phases are processed and completed one at a time. Phases do not overlap.
- Waterfall model works well for smaller projects where requirements are very well understood.

Disadvantages of waterfall model:

- Once an application is in the **testing** stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
- 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.

When to use the waterfall model:

- This model is used only when the requirements are very well known, clear and fixed.
- Product definition is stable.
- Technology is understood.
- There are no ambiguous requirements

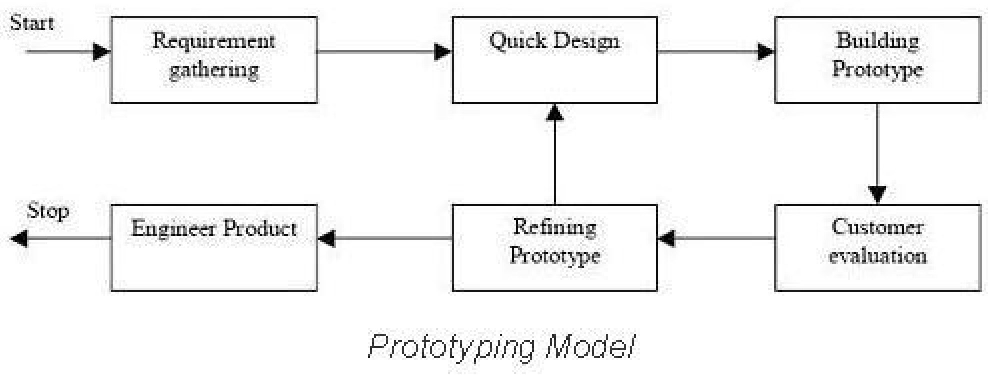
- Ample resources with required expertise are available freely
- The project is short.

Very less customer interaction is involved during the development of the product. Once the product is ready then only it can be demoed to the end users. Once the product is developed and if any failure occurs then the cost of fixing such issues are very high, because we need to update everywhere from document till the logic.

Prototype Model

The basic idea in Prototype model is that instead of freezing the requirements before a design or coding can proceed, a throwaway prototype is built to understand the requirements. This prototype is developed based on the currently known requirements. Prototype model is a software development model. By using this prototype, the client can get an "actual feel" of the system, since the interactions with prototype can enable the client to better understand the requirements of the desired system. Prototyping is an attractive idea for complicated and large systems for which there is no manual process or existing system to help determining the requirements. The prototype are usually not complete systems and many of the details are not built in the prototype. The goal is to provide a system with overall functionality.

Figure 6 Prototype model



Advantages of Prototype model:

- Users are actively involved in the development
- Since in this methodology a working model of the system is provided, the users get a better understanding of the system being developed.
- Errors can be detected much earlier.
- Quicker user feedback is available leading to better solutions.
- Missing functionality can be identified easily
- Confusing or difficult functions can be identified
- Requirements validation, Quick implementation of, incomplete, but functional, application.

Disadvantages of Prototype model:

- Leads to implementing and then repairing way of building systems.
- Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.
- Incomplete application may cause application not to be used as the full system was designed Incomplete or inadequate problem analysis.

When to use Prototype model:

- Prototype model should be used when the desired system needs to have a lot of interaction with the end users.
- Typically, online systems, web interfaces have a very high amount of interaction with end users, are best suited for Prototype model. It might take a while for a system to be built that allows ease of use and needs minimal training for the end user.
- Prototyping ensures that the end users constantly work with the system and provide a feedback which is incorporated in the prototype to result in a useable system. They are excellent for designing good human computer interface systems.

The Spiral Model

The spiral model is similar to the **incremental model**, with more emphasis placed on risk analysis. The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation. A software project repeatedly passes through these phases in iterations (called Spirals in this model). The baseline spiral, starting in the planning phase, requirements are gathered and risk is assessed. Each subsequent spiral build on the baseline spiral.

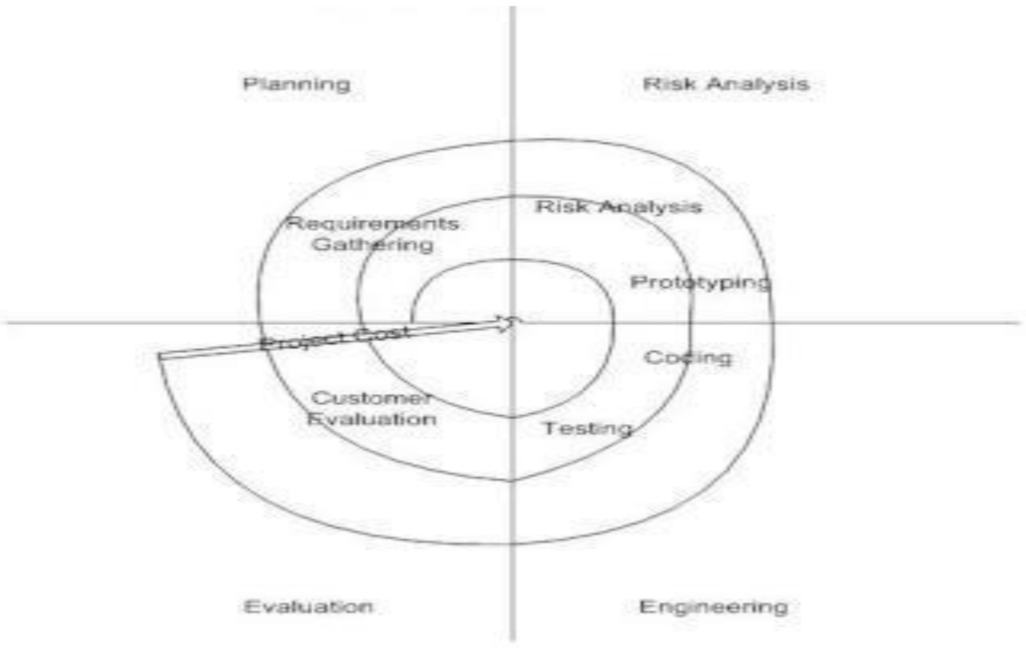
Planning Phase: Requirements are gathered during the planning phase. Requirements like 'BRS' that is 'Business Requirement Specifications' and 'SRS' that is 'System Requirement specifications'.

Risk Analysis: In the **risk analysis phase**, a process is undertaken to identify risk and alternate solutions. A prototype is produced at the end of the risk analysis phase. If any risk is found during the risk analysis then alternate solutions are suggested and implemented.

Engineering Phase: In this phase software is **developed**, along with **testing** at the end of the phase. Hence in this phase the development and testing are done.

E**valuation phase:** This phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.

Figure 7 **Spiral model**



Advantages of Spiral model:

- High amount of risk analysis hence, avoidance of Risk is enhanced.
- Good for large and mission-critical projects.
- Strong approval and documentation control.
- Additional Functionality can be added at a later date.
- Software is produced early in the software life cycle.

Disadvantages of Spiral model:

- Can be a costly model to use.
- Risk analysis requires highly specific expertise.
- Project's success is highly dependent on the risk analysis phase.
- Doesn't work well for smaller projects.

When to use Spiral model:

- When costs and risk evaluation is important
- For medium to high-risk projects
- Long-term project commitment unwise because of potential changes to economic priorities
- Users are unsure of their needs
- Requirements are complex
- New product line
- Significant changes are expected (research and exploration)

MODELING AND DESIGN SYSTEMS:

Modeling and Design Systems

Development Methodologies

The term *software development methodology* is used to describe a framework for the development of information systems. A particular methodology is usually associated with a specific set of tools, models and methods that are used for the analysis, design and implementation of information systems, and each tends to favour a particular lifecycle model. Often, a methodology has its own philosophy of system development that practitioners are encouraged to adopt, as well as its own system of recording and documenting the development process. Many methodologies have emerged in the past few decades in response to the perceived need to manage different types of projects using different tools and methods. Each methodology has its own strengths and weaknesses, and the choice of which approach to use for a given project will depend on the scale of the project, the nature of the business environment, and the type of system being developed. The following sections describe a small number of software development approaches that have evolved over the years.

Structured Systems Analysis Methodology (SSADM)

Structured Systems Analysis Methodology (SSADM) is a highly structured and rigorous approach to the analysis and design of information systems, one of a number of such methodologies that arose as a response to the large number of information system projects that either failed completely or did not adequately fulfil customer expectations.

Early large-scale information systems were often developed using the Cobol programming language together with indexed sequential files to build systems that automated processes such as customer billing and payroll operations. System development at this time was almost a black art, characterised by minimal user involvement. As a consequence, users had little sense of ownership of, or commitment to, the new system that emerged from the process. A further consequence of this lack of user involvement was that system requirements were often poorly understood by developers, and many important requirements did not emerge until late in the development process, leading to costly re-design work having to be undertaken. The situation was not improved by the somewhat arbitrary selection of analysis and design tools, and the absence of effective computer aided software engineering (CASE) tools.

Structured methodologies use a formal process of eliciting system requirements, both to reduce the possibility of the requirements being misunderstood and to ensure that *all* of the requirements are known before the system is developed. They also introduce rigorous techniques to the analysis and design process. SSADM is perhaps the most widely used of these methodologies, and is used in the analysis and design stages of system development. It does not deal with the implementation or testing stages.

SSADM is an open standard, and as such is freely available for use by companies or individuals. It has been used for all government information systems development since 1981, when it was first released, and has also been used by many companies in the expectation that its use will result in robust, high-quality information systems. SSADM is still widely used for large scale information systems projects, and many proprietary CASE tools are available that support SSADM techniques.

The SSADM standard specifies a number of modules and stages that should be undertaken sequentially. It also specifies the deliverables to be produced by each stage, and the techniques to be used to produce those deliverables. The system development life cycle model adopted by SSADM is essentially the waterfall model, in which each stage must be completed and signed off before the next stage can begin.

SSADM techniques

SSADM revolves around the use of three key techniques that derive three different but complementary views of the system being investigated. The three different views of the system are cross referenced and checked against each other to ensure that an accurate and complete overview of the system is obtained. The three techniques used are:

- ***Logical Data Modelling (LDM)*** - this technique is used to identify, model and document the data requirements of the system. The data held by an organisation is concerned with *entities* (things about which information is held, such as customer orders or product details) and the *relationships* (or associations) between those entities. A logical data model consists of a *Logical Data Structure* (LDS) and its associated documentation. The LDS is sometimes referred to as an *Entity Relationship Model* (ERM). *Relational data analysis* (or *normalisation*) is one of the primary techniques used to derive the system's data entities, their *attributes* (or *properties*), and the relationships between them.
- ***Data Flow Modelling*** - this technique is used to identify, model and document the way in which data flows into, out of, and around an information system. It models *processes* (activities that act on the data in some way), *data stores* (the storage areas where data is held), *external entities* (an external entity is either a source of data flowing into the system, or a destination for data flowing out of the system), and *data flows* (the paths taken by the data as it moves between processes and data stores, or between the system and its external entities). A data flow model consists of a set of integrated *Data Flow Diagrams* (DFDs), together with appropriate supporting documentation.
- ***Entity Behaviour Modelling*** - this technique is used to identify, model and document the events that affect each entity, and the sequence in which these events may occur. An entity behaviour model consists of a set of Entity Life History (ELH) diagrams (one for each entity), together with appropriate supporting documentation.

SSADM's structured approach

Activities within the SSADM framework are grouped into five main modules. Each module is sub-divided into one or more stages, each of which contains a set of rigorously defined tasks. SSADM's modules and stages are briefly described in the table below.

Table 1 The SSADM framework

Module	**Stage**	**Description**
Feasibility Study (module 1)	**Feasibility** (Stage 0)	The high-level analysis of a business area to determine whether a proposed system can cost effectively support the business requirements identified. A *Business Activity Model* (BAM) is produced that describes the business activities and events, and the business rules in operation. Problems associated with the current system, and the additional services required, are identified. A high-level data flow diagram is produced that describes the current system in terms of its existing processes, data stores and data flows. The structure of the system data is also investigated, and an initial LDM is created.
Requirements Analysis (module 2)	**Investigation of Current Environment** (stage 1)	The systems requirements are identified and the current business environment is modelled using data flow diagrams and logical data modelling.
	Business System Options (stage 2)	Up to six business system options are presented, of which one will be adopted. Data flow diagrams and logical data models are produced to support each option. The option selected defines the boundary of the system to be developed.
Requirements Specification (module 3)	**Definition of Requirements** (stage 3)	Detailed functional and non-functional requirements (for example, the levels of service required) are identified and the required processing and system data structures are defined. The data flow diagrams and logical data model are refined, and validated against the chosen business system option. The data flow diagrams and logical data model are then validated against the entity life histories, which are also produced during this stage. Parts of the system may be produced as prototypes and demonstrated to the customer to confirm correct interpretation of requirements and obtain agreement on aspects of the user interface.
Logical System Specification	**Technical System Options**	Up to six technical options for the development and implementation of the system are proposed, and one

(module 4)	(stage 4)	is selected.
	Logical Design (stage 5)	In this stage the logical design of the system, including user dialogues and database enquiry and update processing, is undertaken.
Physical Design (module 5)	**Physical Design** (stage 6)	The logical design and the selected technical system option provide the basis for the physical database design and a set of program specifications.

SSADM is well-suited to large and complex projects where the requirements are unlikely to change significantly during the project's life cycle. Its documentation-oriented approach and relatively rigid structure makes it inappropriate for smaller projects, or those for which the requirements are uncertain, or are likely to change because of a volatile business environment.

Object-Oriented Methodologies

Object-oriented Life Cycle Model in Software Engineering

The object-oriented life cycle model considers 'objects' as the basis of the software engineering process. The development team starts by observing and analyzing the system they intend to develop before defining the requirements. Once the process is over, they focus on identifying the objects of the system. Now, an object could be anything; it can have a physical existence like a customer, car, etc. An object also constitutes intangible elements like a process or a project.

Advantages of Object-Oriented Life Cycle Model

Apart from enhancing the system performance, object-oriented programming offers some advantages such as:

- Since it is data-focused and easy to work with problem domains.
- It uses encapsulation and data hiding process that allows a developer to build tamper-proof systems.
- It enables software modularity, making it easier to manage and maintain complex software.
- It allows developers to create new modules using existing models, saving time and development cost of organizations.

The primary objectives of the Object-Oriented Model

- Object-oriented Analysis
- Object-oriented Design

- Object-oriented Implementation

Object-Oriented Analysis (OOA)

The object-oriented analysis consists of the process where a development team evaluates the system and organizes the requirements as objects. Contrary to traditional structural analysis, the OOA heavily depends on advanced data like **Use Cases** and **Object Models**.

Use case

Use Cases are written descriptions about how users will behave when they enter your website or application. It comprises the goals of each user from the point of their entry to exit.

Object Model

An object model allows the development team to create an architectural software or system model before implementing or programming. It helps in defining a software/system in objects and classes. It informs the developers about

- Interaction between different models
- Inheritance
- Encapsulation
- Other types of object-oriented interfaces

The OOA starts with analysing the problem domain and produce a conceptual model by thoroughly evaluating the information in the given area. There is an abundance of data available from various sources like:

- Formal document
- Requirement statements
- Primary data collected through stakeholders

Once the analysis is complete, the development team prepares a conceptual model describing the system's functionalities and requirements.

Object-oriented Design

It is the next development stage of the object-oriented life cycle model where the analysts design the desired system's overall architecture. The system is divided into a set of interacting subsystems. The analyst considers the specifications from the system analysis. It all about evaluating what the end-users expect from the new system.

As per the object-oriented design, the system is considered a collection of objects, with each object handling a specific state data. For example, in banking software, each account may feature some exclusive objects with separate data and functions.

The philosophy behind an object-oriented design is to create a set of interacting objects as seen in the real world. Instead of process-based structural programming, developers create objects through data structures.

Each programming language comes with several data types, with each type comprising various variables. Similarly, objects also feature certain predefined data types.

Useful definitions for Object-oriented design

Class

A class refers to a collection of similar objects. It is created as a blueprint to define variables and methods that share certain similarities with objects.

As stated above, an object-oriented design bears resemblances with the real world. Let's say you have purchased a smartphone. Now, your smartphone is just one of the several 'smartphones' available in the world. We can consider 'smartphones' as a class of objects, and your smartphone object is an instance of a class of objects. Smartphones feature many states (operating System, RAM, and motherboard) and behaviour (play music, call, messaging) in common. However, the state of each smartphone is independent and can be different from other smartphones.

While manufacturing smartphones, manufacturers can use the exact blueprint to build many smartphones as they share common characteristics. This allows manufacturers to create new blueprints more efficiently.

Likewise, in object-oriented programming, developers can use many similar objects to create blueprints. This is called a class.

Abstraction

Abstraction is the essence used by developers to build classes. Developers observe a set of similar objects and characteristics of importance to define classes. Abstractions are divided into two parts- global abstractions and local abstractions.

Global abstractions are static, providing one interface to users for a limited time. Meanwhile, Local abstractions are responsible for providing a view based on user/developer's requirements.

The abstraction of objects varies as per the application. For example, while defining a smartphone class of users, developers might set attributes like color, features, price, etc. However, for manufacturing firms, developers may set attributes containing such as the manufacturing costs per smartphone, quality control, turnaround, etc.

Inheritance

The concept of inheritance in object-oriented design defines the process of reusing 'objects.' Developers can define a new class type using a similar existing class.

For example, hatchbacks, sedans, SUVs, and 4wds are all a form of motor vehicles. So, as per object-oriented programming, we can refer to hatchbacks, sedans, SUVs, etc., as subclasses of the motor vehicle class. Similarly, the motor vehicle class is the superclass of the subclasses.

Now, each subclass (sedans, hatchback, SUVs) inherits some states (speed, cadence, etc.), methods, and behaviors (braking, changing gear) of the superclass. However, the state and behaviors of subclasses are not limited to what has been provided to them by the superclass. Developers can add variables and methods to subclasses as required.

Object-oriented Implementation

In this phase, developers translate the class objects and the interrelationships of classes and code them using a programming language. This is the phase to create databases and establish functionalities for the system.

The object-oriented methodology focuses on identifying objects in the system. Developers closely observe each object to identify characteristics and behavioral patterns. The developers ensure that the object recognizes and responds perfectly to an event.

Let's consider a smartphone screen as an object and the touch on a specific icon as an event. Now, when the user touches an icon, the screen opens up an application. This means the smartphone screen (object) responds to the event (touch) by opening an application.

The object-oriented implementation methodology supports three basic models

Object Model − It describes the objects and their interrelationships. This model observes objects as static and discards their dynamicity.

Dynamic Model − This model focuses on the changes in states of various objects related to specific events.

Functional Model − This describes the changes in the flow of data throughout the system.

The object model describes the essential elements of a system. When all the models are combined, it represents the complete function of the system.

RAPID APPLICATION DEVELOPMENT **(RAD)**

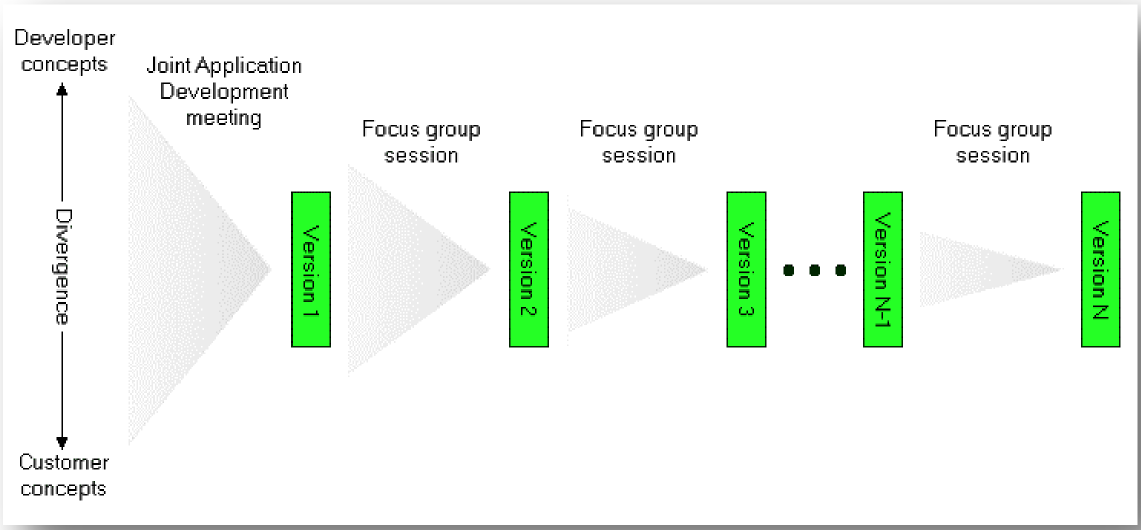
Rapid application development (RAD) is an iterative and incremental software development process that is designed to produce software as quickly as possible. The term tends to refer to a range of techniques geared to the rapid development of applications, such as the use of various application development frameworks. RAD was an early response to more structured and formal approaches like SSADM which were not felt to be appropriate for projects undertaken within a highly volatile and evolving business environment.

The philosophy behind RAD is that there is an acceptable trade-off between the speed of development and the overall functionality or performance of the software delivered. Put another way, RAD can deliver a working solution that provides 80% of the required functionality in 20% of the time required by more traditional approaches. Two major benefits of this approach are that the customer gets to see results very quickly, and that a production version of the software will be available within a relatively short time frame, greatly reducing the likelihood that the customer's business environment will have undergone significant change by the time the new system is delivered. The down side is that some of the desirable (but non-essential) features of the software may be sacrificed in order to speed development, and the performance of the resulting system, while acceptable, may not be optimal. System acceptance is based upon the system achieving the agreed minimum functionality and usability.

A RAD team is usually small (maybe six or so people including developers and users), and developers are usually required both experienced and multi-skilled, since they will be combining the role of analyst, designer and programmer. The project begins with an initial *Joint Application Development* (JAD) meeting during which developers and customer representatives determine the initial requirements of the system and agree a time frame in which a prototype system will be ready. The developers design, build and test a prototype system that reflects these initial requirements. The customer then evaluates the prototype system to determine how far it meets their requirements, and what functionality or features need to be improved or added.

A focus group meeting then takes place, during which the customer reports back to the development team. The requirements specification is revised to incorporate new features and improvements, and the time frame for the next iteration is agreed. Features that are deemed to be of secondary importance may, by negotiation, be dropped from the new requirements specification if they will negatively impact on the time frame for the new prototype. The cycle of iterations and focus group meetings continues until a final prototype is accepted by the customer as a production version of the new system.

Figure 8 The Rapid Application Development life cycle



The "time box" in which an iteration occurs is short (usually from a few days up to about three weeks). Documentation of requirements and design documentation is usually restricted to notes taken from meetings, rather than the formal documentation associated with more structured methodologies, and will consist of the minimum documentation required to facilitate the development and maintenance of the system. The entire life cycle is relatively short (usually a few months), and should result in a steady convergence between the customer's concept of the new system and that of the development team, resulting in a workable business solution that is fit for its intended purpose.

One of the benefits claimed for RAD was that, because customers often had only a vague idea of what they wanted, the availability of a working prototype would help to crystalise their thoughts in this respect and enable them to evolve a more definitive set of requirements. Whereas some system development methodologies attempted to determine the complete set of requirements in advance in an attempt to eliminate future changes to the scope of the project, RAD was able to incorporate change as part of an evolutionary development process.

RAD leveraged the benefits of a number of software development tools in order to speed up the development process, including a range of *computer aided software engineering* (CASE) tools. Code re-use, the use of object-oriented programming languages, and the utilisation of third-party software components were all embraced by RAD developers. Fourth generation visual programming languages, the forerunners of today's *integrated development environments* (IDEs) were used to create the *graphical user interface* (GUI), while code production was further speeded through the use of an appropriate *application programming interface* (API) that provided much of the base code for the application.

RAD tended to be used successfully for projects that did not have a high degree of criticality, and where the trade-off between a short time frame for development on the one hand, and quality and performance on the other, was acceptable. It was not suitable for systems where optimal performance was required, or that must interoperate with existing systems. The flexibility of RAD lay in the ability to produce results quickly and adapt specifications to meet an evolving set of customer requirements. From the customer's point of view, seeing a working prototype early on in the proceedings helped them to focus on what they did or didn't want from the system, and the continuing dialogue with the development team meant that developers had a good understanding of the customer's requirements.

The speed of development and the relatively small size of development teams tended to result in reduced development costs, although the absence of classic project milestones sometimes made it difficult to accurately measure progress. Today, some of the principles of RAD have been adopted by practitioners of agile development methods, themselves a response to an increasingly volatile business environment.

Agile software development

Agile software development refers to a group of loosely related software development methodologies that are based on similar principles. Notable examples include the *Unified Software Development Process* (USDP) and *Extreme Programming* (XP). Agile methodologies are characterised by short life-cycle iterations (typically measured in weeks), with minimal planning and documentation. The goal is to deliver working software to the customer at the end of each cycle. Each iteration involves a number of phases including planning, requirements analysis, implementation, and testing. This incremental, iterative approach helps to reduce overall risk, while enabling the output of the project to be adapted to meet changing requirements or circumstances. Documentation is generally limited to what the customer requires. Agile methodologies have evolved as an alternative to more traditional, process-driven methodologies.

The emphasis is on frequent (usually daily) face-to-face communication within the project team, and between the project team and the customer. Written documentation is of secondary importance, and meetings are usually formal but brief. Project teams are typically small (usually less than a dozen people) to facilitate communication and collaboration. Where agile methods are applied to larger projects, the different parts of a project may be allocated to several small teams of developers.

Each iteration of the project life cycle results in the production of working software, which is then evaluated by the customer before the next iteration begins. The production of working software, rather than the completion of extensive project documentation, is seen as the primary measure of progress. The software produced at the end of an iteration has been fully developed and tested, but embodies only a subset of the functionality planned for the project as a whole. The aim is to deliver functionality incrementally as the project progresses. Further functionality will be added, and existing features will be refined, throughout the life of the project.

Agile methods are favoured over more structured methodologies for projects where requirements are not initially well defined, or are likely to change over the lifetime of the project. They work well where the project team is small, and comprised of experienced developers. Both the project management techniques and the development tools used are selected on a project-by-project basis, allowing the overall process to be tailored to the needs of a particular project. The short duration of iterations and the absence of a rigid set of requirements or design documentation allow developers to respond quickly to changing requirements and circumstances. The emphasis on constant interaction between customers and developers provides continual feedback that helps to keep the project on track.

Agile methods are not so well suited to large-scale projects where the requirements are well defined, where the business environment is relatively non-volatile, or where the predominant organisational culture is intolerant of a lack of structure or documentation. The emphasis on frequent face-to-face communication as an essential element of the development process means that agile methods do not lend themselves easily to projects that are distributed over a wide geographical area, or that require large teams of developers. Critics of agile methods have also pointed out the difficulties that may arise in terms of negotiating a contract or determining the cost of a project where the scope of the project is not initially well-defined, and requirements are unclear.

Unified Software Development Process (USDP)

The *Unified Software Development Process* (USDP) is an iterative and incremental software development process framework which, it is claimed, can be adopted for the development of both small- and large-scale information systems. The development cycle is divided into four main phases:

- ***Inception*** - this is usually a fairly short phase that is primarily used to establish the scope and objectives of the project. It lays down both the overall aims and the specific functional objectives, such as being able to log into and out of the system. These specific functional objectives are referred to as use cases. The phase will also identify one or more candidate architectures for the system, identify risks, and determine a preliminary project schedule and cost estimate. The end of the inception phase is marked by the Objective milestone.

- ***Elaboration*** - in this phase, most of the system requirements, the known risks, and the system architecture are established. *Use case diagrams*, *conceptual diagrams* and *package diagrams* are created. A partial implementation of the system is produced in a series of short, time-boxed iterations that includes the core architectural components, and establishes an *executable architecture baseline*. The other deliverable from this phase is a blueprint for the next phase (the *construction* phase) that includes estimates of the cost and the time required for completion. The end of the elaboration phase is marked by the *Architecture milestone*.

- ***Construction*** - the remaining parts of the system are built on the foundations laid down in the previous phase in a series of short, time-boxed iterations, each resulting in a software release. A number of common *Unified Modelling Language* (UML) diagrams are used during this phase for the purpose of specifying visualising, constructing and documenting the system. The end of the construction phase is marked by the *Initial Operational Capability milestone*.

- ***Transition*** - the system is deployed to its operational environment and user community. Feedback from an initial release may lead to further iterations within the transition phase that incorporate further refinements to the system. This phase may also include activities such as data conversion and user training. The end of the transition phase is marked by the *Product Release milestone*.

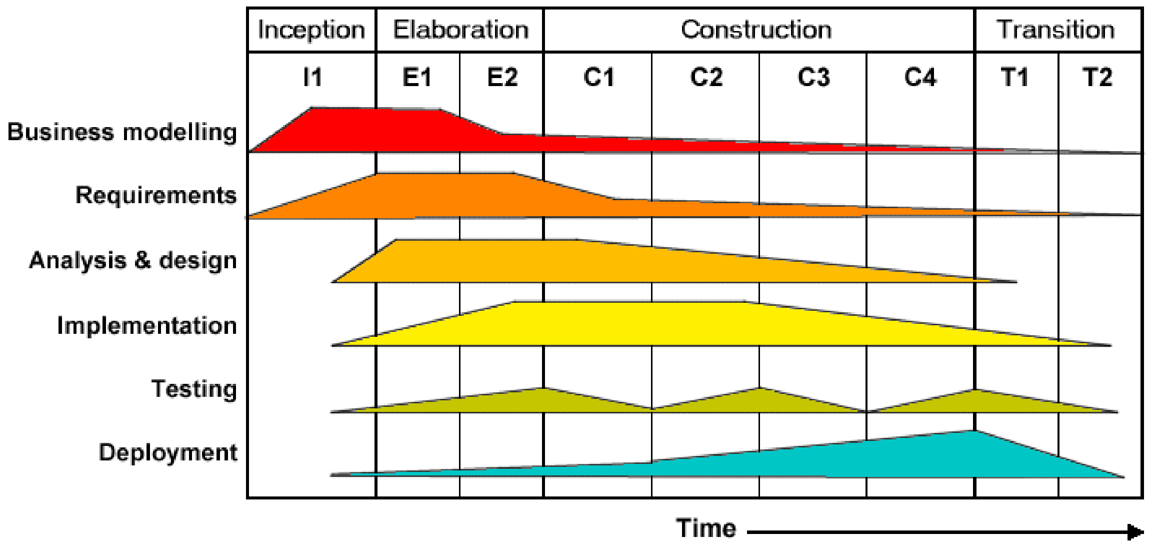
The *system architecture* describes the various functional subsystems that make up the system, such as those responsible for handling input and output, data communications, and information reporting, and the interactions between them and the rest of the system. A *risk* is any obstacle to success (e.g. insufficient or inexperienced personnel, lack of funding, or severe time restrictions. Each *iteration* results in a single release of the system, although there can be one or more intermediate builds within a single iteration. The feedback from each release is used to shape future iterations.

The unified process defines six core process disciplines:

- Business modelling
- Requirements
- Analysis & Design
- Implementation
- Testing
- Deployment

Most iterations will include some work in most of the process disciplines. The relative emphasis placed on each activity, and the effort it requires, will change over the course of the project. This is illustrated by the following diagram.

SMD Ejaz
Assistant Professor
AITS, Rajampeta
WhatsApp: 9133915846



The Unified System Development Process lifecycle

Extreme Programming

Extreme Programming (or XP) is an agile software development methodology that takes traditional software engineering practices to "extreme" levels to achieve a development process that is more responsive to customer needs than traditional methods, while creating good quality software. Changing requirements are seen as an inescapable feature of software development projects in an increasingly unpredictable business environment. XP Practitioners believe that a software development methodology that embodies the capacity to adapt to changing requirements is a more realistic approach than trying to define all of the requirements at the start of a project. Rapidly-changing requirements demand shorter development life-cycles, and are incompatible with traditional methods of software development.

Individual developers are assigned specific tasks, and are responsible for their completion. No code is written until unit tests have been designed for individual code components and subsystems. The customer is responsible for defining appropriate acceptance tests that are subsequently used to validate the software produced during an iteration. At the end of an iteration, the development team delivers a working system to the customer. The system may not be complete, but all functionality implemented works. A further meeting is scheduled to plan the next iteration, and the cycle begins again.

The Extreme Programming methodology encompasses a set of values, principles and practices designed to facilitate the rapid development of high-quality software that satisfies customer requirements.

The twelve core practices of XP are described below.

- ***The planning game*** - the development team collaborates with the customer to produce working software as quickly as possible. The customer produces a list of the required system features, each described in a *user story*, which gives the feature a name and outlines its functionality. User stories are typically written on index cards. The development team estimates the effort required to code each story, and how much effort the team can produce in a single iteration. The customer decides which user stories to implement, and in what order, as well as how often to produce a production release of the system.

- ***Small releases*** - the first iteration produces a working software release that embodies the functionality identified by the customer as being the most essential. Subsequent iterations add additional features as requested by the customer. Iterations are of fixed length (typically from two to three weeks).
- ***System metaphor*** - each project has an organising metaphor, which provides an easy to remember naming convention.
- ***Simple design*** - the simplest possible design is used that will satisfy customer requirements. Because of the high probability of changes to requirements, only currently known requirements will be considered.
- ***Test driven development*** - unit tests are written by developers to test functionality as they write code. Acceptance tests are specified by the customer to test that the overall system is functioning as expected. All tests must be successfully completed before software is released.
- ***Refactoring*** - any duplicate or unnecessary code generated in a coding session is eliminated, fostering the utilisation of re-usable code.
- ***Pair programming*** - all code is written by two programmers working together on one computer, with the aim of producing high quality code. One person will focus on coding while the other will focus on strategic issues.
- ***Collective code ownership*** - no one programmer "owns" a code module. Any developer can be required to work on any part of the code at any time.
- ***Continuous integration*** - all changes are integrated into the system daily. Integration testing must be successful carried out before further integration occurs.
- ***Sustainable pace*** - developers are expected to be able to go home on time. Excessive overtime is taken as a sign that something is wrong with the development process.
- ***Whole team*** - the development team has continuous access to a customer representative.
- ***Coding standards*** - all programmers are expected to write code to the same standards. Ideally, it should not be possible to tell which member of the development team has written a code module simply by examining the code.

Extreme Programming may be appropriate for relatively small-scale projects where the requirements change rapidly, or where some initial development is needed before previously unforeseen implementation problems can be determined. It may not work so well for larger projects, or projects where the requirements are unlikely to change.

Computer-Aided-Software-Engineering (CASE)

CASE stands for **C**omputer **A**ided **S**oftware **E**ngineering. It means, development and maintenance of software projects with help of various automated software tools.

CASE Tools

CASE tools are set of software application programs, which are used to automate SDLC activities. CASE tools are used by software project managers, analysts and engineers to develop software system.

There are number of CASE tools available to simplify various stages of Software Development Life Cycle such as Analysis tools, Design tools, Project management tools, Database Management tools, Documentation tools are to name a few.

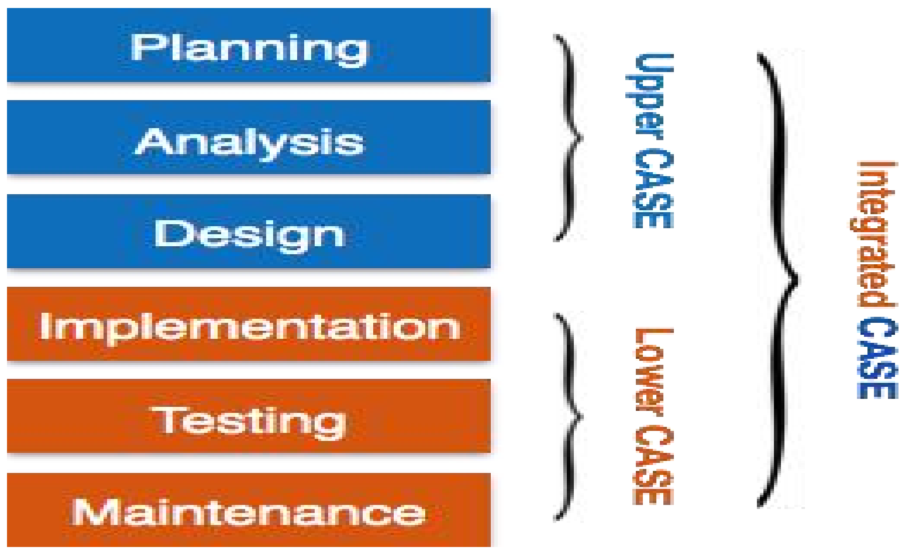
Use of CASE tools accelerates the development of project to produce desired result and helps to uncover flaws before moving ahead with next stage in software development.

Components of CASE Tools

CASE tools can be broadly divided into the following parts based on their use at a particular SDLC stage:

- **Central Repository** - CASE tools require a central repository, which can serve as a source of common, integrated and consistent information. Central repository is a central place of storage where product specifications, requirement documents, related reports and diagrams, other useful information regarding management is stored. Central repository also serves as data dictionary.

Figure 9 Components of CASE Tools



- **Upper Case Tools** - Upper CASE tools are used in planning, analysis and design stages of SDLC.
- **Lower Case Tools** - Lower CASE tools are used in implementation, testing and maintenance.
- **Integrated Case Tools** - Integrated CASE tools are helpful in all the stages of SDLC, from Requirement gathering to Testing and documentation.

CASE tools can be grouped together if they have similar functionality, process activities and capability of getting integrated with other tools.

Scope of Case Tools

The scope of CASE tools goes throughout the SDLC.

Case Tools Types

Now we briefly go through various CASE tools

Diagram tools

These tools are used to represent system components, data and control flow among various software components and system structure in a graphical form. For example, Flow Chart Maker tool for creating state-of-the-art flowcharts.

Process Modeling Tools

Process modeling is method to create software process model, which is used to develop the software. Process modeling tools help the managers to choose a process model or modify it as per the requirement of software product. For example, EPF Composer

Project Management Tools

SMD Ejaz
Assistant Professor
AITS, Rajampeta
WhatsApp: 9133915846

These tools are used for project planning, cost and effort estimation, project scheduling and resource planning. Managers have to strictly comply project execution with every mentioned step in software project management. Project management tools help in storing and sharing project information in real-time throughout the organization. For example, Creative Pro Office, Trac Project, Basecamp.

Documentation Tools

Documentation in a software project starts prior to the software process, goes throughout all phases of SDLC and after the completion of the project.

Documentation tools generate documents for technical users and end users. Technical users are mostly in-house professionals of the development team who refer to system manual, reference manual, training manual, installation manuals etc. The end user documents describe the functioning and how-to of the system such as user manual. For example, Doxygen, DrExplain, Adobe RoboHelp for documentation.

Analysis Tools

These tools help to gather requirements, automatically check for any inconsistency, inaccuracy in the diagrams, data redundancies or erroneous omissions. For example, Accept 360, Accompa, Case Complete for requirement analysis, Visible Analyst for total analysis.

Design Tools

These tools help software designers to design the block structure of the software, which may further be broken down in smaller modules using refinement techniques. These tools provide detailing of each module and interconnections among modules. For example, Animated Software Design

Configuration Management Tools

An instance of software is released under one version. Configuration Management tools deal with –

- Version and revision management
- Baseline configuration management
- Change control management

CASE tools help in this by automatic tracking, version management and release management. For example, Fossil, Git, Accu REV.

Change Control Tools

These tools are considered as a part of configuration management tools. They deal with changes made to the software after its baseline is fixed or when the software is first released. CASE tools automate change tracking, file management, code management and more. It also helps in enforcing change policy of the organization.

Programming Tools

These tools consist of programming environments like IDE (Integrated Development Environment), in-built modules library and simulation tools. These tools provide comprehensive aid in building software product and include features for simulation and testing. For example, Cscope to search code in C, Eclipse.

Prototyping Tools

Software prototype is simulated version of the intended software product. Prototype provides initial look and feel of the product and simulates few aspects of actual product.

Prototyping CASE tools essentially come with graphical libraries. They can create hardware independent user interfaces and design. These tools help us to build rapid prototypes based on existing information. In addition, they provide simulation of software prototype. For example, Serena prototype composer, Mockup Builder.

Web Development Tools

These tools assist in designing web pages with all allied elements like forms, text, script, graphic and so on. Web tools also provide live preview of what is being developed and how will it look after completion. For example, Fontello, Adobe Edge Inspect, Foundation 3, Brackets.

Quality Assurance Tools

Quality assurance in a software organization is monitoring the engineering process and methods adopted to develop the software product in order to ensure conformance of quality as per organization standards. QA tools consist of configuration and change control tools and software testing tools. For example, SoapTest, AppsWatch, JMeter.

Maintenance Tools

Software maintenance includes modifications in the software product after it is delivered. Automatic logging and error reporting techniques, automatic error ticket generation and root cause Analysis are few CASE tools, which help software organization in maintenance phase of SDLC. For example, Bugzilla for defect tracking, HP Quality Center.

ALTERNATIVE SYSTEMS-BUILDING APPROACHES

Systems differ in terms of their size and technological complexity and in terms of the organizational problems they are meant to solve. A number of systems-building approaches have been developed to deal with these differences. This section describes these alternative methods:

1. **The traditional systems life cycle,**
2. **prototyping,**
3. **end-user development,**
4. **application software packages**

5. **outsourcing.**

Traditional Systems Life Cycle

The systems life cycle is the oldest method for building information systems. The life cycle methodology is a phased approach to building a system, dividing systems development into formal stages. Systems development specialists have different opinions on how to partition the systems-building stages, but they roughly correspond to the stages of systems development that we have just described.

The systems life cycle methodology maintains a very formal division of labor between end users and information systems specialists. Technical specialists, such as system analysts and programmers, are responsible for much of the systems analysis, design, and implementation work; end users are limited to providing information requirements and reviewing the technical staff 's work. The life cycle also emphasizes formal specifications and paperwork, so many documents are generated during the course of a systems project.

The systems life cycle is still used for building large complex systems that require a rigorous and formal requirements analysis, predefined specifications, and tight controls over the systems-building process. However, the systems life cycle approach can be costly, time consuming, and inflexible. Although systems builders can go back and forth among stages in the life cycle, the systems life cycle is predominantly a "waterfall" approach in which tasks in one stage are completed before work for the next stage begins. Activities can be repeated, but volumes of new documents must be generated and steps retraced if requirements and specifications need to be revised. This encourages freezing of specifications relatively early in the development process. The life cycle approach is also not suitable for many small desktop systems, which tend to be less structured and more individualized.

Prototyping

Proto-typing consists of building an experimental system rapidly and inexpensively for end users to evaluate. By interacting with the prototype, users can get a better idea of their information requirements. The prototype endorsed by the users can be used as a template to create the final system.

The prototype is a working version of an information system or part of the system, but is meant to be only a preliminary model. Once operational, the prototype will be further refined until it conforms precisely to users' requirements. Once the design has been finalized, the prototype can be converted to a polished production system.

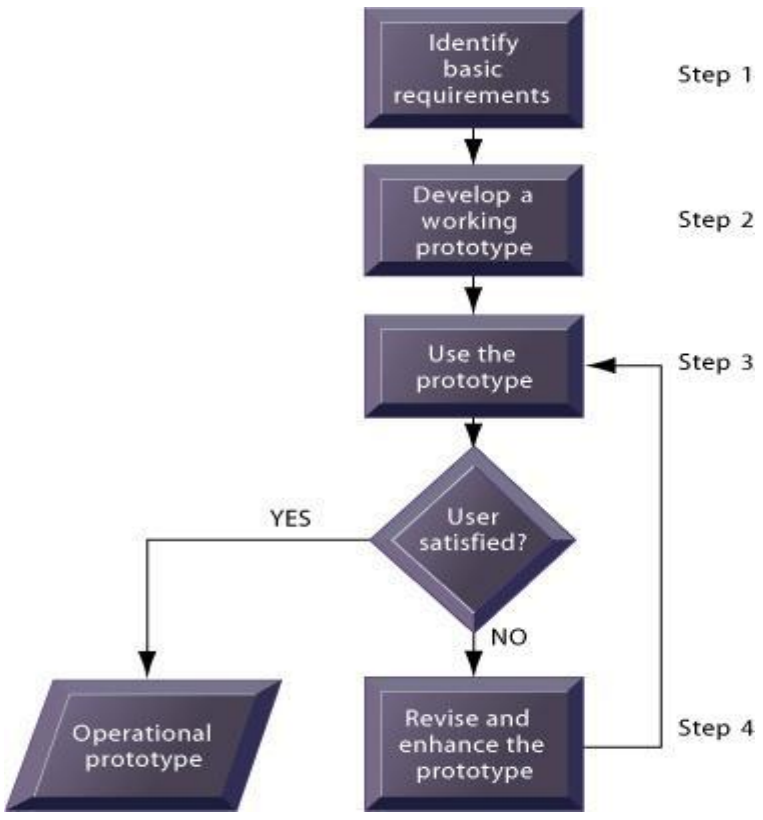
The process of building a preliminary design, trying it out, refining it, and trying again has been called an iterative process of systems development because the steps required to build a system can be repeated over and over again. Prototyping is more explicitly iterative than the conventional life cycle, and it actively promotes system design changes. It has been said that prototyping replaces unplanned rework with planned iteration, with each version more accurately reflecting users' requirements.

STEPS IN PROTOTYPING

Figure 14-11 shows a four-step model of the prototyping process, which consists of the following:

1. **Step 1:** Identify the user's basic requirements. The system designer (usually an information systems specialist) works with the user only long enough to capture the user's basic information needs.
2. **Step 2:** Develop an initial prototype. The system designer creates a working prototype quickly, using tools for rapidly generating software.
3. **Step 3:** Use the prototype. The user is encouraged to work with the system to determine how well the prototype meets his or her needs and to make suggestions for improving the prototype.
4. **Step 4:** Revise and enhance the prototype. The system builder notes all changes the user requests and refines the prototype accordingly. After the prototype has been revised, the cycle returns to step 3. Steps 3 and 4 are repeated until the user is satisfied.

Figure 10 STEPS IN PROTOTYPING


Identify basic requirements
Step 1
Develop a working prototype
Step 2
Use the prototype
Step 3
YES
User satisfied?
NO
Operational prototype
Revise and enhance the prototype
Step 4

The process of developing a prototype can be broken down into four steps. Because a prototype can be developed quickly and inexpensively, systems builders can go through several iterations, repeating steps 3 and 4, to refine and enhance the prototype before arriving at the final operational one.

When no more iterations are required, the approved prototype then becomes an operational prototype that furnishes the final specifications for the application. Sometimes the prototype is adopted as the production version of the system.

ADVANTAGES AND DISADVANTAGES OF PROTOTYPING

Prototyping is most useful when there is some uncertainty about requirements or design solutions. Prototyping is especially useful in designing an information system's end-user interface (the part of the system with which end users interact, such as online display and data-entry screens, reports, or Web pages). Because prototyping encourages intense end-user involvement throughout the systems development life cycle, it is more likely to produce systems that fulfil user requirements.

However, rapid prototyping can gloss over essential steps in systems development. If the completed prototype works reasonably well, management may not see the need for reprogramming, redesign, or full documentation and testing to build a polished production system. Some of these hastily constructed systems may not easily accommodate large quantities of data or a large number of users in a production environment.

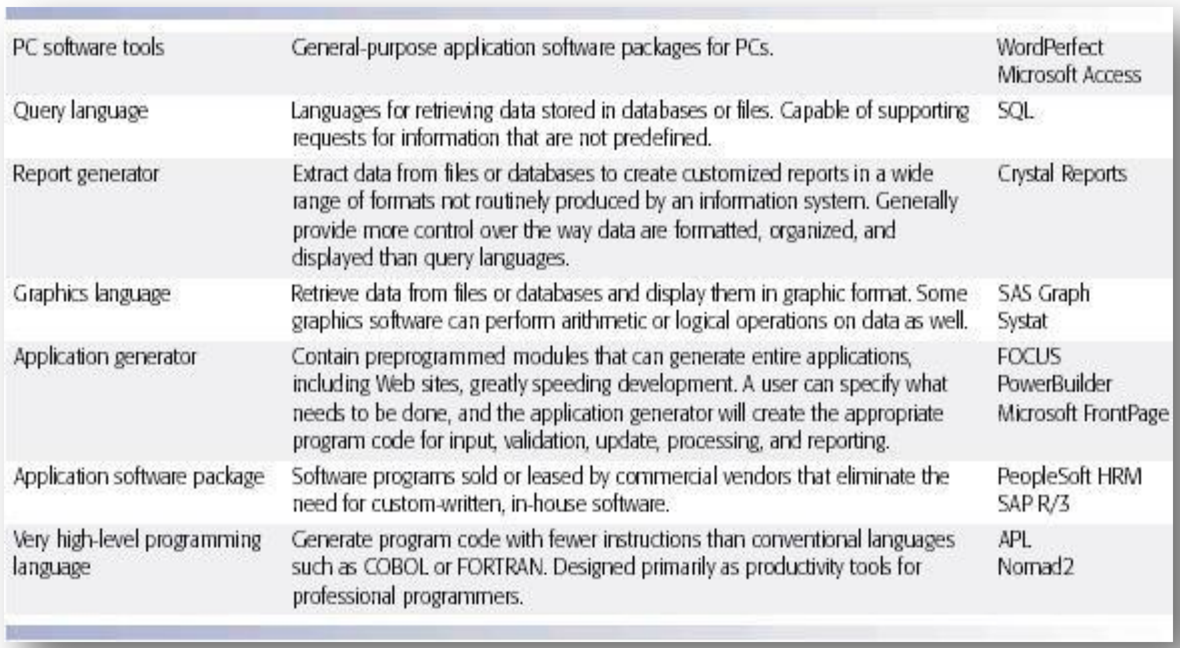
End-User Development

Some types of information systems can be developed by end users with little or no formal assistance from technical specialists. This phenomenon is called end-user development. A series of software tools categorized as fourth-generation languages makes this possible. Fourth-generation languages are software tools that enable end users to create reports or develop software applications with minimal or no technical assistance. Some of these fourth-generation tools also enhance professional programmers' productivity.

Fourth-generation languages tend to be nonprocedural, or less procedural, than conventional programming languages. Procedural languages require specification of the sequence of steps, or procedures, that tell the computer what to do and how to do it. Nonprocedural languages need only specify what has to be accomplished rather than provide details about how to carry out the task.

Table 4 shows that there are seven categories of fourth-generation languages: PC software tools, query languages, report generators, graphics languages, application generators, application software packages, and very high-level programming languages. The table shows the tools ordered in terms of ease of use by nonprogramming end users. End users are most likely to work with PC software tools and query languages. Query languages are software tools that provide immediate online answers to requests for information that are not predefined, such as “Who are the highest-performing sales representatives?” Query languages are often tied to data management software and to database management systems

Figure 11 Categories of Fourth-Generation Languages



PC software tools	General-purpose application software packages for PCs.	WordPerfect Microsoft Access
Query language	Languages for retrieving data stored in databases or files. Capable of supporting requests for information that are not predefined.	SQL
Report generator	Extract data from files or databases to create customized reports in a wide range of formats not routinely produced by an information system. Generally provide more control over the way data are formatted, organized, and displayed than query languages.	Crystal Reports
Graphics language	Retrieve data from files or databases and display them in graphic format. Some graphics software can perform arithmetic or logical operations on data as well.	SAS Graph Systat
Application generator	Contain preprogrammed modules that can generate entire applications, including Web sites, greatly speeding development. A user can specify what needs to be done, and the application generator will create the appropriate program code for input, validation, update, processing, and reporting.	FOCUS PowerBuilder Microsoft FrontPage
Application software package	Software programs sold or leased by commercial vendors that eliminate the need for custom-written, in-house software.	PeopleSoft HRM SAP R/3
Very high-level programming language	Generate program code with fewer instructions than conventional languages such as COBOL or FORTRAN. Designed primarily as productivity tools for professional programmers.	APL Nomad2

On the whole, end-user-developed systems can be completed more rapidly than those developed through the conventional systems life cycle. Allowing users to specify their own business needs improves requirements gathering and often leads to a higher level of user involvement and satisfaction with the system. However, fourth-generation tools still cannot replace conventional tools for some business applications because they cannot easily handle the processing of large numbers of transactions or applications with extensive procedural logic and updating requirements.

End-user computing also poses organizational risks because it occurs outside of traditional mechanisms for information systems management and control. When systems are created rapidly, without a formal development methodology, testing and documentation may be inadequate. Control over data can be lost in systems outside the traditional information systems department. To help organizations maximize the benefits of end-user applications development, management should control the development of end-user applications by requiring cost justification of end-user information system projects and by establishing hardware, software, and quality standards for user-developed applications.

Application Software Packages and Outsourcing

The software for most systems today is not developed in-house but is purchased from external sources. Firms can rent the software from an application service provider, they can purchase a software package from a commercial vendor, or they can have a custom application developed by an outside outsourcing firm.

The Window on Technology illustrates a company that is using multiple approaches to obtain better systems. Elie Tahari Limited is using software packages for business transaction systems and for end-user computing tools. The company outsourced the cleansing of its retail point-of-sale data to another company that could do the work more efficiently than Tahari's in-house staff. By combining all of these approaches, Tahari came up with a powerful set of systems and tools that increased operational efficiency and the ability to take advantage of market trends while allowing the firm to concentrate on its core competency—fashion design.

APPLICATION SOFTWARE PACKAGES

During the past several decades, many systems have been built on an application software package foundation. Many applications are common to all business organizations—for example, payroll, accounts receivable, general ledger, or inventory control. For such universal functions with standard processes that do not change a great deal over time, a generalized system will fulfil the requirements of many organizations.

If a software package can fulfil most of an organization's requirements, the company does not have to write its own software. The company can save time and money by using the prewritten, predesigned, pretested software programs from the package. Package vendors supply much of the ongoing maintenance and support for the system, including enhancements to keep the system in line with ongoing technical and business developments.

If an organization has unique requirements that the package does not address, many packages include capabilities for customization. Customization features allow a software package to be modified to meet an organization's unique requirements without destroying the integrity of the package software. If a great deal of customization is required, additional programming and customization work may become so expensive and time consuming that they negate many of the advantages of software packages.

Figure 4 shows how package costs in relation to total implementation costs rise with the degree of customization. The initial purchase price of the package can be deceptive because of these hidden implementation costs. If the vendor releases new versions of the package, the overall costs of customization will be magnified because these changes will need to be synchronized with future versions of the software.

As the number of modifications to a software package rise, so does the cost of implementing the package. Savings promised by the package can be whittled away by excessive changes.

When a system is developed using an application software package, systems analysis will include a package evaluation effort. The most important evaluation criteria are the functions provided by the package, flexibility, user friendliness, hardware and software resources, database requirements, installation and maintenance efforts, documentation, vendor quality, and cost. The package evaluation process often is based on a Request for Proposal (RFP), which is a detailed list of questions submitted to packaged-software vendors.

When a software package solution is selected, the organization no longer has total control over the system design process. Instead of tailoring the system design specifications directly to user requirements, the design effort will consist of trying to mould user requirements to conform to the features of the package. If the organization's requirements conflict with the way the package works and the package cannot be customized, the organization will have to adapt to the package and change its procedures. Even if the organization's business processes seem compatible with those supported by a software package, the package may be too constraining if these business processes are continually changing (Prahalad and Krishnan, 2002).

OUTSOURCING

If a firm does not want to use its internal resources to build or operate information systems, it can outsource the work to an external organization that specializes in providing these services. Application service providers (ASPs), which we describe in Chapter 6, are one form of outsourcing. Subscribing companies would use the software and computer hardware provided by the ASP as the technical platform for their systems. In another form of outsourcing, a company could hire an external vendor to design and create the software for its system, but that company would operate the system on its own computers.

Outsourcing has become popular because some organizations perceive it as providing more value than an in-house computer centre or information systems staff. The provider of outsourcing services benefits from economies of scale and complementary core competencies that would be difficult for a firm that does not specialize in information technology services to replicate.

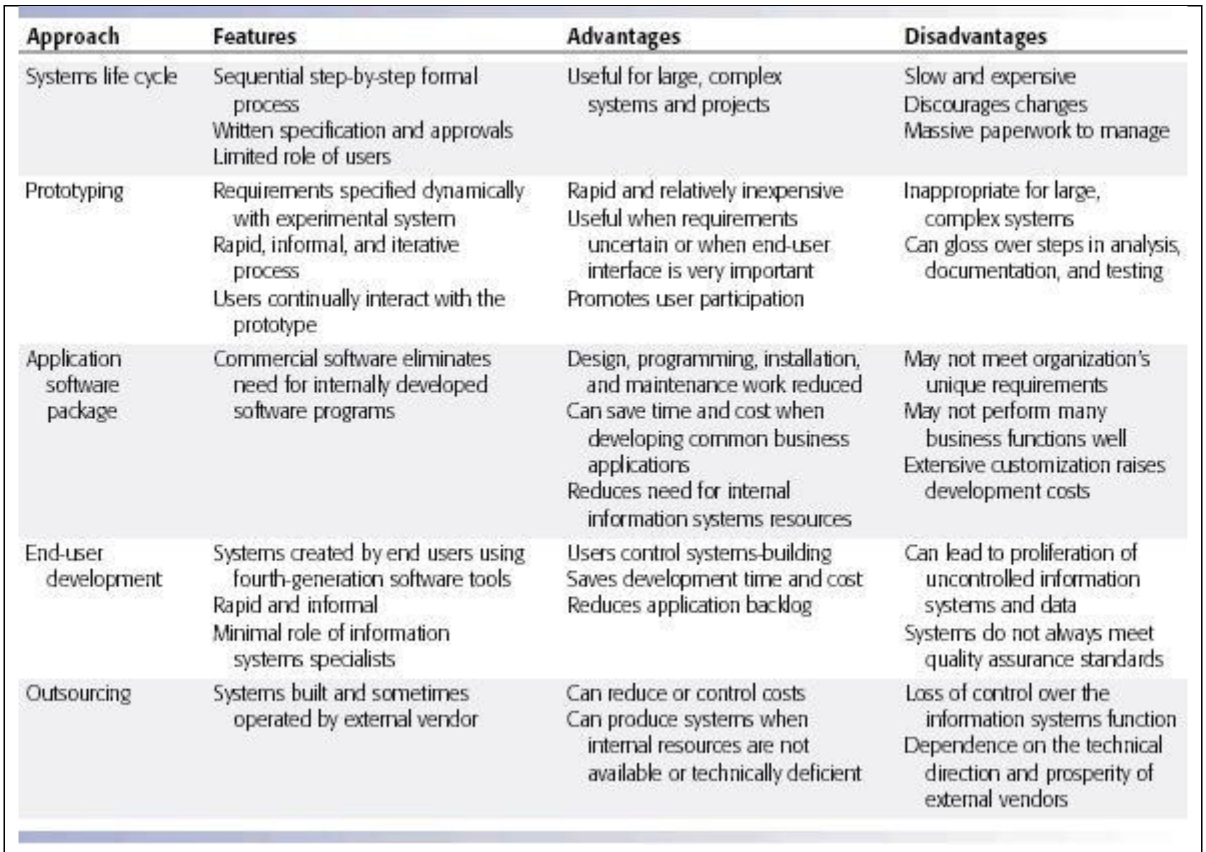
The vendor's specialized knowledge and skills can be shared with many different customers, and the experience of working with so many information systems projects further enhance the vendor's expertise. Outsourcing enables a company with fluctuating needs for computer processing to pay for only what it uses rather than build its own computer centre, which would be underutilized when there is no peak load. Some firms outsource because their internal information systems staff cannot keep pace with technological change or innovative business practices or because they want to free up scarce and costly talent for activities with higher paybacks.

Not all organizations benefit from outsourcing, and the disadvantages of outsourcing can create serious problems for organizations if they are not well understood and managed. Many firms underestimate costs for identifying and evaluating vendors of information technology services, for transitioning to a new vendor, and for monitoring vendors to make sure they are fulfilling their contractual obligations. These hidden costs can easily undercut anticipated benefits from outsourcing. When a firm allocates the responsibility for developing and operating its information systems to another organization, it can lose control over its information systems function. If the organization lacks the expertise to negotiate a sound contract, the firm's dependency on the vendor could result in high costs or loss of control over technological direction

Firms should be especially cautious when using an outsourcer to develop or to operate applications that give it some type of competitive advantage. A firm is most likely to benefit from outsourcing if it understands exactly how the outsourcing vendor will provide value and can manage the vendor relationship using an appropriate outsourcing strategy.

Table 14-6 compares the advantages and disadvantages of each of the systems-building alternatives.

Figure 12 Comparison of Systems-Development Approaches



Approach	Features	Advantages	Disadvantages
Systems life cycle	Sequential step-by-step formal process Written specification and approvals Limited role of users	Useful for large, complex systems and projects	Slow and expensive Discourages changes Massive paperwork to manage
Prototyping	Requirements specified dynamically with experimental system Rapid, informal, and iterative process Users continually interact with the prototype	Rapid and relatively inexpensive Useful when requirements uncertain or when end-user interface is very important Promotes user participation	Inappropriate for large, complex systems Can gloss over steps in analysis, documentation, and testing
Application software package	Commercial software eliminates need for internally developed software programs	Design, programming, installation, and maintenance work reduced Can save time and cost when developing common business applications Reduces need for internal information systems resources	May not meet organization's unique requirements May not perform many business functions well Extensive customization raises development costs
End-user development	Systems created by end users using fourth-generation software tools Rapid and informal Minimal role of information systems specialists	Users control systems-building Saves development time and cost Reduces application backlog	Can lead to proliferation of uncontrolled information systems and data Systems do not always meet quality assurance standards
Outsourcing	Systems built and sometimes operated by external vendor	Can reduce or control costs Can produce systems when internal resources are not available or technically deficient	Loss of control over the information systems function Dependence on the technical direction and prosperity of external vendors

The Window on Organizations describes how some financial services firms deal with the issue of selecting systems-building alternatives. Some firms opt to purchase the technology for new wealth management systems from outside vendors because they believe their strategic advantage lays in their knowledge of clients and investment selection. Other firms believe that the technology is a source of competitive differentiation as well and choose to build their systems in-house.

SMD Ejaz
Assistant Professor
AITS, Rajampeta
WhatsApp: 9133915846