Module - 3

System Development Models and System Analysis and Design

3.1 Terminologies

3.11 Various Terminologies used in System Development Models and System Analysis and Design are defined in below section.

3.12 System: A system is an orderly grouping of interdependent components linked together according to a plan to achieve a specific objective.

3.13 Characteristics of a system: Organization, interaction, interdependence, integration and a central objective.

3.14 Elements of a system: Input, processor(s), control, output, feedback, environment, boundaries and interface.

3.15 Types of system:

3.15 a. Physical or abstract system

Physical Systems are tangible entities that may be static or dynamic in operation.
Abstract Systems are conceptual or non-physical entities.

3.15 b. Open or closed system.

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.
A closed system is isolated from environmental influences.

3.16 Systems Development: The activities that go into producing an information systems solution to an organizational problem or opportunity. Systems development is a structured kind of problem solving with distinct activities. These activities consist of recognition of need, feasibility study, analysis, design, implementation, post-implementation, and maintenance.

3.17 Systems Analysis: The analysis of a problem that the organization will try 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.

3.18 System Design: Details how a system will meet the information requirements as determined by the systems analysts.

3.19 Logical Design: Lays out the components of the information system and their relationship to each other, as they would appear to users.

3.20 Physical Design: The process of translating the abstract logical model into the specific technical design for the new system.

3. 2 Concept of System Development Life Cycle (SDLC)

3.21 Systems Development

- The activities that go into producing an information systems solution to an organizational problem or opportunity.
- Systems development is a structured kind of problem solving with distinct activities. These activities consist of recognition of need, feasibility study, analysis, design, implementation, post-implementation, and maintenance.

3.22 Systems Analysis

The analysis of a problem that the organization will try to solve with an information system may be termed as Systems Analysis. 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.

3.23The Systems Development Life Cycle (SDLC)

SDLC is a logical sequence of activities used to identify new systems needs and to develop new systems to support those needs. SDLC is a model for reducing risk through planning, execution, control, and documentation of key activities.

3.3 Various stages involved in SDLC

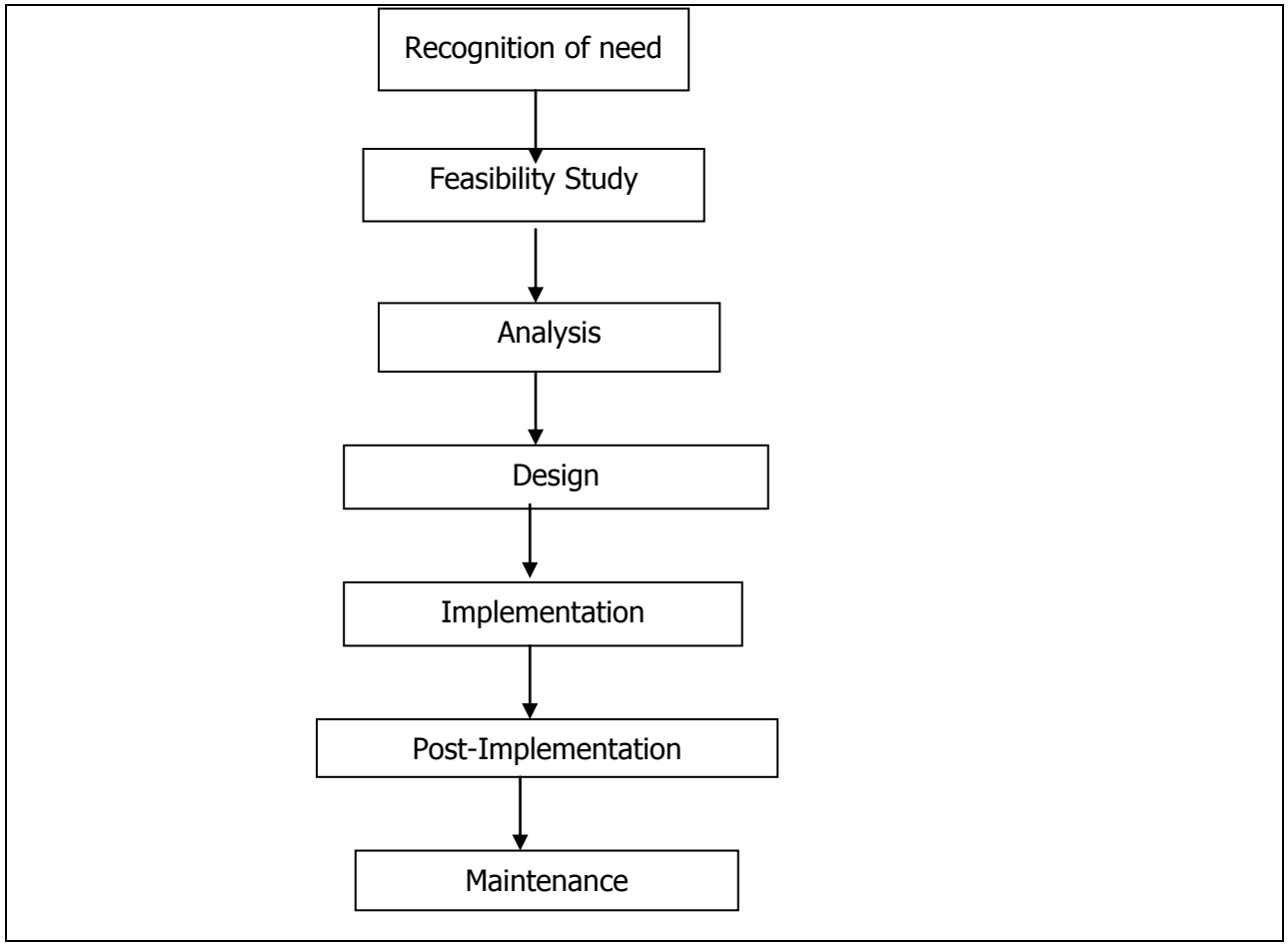


Figure 3.1 System Development Life Cycle (SDLC)

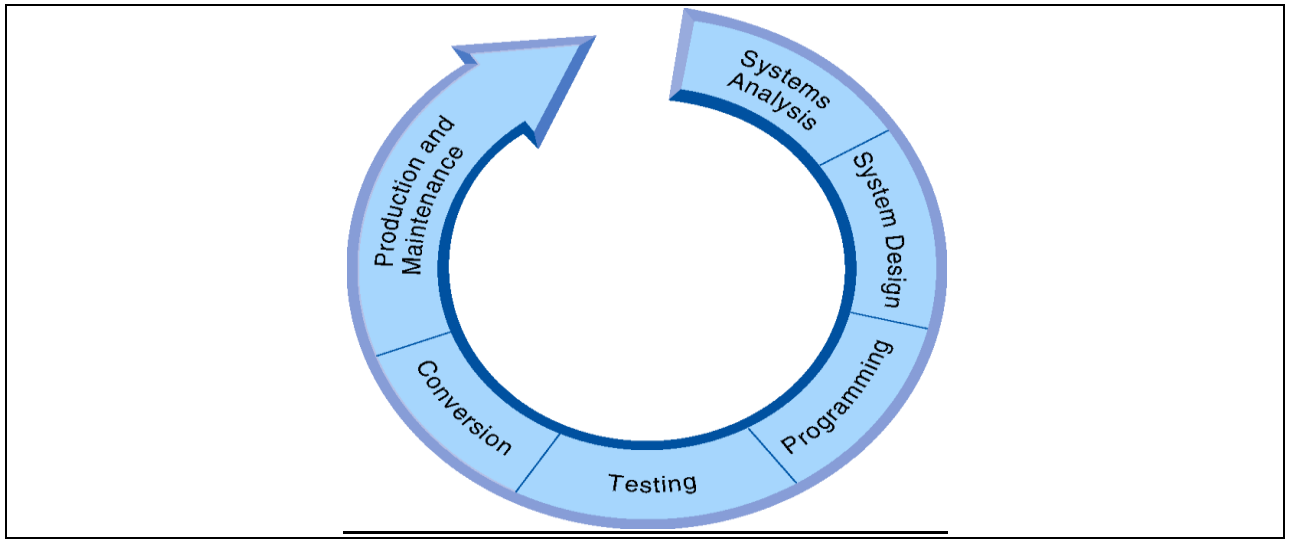
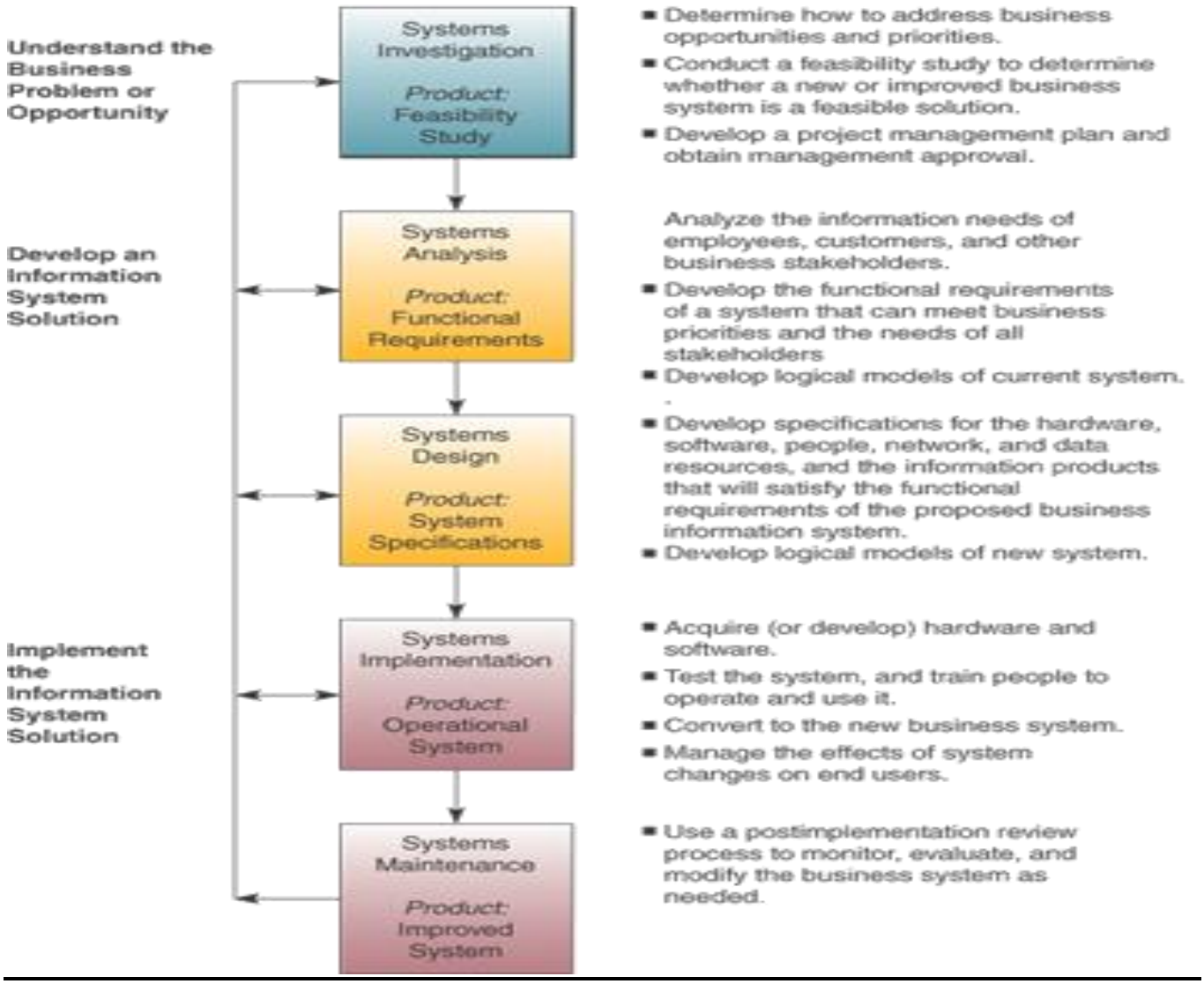


Table 3.1 various stages of System Development Life Cycle

Stage	Key Question	Result
1. Recognition of need: Preliminary survey / Initial investigation	What is the problem or opportunity	Statement of scope and objective Performance criteria
2. Feasibility study: a) Evaluation of existing system and procedures b) Analysis of alternative candidate systems c) Cost estimates.	What are the user's demonstrable needs? Is the problem worth solving? How can the problem be redefined?	Technical / behavioral feasibility Cost/benefit analysis System scope and objectives Statement of new scope and objectives
3. Analysis**:** a) Detailed evaluation of present system b) Data Collection	What must be done to solve the problem? What are the facts?	Logical model of system E.g. data directory, data flow diagram Pertinent data
4. Design: General design specifications Detailed design specifications Output Input Result	In general, how must the problem be solved? Specifically, how must the problem be solved? What is the system (processing) flow? Does the user approve the system?	Design of alternative solutions Find cost/benefit analysis Hardware specifications Cost estimates Implementation specification Implementation schedule Approval of systems by user programs
Program construction Testing Unit testing Combined module testing User acceptance testing	How well do individual programs/modules test out? How ready are programs for acceptance test?	Test plans Security, audit, and operating procedures Actual hardware use Formal system test
5. Implementation: User training File/System conversion	What is the actual operation? Are user manuals ready? Are there delays in loading files?	Training program User-friendly documentation
6. Post-implementation and maintenance: Evaluation Maintenance Enhancements	Is the key system running? Should the system be modified?	User requirements User standards met Satisfied user



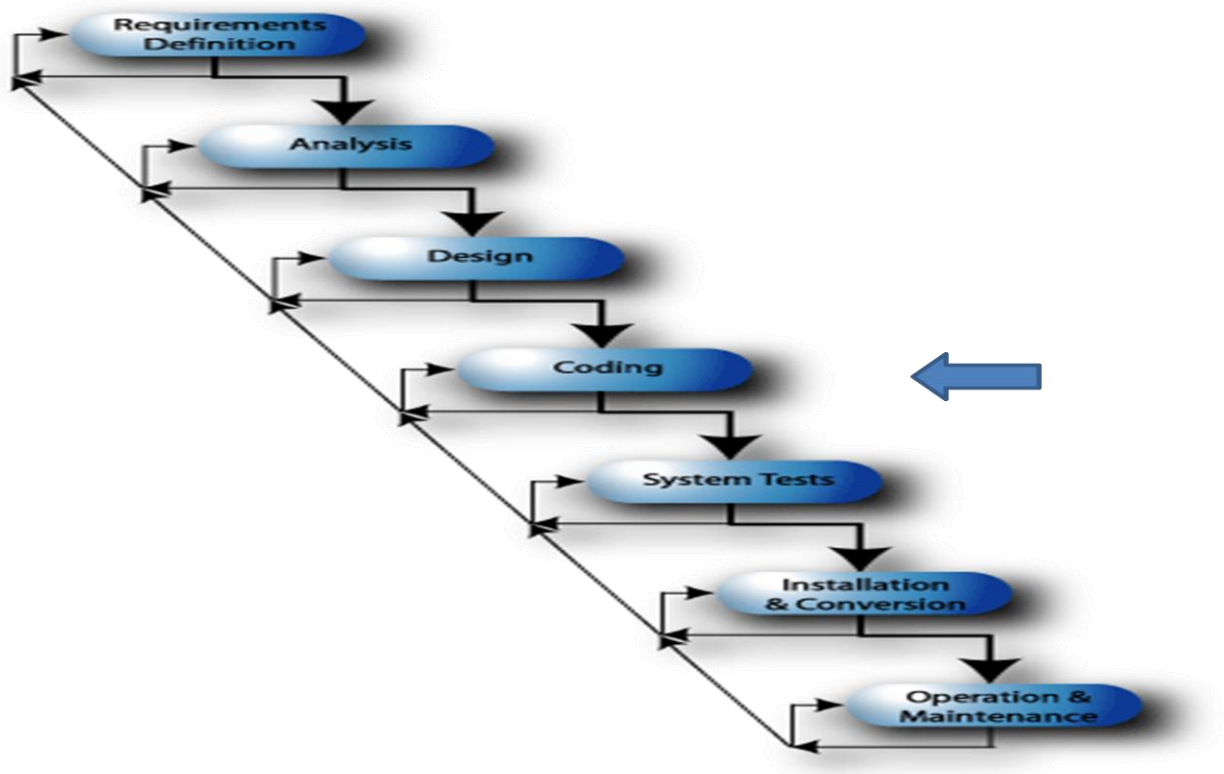
Description of SDLC stages

3.3 Software Process Models

Development strategies to solve software problems are referred to as process models or software process model. A process model for software is chosen based on the nature of the project to be used, the controls and deliverables that are required.

3.31 Linear Model or Waterfall Model:

A classic SDLC first identified in 1970 as a formal alternative to the code-and-fix software development method prevalent at that time.



Strengths:

- Tackles complexity in an orderly way, working well for projects that are well understood but still complex.
- It is easy to understand, with a simple goal-to complete required activities.
- It provides a template into which methods for analysis, design, code, test, and support can be placed.
- It works well when quality requirements dominate cost and schedule requirements.
- It allows for tight control by project manager.
- Its milestones are well understood.
- It is easy to track the progress of the project using a time line or Gantt chart – the completion of each phase is used as a milestone.

Weaknesses:

- It has an inherently linear sequential nature – any attempt to go back two or more phases to correct a problem or deficiency results in major increases in cost and schedule.
- It does not handle the reality of iterations among phases so common in software development.
- It can present a false impression status and progress – “35% done”

- Integration happens in one big bang at the end.
- There is insufficient opportunity for a customer to preview the system until very late in the life cycle. Users are involved while gathering requirements in the beginning and at the end, during acceptance testing.
- Users cannot see quality until the end
- It is not possible for the user to get used to the system gradually. All training must occur at the end of the life cycle.
- Each phase is a prerequisite for succeeding activities
- All requirements must be known at the beginning of the life cycle. The model is not equipped to handle dynamic changes in requirements over the life cycle, as deliverables are “frozen”
- It is document-driven, and the amount of documentation can be excessive.
- The entire software project is being worked at one time. There is no way to partition the system for delivery of pieces of the system at one time.

When to use?

- The requirements and the implementation of those requirements are very well understood
- Stable product definition and well understood technical methodologies
- A project to build another of the same type or product (existing designs)
- New versions
- Porting an existing product to a new plateform

Essence:

- It progresses through the orderly sequence steps.
- It assumes that each subsequent phase will begin when activities in the current phase have been completed.
- Each phase has defined entry and exit criteria: inputs and outputs.
- Transition from one phase to the next is accomplished by passing a formal review.
- At critical points on the waterfall model, base lines are established, the last of which is the product base line. The final base line is accompanied by an acceptance review.
- Porting an existing product to a new platform

3.32 The Prototype Model

The Prototype Model

Prototyping is more explicitly iterative and it actively promotes system design changes.Process of building an experimental system quickly and inexpensively for demonstration and evaluation so that end users can better determine information requirements. Prototyping replaces unplanned rework with planned iteration, with each version more accurately reflecting user requirements. Less formal, requirements are determined dynamically as the prototype is constructed.

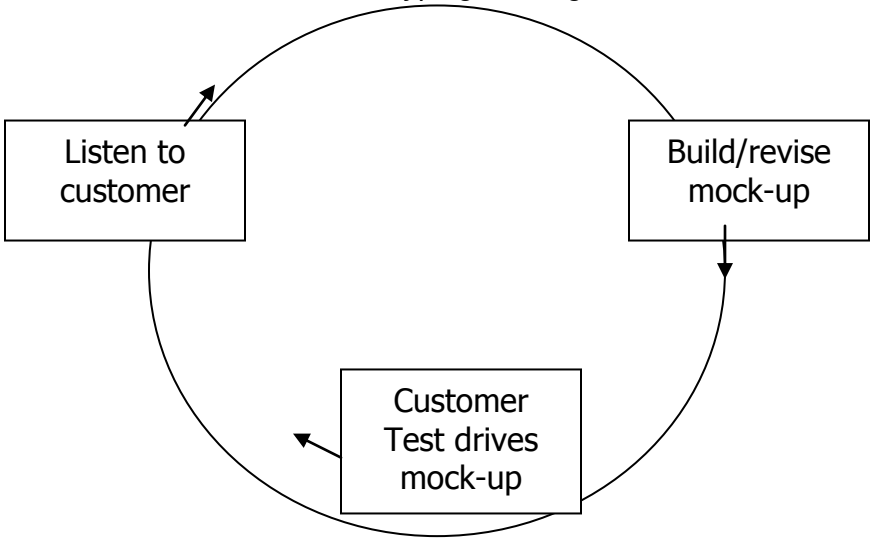
A customer defines a set of general objectives for software but does not identify detailed input, processing, or output requirements or the developer may be unsure of the efficiency of an algorithm, the adoptability of such system, or the form that human/machine interaction should take. This serves as a mechanism for identifying (defining) software requirements. The prototype can serve as "the first system". Users get a feel for the actual system and developers get to build something immediately.

Steps Involved in Prototype Model

1. Identify the user's basic requirements
2. Develop a working prototype
3. Use the prototype
4. Revise and enhance the prototype

- Requirements gathering: Developer and customer meet and define the overall objectives for the software, identify whatever requirements are known, and outline areas where further definition is mandatory.
- A "quick design" then occurs which focuses on a representation of those aspects visible to the customer/user (Eg. Input approaches and output formats).
- The quick design lends to the construction of a prototype.
- The prototype is evaluated by the customer/user and used to refine requirements for the software to be developed.
- Iteration occurs as the prototype is tuned to satisfy the needs of the customer, while at the same time enabling the developer to better understand what needs to be done.

The Prototyping Paradigm



Strengths:

- Most useful when there is some uncertainty about requirements or design solutions. (Difficult to specify in advance or they may change substantially as implementation progresses).
- Especially valuable for the design of the end-user interface of an Information System.
- When the system builders may be unsure of certain technical features of the design solution
- Applications that are oriented to simple data manipulation and records management.
- Systems that are based on batch processing or that rely on heavy calculations and complex procedural logic are unsuitable.
- Suitable for smaller applications

Weaknesses:

- Rapid prototyping can gloss over essential steps in system development
- Prototyping system to be fine tuned
- Hastily constructed systems may be difficult to maintain and support in regular production
- The customer sees what appears to be a working version of the software unaware that the prototype is held together.
- When informed that the product must be rebuilt so that high levels of quality can be maintained, the customer cries foul demands that “a few fixes” be applied to make a prototype a working product and often s/w development management relents.
- The developer often makes implementation compromises in order to get a prototype working quickly (algorithm, OS, PL).

- The less-than ideal choice becomes an integral part of the system.
- Need to be tested and documented.

When to use?

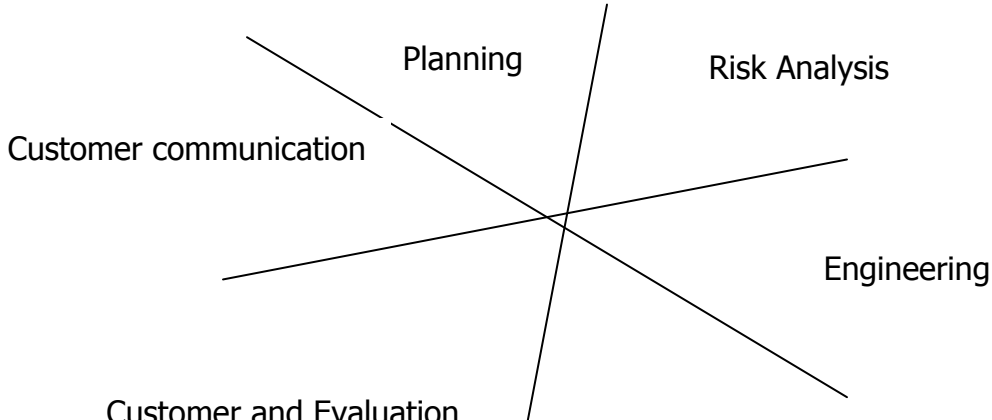
- On systems that may be modularized and that are scalable.
- On systems with reasonably well-known requirements.
- When the end users can be involved throughout the life cycle.
- When users are willing to become heavily involved in the use of automated tools.
- On systems that can be time boxed to deliver the functionality in increments.
- When reusable parts are available through automated software repositories.
- When the project team is familiar with the problem domain, skilled in the use of the development tools, and highly motivated.

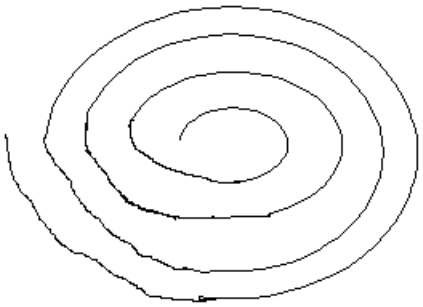
3.33 Spiral Model

Spiral Model

- Introduced by Dr. Barry Boehm, 1988
- Is an evolutionary process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model
- It provides the potential for rapid development for incremental versions of the software
- Software is developed in a series of incremental releases
- During early iterations, the incremental release might be a paper model or prototype
- During later iterations, increasingly more complete versions of the engineered system are produced
- The spiral model encompasses the strengths of the waterfall model while including risk analysis, risk management, and support and management processes.
- It also allows for the development of the product to be performed using a prototype technique or RAD through the use of 4 GLS

A spiral model is divided into a number of framework activities, also called **task regions**. Typically, there are six task regions.





1. **Customer Communication**: Tasks required establishing effective communication between developer and customer.
2. **Planning**: Tasks required defining resources, time lines, and other project related information.
3. **Risk Analysis**: Tasks required to assess both technical and management risks.
4. **Engineering**: Tasks required building one or more representations of the applications.
5. **Construction and release**: Tasks required constructing, testing, installing, and providing user support (Eg. Documentation and training).
6. **Customer Evaluation**: Tasks required obtaining customer feedback based on evaluation of the software representations created during the engineering stage and implemented during the installation stage.

Each of the regions is populated by a set of work tasks, called **task set**, that are adopted to the characteristics of the project to be undertaken. For each iteration, determine objectives, alternatives, and constraints; identify and resolve risks; evaluate alternatives; develop the deliverables for that iteration and verify that they are correct; plan the next iteration; and commit to an approach for the next iteration as shown in figure 3.3. The idea is to minimize risk through successive refinements of user requirements. Each “mini-project” (travel round the spiral) addresses one or more major risks, beginning with the highest. Risks include poorly understood requirements, poorly understood architecture, potential performance problems, and problems in the underlying technology.

Figure 3.3 Spiral Model of System Development Process

Strengths:

- The spiral model allows users to see the system early, through the use of rapid prototyping in the development life cycle
- It provides early indications of insurmountable risks, without much cost
- It allows users to be closely tied to all planning, risk analysis, development, and evaluation activities
- It splits a potentially large development effort into small chunks in which critical, high-risk functions are implemented first
- The model allows for flexible design by embracing the strengths of the waterfall model while allowing for iteration throughout the phases of that model
- It does not rely on the impossible task of getting the design perfect
- Management control of quality, correctness, cost, schedule, and staffing is improved through reviews at the conclusion of each iteration
- It enhances predictability through clarification of objectives
- All the money needed for the project need not be allocated up-front

- Takes the advantage of the strengths of the incremental model with incremental releases

Weaknesses:

- If the project is low-risk or small, this model can be an expensive one. The time spent evaluating the risk after each spiral is costly
- The model is complex, and developers, managers, and customers may find it too complicated to use
- The spiral may continue indefinitely, generated by each of the customer's responses to the build initiating a new cycle, closure (convergence on a solution) may be difficult to achieve
- Considerable risk assessment expertise is required
- Use of the model may be expensive and even unaffordable – time spent on planning, resetting objectives, doing risk analysis, and prototyping may be excessive
- Developers must be reassigned during non-development-phase activities
- It can be hard to define objective, verifiable milestones that indicate readiness to proceed through the next iteration
- Industry has not had as much experience with the spiral model as it has with others

When to use?

- For projects that represent a medium to high risk
- When it is unwise to commit to a long-term project due to potential changes in economic priorities, and when these uncertainties may limit the available time frame.
- When the technology is new and tests of basic concepts are required.
- When users are unsure of their needs
- When requirements are complex
- When significant changes are expected as with research or exploration
- For organizations that cannot afford to allocate all the necessary project money up-front, without getting some feed back along the way
- When benefits are uncertain and success is not guaranteed
- With computation-intensive systems, such as DSS
- With business projects as well as aerospace, defence, and engineering projects
- On long projects that may make managers or customers nervous
- To demonstrate quality and attainment of objectives in short period of time

3.34 Alternatives to SDLC methodologies

The traditional SDLC approach works best on projects in which users have a clear idea about their requirements. Projects that require major changes in existing processes, through reengineering or development of new processes or those that build upon inter-organizational and international systems using Web technologies indicate a need for alternatives or supplements to conventional SDLC methodologies.

Some alternatives:

- Prototyping
- Joint application design (JAD)
- Rapid application development (RAD)
- Object-oriented development (OO)
- Extreme Programming (XP)
- Component-based development

Prototyping (*evolutionary development*):

- Instead of spending a lot of time producing very detailed specifications, the developers find out only generally what the users want. The developers do not develop the complete system all at once.
- Instead they *quickly* create a prototype, which either contains portions of the system of most interest to the users, or is a small-scale working model of the entire system.
- After reviewing the prototype with the users, the developers refine and extend it. This process is continued until the final specifications.

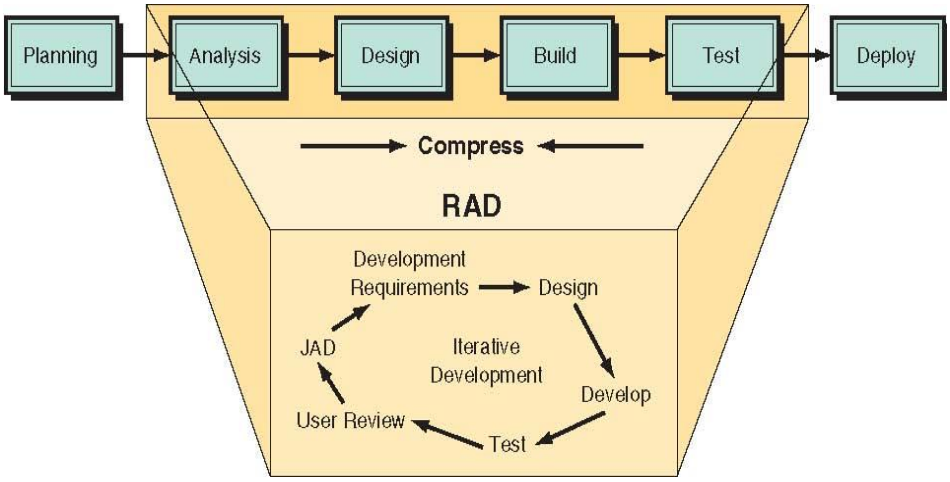
Joint application design (JAD)

- A group-based method for collecting user requirements and creating system designs. It is used within the systems analysis and design stages of the SDLC.
- Unlike the traditional SDLC, where the analysts interview individual users of the new information system to understand their needs JAD has a meeting in which all users meet simultaneously with analysts. During the meeting, all users jointly define and agree upon systems requirements.

Rapid application development (RAD)

- Rapid application development (RAD) methodologies and tools make it possible to develop systems faster, especially systems where the user interface is an important component.

  - GUI development environment: the ability to create many aspects of an application by “drag-and-drop” operations.
  - Reusable components: a library of common, standard “objects” such as buttons and dialog boxes.
  - Code generator. After the developer drags-and-drops the standard objects into the design, the package automatically writes computer programs to implement the reports, input screens, buttons, dialog boxes, and so forth.
  - Programming language: such as BASIC, Object Pascal, or C.



Object-oriented development (OO)

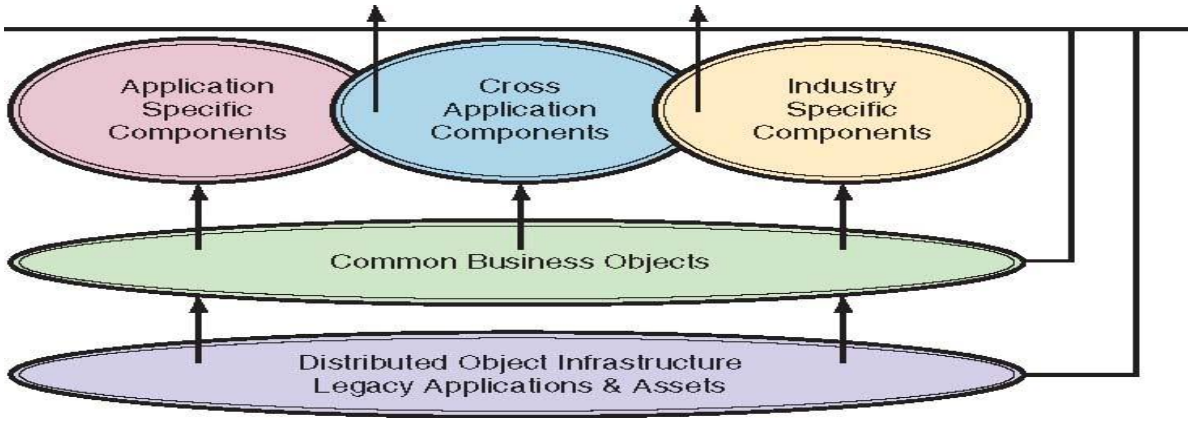
Object-oriented development (OO) is a fundamentally different view of computer systems than that found in traditional SDLC approaches. It begins not with the task to be performed, but with the aspects of the real world that must be modeled to perform that task.

Extreme programming (XP)

- Extreme programming (XP) is a discipline of software development based on values of simplicity, communication, and feedback.
- XP teams use a simple form of planning and tracking to decide what should be done next. Focused on business value, the team produces the software in a series of small, fully integrated releases that pass all the tests the customer has specified.

Component Based Development

- Component-based development is the evolution beyond objects. They are self-contained packages of functionality that have clearly defined, open interfaces that offer high-level application services.
- These business objects provide major chunks of application functionality (e.g., pre-programmed work flow, transaction processing, and user event notification) that can be connected together to create complete business applications.



3.4 Structured Analysis

Terminologies used in Structured Analysis are defined in this section.

Top-down refers to an approach that progress from the highest, most abstract level to the lowest level of detail – from the general to the specific.

Structured refers to the fact that techniques are instructions that are carefully drawn up, often step-by-step, with each step building upon a previous one.

Structured Analysis: Top-down method for defining system inputs, processes, and outputs, and for partitioning systems into sub-systems or modules that show a logical graphic model of information flow.

Data Flow Diagram (DFD): Also known as bubble chart". Data flow diagrams show how data flow to, from and within an information system and the processes that transform the data. DFD are constructed using four basic symbols. These symbols consist of the following:

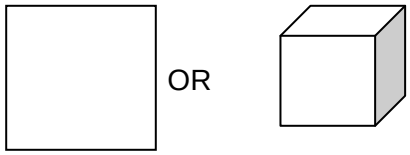
1. The data flow symbol, an arrow showing the flow of data
2. The process symbol, rounded boxes or bubbles depicting processes that transform the data
3. The data short symbol, an open rectangle indicating where data are stored
4. The external entity symbols either a rectangle or square, indicating the sources or destination of data.

Data Flows: The movement of data between processes, external entities, and data stores in a DFD.

Processes: Portray the transformation of input data flows to output data flows in a DFD. Each has a unique reference number and is named with a verb-object phase.

Data Stores: Manual or automated inventories of data. They consist of computer files or data bases, file cabinets, card files, microfiche, or a binder of paper reports. The name of the data store is written inside the data store symbol.

External Entities: Originators or receivers of information outside the scope of the system portrayed in the data flow diagram. Some time called outside interfaces.



Source or destination of data
(External Entity)

Data Flow

Process that transforms data Flow

Data Store

Data Dictionary: An automated or manual tool for storing and organizing information about the data maintained in a database. A data dictionary is a structured repository of data about data. It is a set of rigorous definitions of all DFD data elements and data structures.

Decision Tables: A decision table is a table of contingencies for defining a problem and the actions to be taken. It is a single representation of the relationships between conditions and actions.

A decision table consists of two parts: Stub and entry. The stub part is divided into an upper quadrant called the condition stub and a lower quadrant called the action stub. The entry part is divided into an upper quadrant, called the condition entry and a lower quadrant called the action entry.

Table 3.2 Elements and Definitions in a Decision Table.

Elements	Location	Definition
Condition stub	Upper left quadrant	Sets forth in question form the condition that may exist
Action stub	Lower left Quadrant	Outline in narrative form the action to be taken to match each condition.
Condition entry	Upper right quadrant	Provides answers to questions asked in the condition stub quadrant.
Action entry	Lower right quadrant	Indicates the appropriate action resulting from the answers to the condition in the condition entry.

Context Diagram: The context diagram always depicts an entire system as a single process with its major inputs and outputs. Subsequent diagrams can then break the system down into greater levels of detail.

3.50 Systems Design

Systems Design

Details how a system will meet the information requirements as determined by the systems and users. It consists of all the specifications that give the system its form and structure. Information systems design is an exacting and creative task demanding imagination, sensitivity to detail and expert skills.

Objectives:

1) The systems designer is responsible for considering alternative technology configuration for carrying out and developing the system as described by the analyst. This may involve analysis of the performance of different process of hardware and software, security capabilities of systems, network alternatives, and the portability or changeability of systems hardware.

2) Designers are responsible for the management and control of the technical realization of systems. Detailed programming specifications, coding of data, demonstration, testing and trial,

the actual procurement of the hardware, software, components are all the responsibility of the designers

3) 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 systems solutions.

Logical Design: Lays out the components of the information system and their relationship to each other as they would appear to the users. It shows what the system solution will do as opposed to how it is actually implemented physically. It describes inputs and outputs, processing functions to be performed, business procedures, data models, and controls - controls specify standards for acceptable performance and methods for measuring actual performance in relation to these standards.

Physical Design: The process of translating the abstract logical model into the specific technical design for the new system. It produces the actual specifications for hardware, software, physical databases, input/output media, manual procedures, and specific controls. Physical design provides the remaining specifications that transform the abstract logical design plan into a functioning system of people and machines.

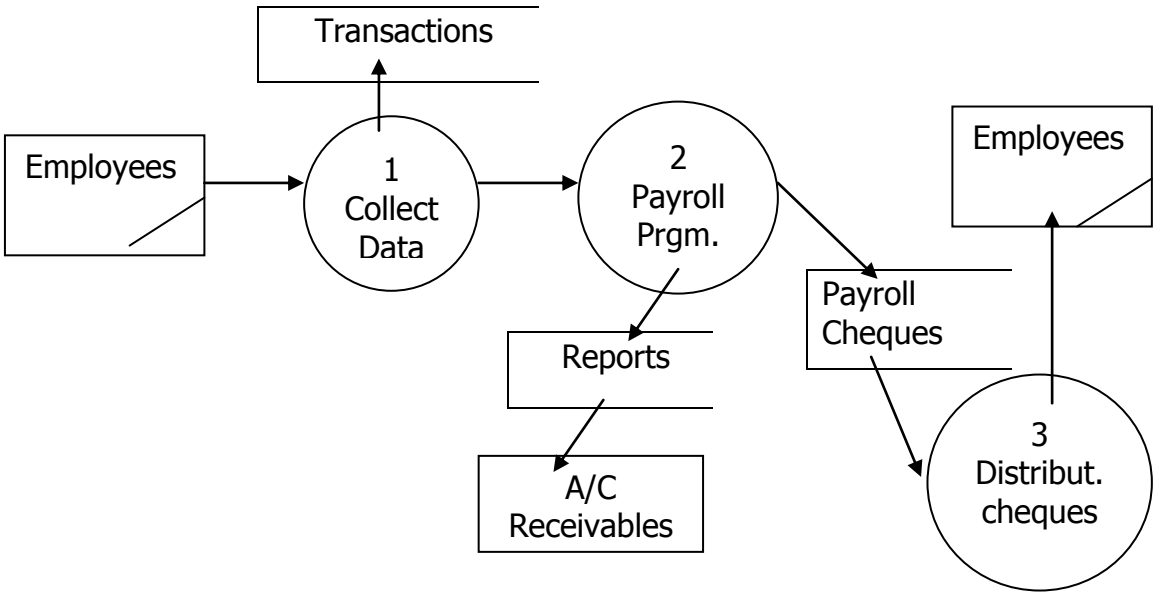
Structured Design: Encompasses a set of design rules and techniques that promotes program clarity and simplicity, thereby reducing the time and effort required for coding, debugging, and maintenance. The main principle of standard design is that a system should be designed from the top down in hierarchical fashion and refined to greater levels of detail.

The design should first consider the main function of a program or system, then break this function into sub functions and decompose each sub function until the lowest level of detail has been reached. In this manner, all high level logic and the design model are developed before detailed program code is written. If structured analysis has been performed, the structured specification document can serve as input to the design process.

Structured Chart: System documentation showing each level of design, the relationship among the levels, and the overall place in the design structure. It can document one program, one system, or part of one program.

Walk through: A review of specification or design document by a small group of people carefully selected based on the skills needed for the particular objectives being tested.

Data Flow Diagram – An Example



3.60 System Analyst

System Analyst

An Analyst is a person who conducts a methodical study and evaluation of an activity such as business to identify its desired objectives in order to determine procedures by which these objectives can be gained. The task of the systems analyst is to eliminate needs and resource constraints and to translate these into a viable operation. A system analyst is one who facilitates the study of the problems and needs of a business to determine how the business system and information technology can best solve the problems and accomplish improvements for the business. An analyst must possess various skills to extensively carry out the job. They may be divided into two categories.

A system analyst is one who facilitates the study of the problems and needs of a business to determine how the business system and information technology can best solve the problems and accomplish improvements for the business processes, improved information

systems, or new or improved computer application. Most of the time it is the combination of all the three.

When information technology is used, the analyst is responsible for the efficient capture of data from its business source, the flow of that data to the computer, the processing and storage of that data by the computer, and the flow of useful and timely information back to the business and its people.

Information technology is a contemporary term that describes the combination of computer technology (hardware and software) with telecommunications technology (data, image, and voice networks).

3.61 Various skills required for a system analyst.

For accomplishing all the things stated above, an analyst must possess various skills. They may be divided specifically into two categories namely interpersonal skills and technical skills. Both are required for system development. Interpersonal skills deal with relationships and the interface of the analyst with people in business. They are useful in establishing trust, resolving conflict, and communicating information. Technical skills, on the other hand, focus on procedures and techniques for operations analysis, systems analysis and computer science.

Interpersonal Skills:

- Communication: Having the ability to articulate and speak the language of the user, a "flare" of mediation, and a knack of working with all managerial levels in the organization.
- Understanding: Identifying problems and assembling their ramifications, having a group of company goals and objectives, and showing sensitivity to the impact of the system on people at work.
- Teaching: Educating people in use of computer systems, selling the system to the user, and giving support when needed.
- Selling: Selling ideas and promoting innovation in problem solving using computers.

Technical Skills:

- Creativity: Helping users model ideas into concrete plans and developing candidate system to match user requirements.

- Problem solving: Reducing problems to their elemental levels for analysis, developing alternative solutions to a given problem, and delineating the pros and cons of candidate system.
- Project Management: Scheduling, performing well under time constraints, coordinating teams and managing costs and expenditures.
- Dynamic Interface: Blending technical and non-technical considerations in functional specifications and general design.
- Questioning attitude and inquiring mind: Knowing the what, when, why, where, who, and how a system works.
- Knowledge of computers and the business function.
- A candidate system is a newly developed system designed to replace the current system.

Multifaceted Role of System Analyst

Among the roles an analyst performs are change agent, monitor, architect, psychologist, marketing man, motivator, and politician.

Marketing Man:

Marketing the system takes place at each step in the system life cycle. Marketing skills and persuasiveness are crucial to the success of the system. Let us see how important the marketing skills are, with an illustration.

In the feasibility study phase, the outcome, i.e. the feasibility report is a g good written presentation documenting the activities involving the candidate system. The pivotal step in marketing the proposed change. Invariably an analyst is expected to give an oral presentation to the end use.

The presentation must aim at informing, confirming, or persuading (marketing skills!).

Informing: This simply means communicating the decisions already reached on system recommendations and the resulting action plans to those who will participate in the presentation.

Confirming: A presentation with this purpose verifies facts and recommendations already discussed and agreed upon. Presentation should be complete. Confirming is itself part of the process of securing approval. It should reaffirm the benefits of the candidate system and provide a clear statement of results to be achieved.

Persuading: This is a presentation pitched toward selling ideas – attempts to convince executives to take action on recommendations for implementing a candidate system.

Politician:

A candidate system must be well designed and acceptable to the user. System acceptance is achieved through user participation in its development, effective user training and proper motivation to use the system. The analyst's role as a motivator becomes obvious during the first few weeks after implementation and during times when turnover results in new people being trained to work with the candidate system.

In implementing a candidate system, the analyst tries to appease all parties involved. Diplomacy and finesse is what the analyst requires. In as much as a politician must have the support of his/her constituency, so is the analyst's goal to have the support of the user's staff. He/she represents their thinking and tries to achieve their goals through computerization.

3.62 Database Administrator (DBA)

Database Administrator (DBA)

3.62. 1 Role of DBA

1. In charge of enterprise database development
2. Improves the integrity and security of organizational databases
3. Uses Data Definition Language (DDL) to develop and specify data contents, relationships, and structure
4. Stores these specifications in a data dictionary or a metadata repository

3.62.2 The functions of the Database Administrator (DBA)

The functions of the Database Administrator (DBA) include the following:

1. Schema Definition: The DBA creates the original Database Schema by writing a set of definitions that is translated by the Data Definition Language (DDL) compiler to a set of tables that is stored permanently in the data dictionary.

2. Storage: Structure and access method definition; The DBA creates appropriate storage structures and access methods by writing a set of definitions which is translated by the data storage and data definition language compiler.

3. Schema and Physical Organization Modification: Programmers accomplish the relatively rare modification either to the database schema or to the description of the physical storage organization by writing a set of definitions that is used by either the DDL compiler or the data storage and data definition language compiler to generate modifications to the appropriate internal system tables (Eg. Data Dictionary).

4. Granting of authorization for data process: The granting of different types of authorization allows the database administrator to regulate which parts of the special system structure that is consulted by the database system whenever access to the data is attempted in the system.

5. Integrity constraint specification: The data values stored in the database must satisfy certain consistency. For eg. The number of hours an employee may work in one week may not exceed a specified by the DBA. The integrity constraints are kept in a special system structure that is consulted by the database system.

****