System Analysis and Design (SAD)

UNIT I

What is a system?

System is an interrelated set of components, with identifiable boundary, working together for some purpose.
Examples

- Solar system
- Digestive systems
- Public transport system
- Central heating system
- Computer system
- Information system

A set of objects and relationships among the objects viewed as a whole and designed to achieve a purpose

A system has nine characteristics:

- Components or Subsystems
- Interrelated components
- A boundary
- A purpose
- An environment
- Interfaces
- Input
- Output
- Constraints

Component: An irreducible part or aggregation of parts that make up a system, also called a subsystem. A subsystem is simply a system within a system.
Automobile is a system composed of subsystems:

- Engine system
- Body system
- Frame system
- Each of these subsystem is composed of sub-sub --systems.
- Engine system: carburetor system, generator system, fuel system, and so son

Interrelated components

- Dependence of one subsystem on one or more subsystems

Boundary

- The line that marks the inside and outside of a system and that sets off the system form its environment

Purpose

- The overall goal or function of a system

Environment

- Everything external to a system that interacts with the system **Interface**
- Point of contact where a system meets its environment or where subsystems meet each other.

Constraint

- A limit to what a system can accomplish **Input**
- Whatever a system takes from its environment in order to fulfill its purpose **Output**
- Whatever a system returns from its environment in order to fulfill its purpose

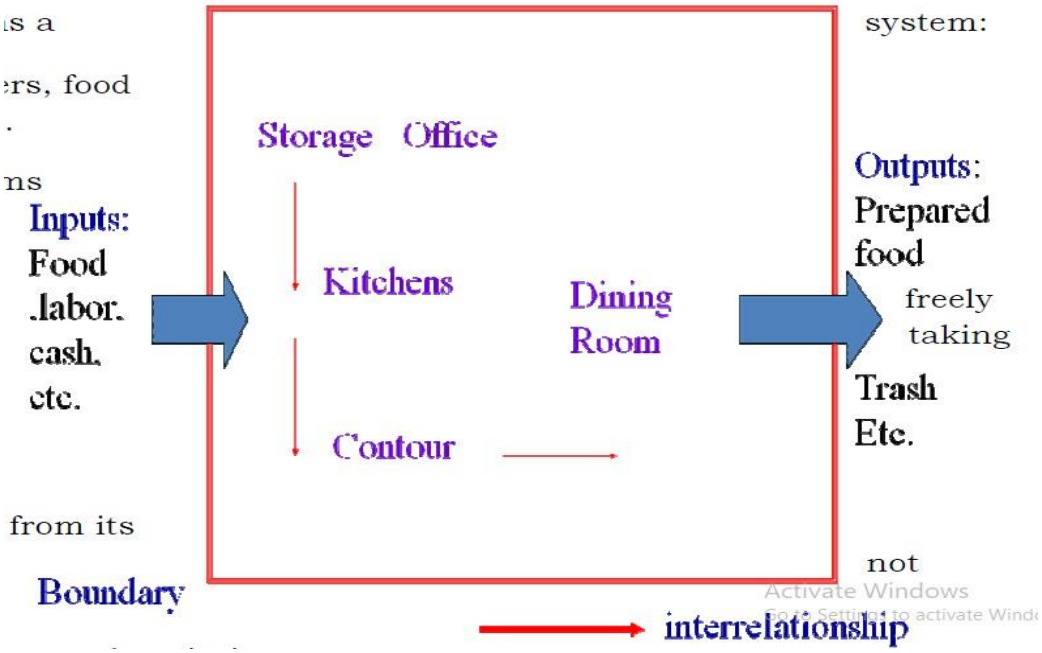


Fig: A Fast food Restaurants as a system

Environments: customers, food, distribution, banks, etc

Bad System

- Fail to meet requirements
- Poor performance
- Poor reliability
- Lack of usability • Example difficulties:
- Not to schedule
- Not to budget

- Runaway = 100% over budget or schedule
- Some problems are simply “wicked” problems

Then, what is good system?

What is an Information System?

O’Brien Defines: Any organized combination of people, hardware, software, communications networks, and data resources that stores, retrieves, transforms, and disseminates information in an organization.

HICKS Defins: IS as a formalised computer information system that can collect, store, process, and report data from various sources to provide the information necessary for management decision making.

Laudon and Laudon (1995) defines: IS as a set of interrelated components that collect (or retrieve), process, store and disseminate information to support decision making, control, analysis and visualisation in an organisation. Components of an Information System

- People
- Network
- Software
- Hardware
- Data

Levels of Information System

a) Operational-level Systems

Support operational managers by keeping track of the elementary activities and of the organisation. The principle purpose of systems at this level is to answer routine questions and track the flow of transactions through the organisation. Covers things such as sales, receipts, cash deposits, payroll, credit decisions, flow of materials.

b) Knowledge-level Systems

Support knowledge and data workers in an organisation. The purpose of these systems to help the organisation discover, organise and integrate new and existing knowledge in to the business, and to help control the flow of paperwork. These systems, specially in the form of collaboration tools, workstations, and office systems, are the fastest growing applications in business today.

c) **Management-level Systems**

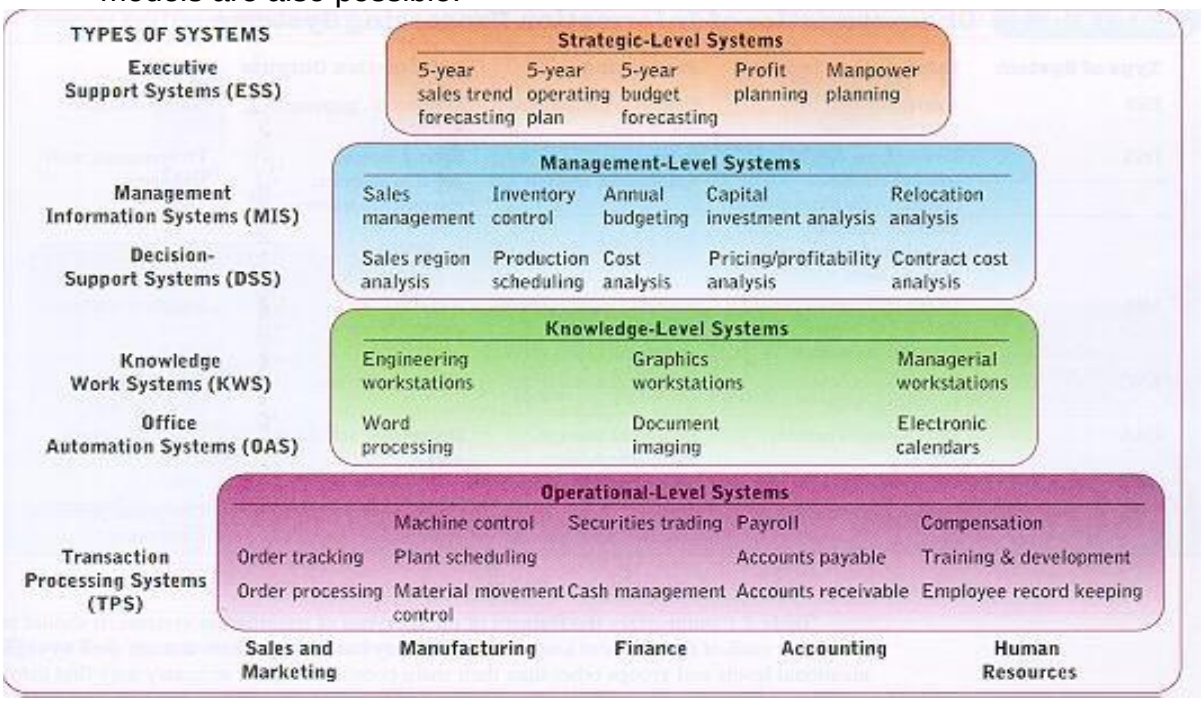
Designed to serve the monitoring, controlling, decision-making, and administrative activities of middle managers. These typically provide periodic reports rather than instant information on operations. Some of these systems support non-routine

decision-making, focusing on less-structured decisions for which information requirements are not always clear. This will often require information from outwith the organisation, as well as from normal operational-level data.

d) Strategic –Level Systems

Help senior management tackle and address strategic issues and long-term trends, both within the organisation and in the external environment. Principal concern is matching organisational capability to changes, and opportunities, occurring in the medium to long term (i.e. 5 - 10 years) in the external environment.

- Typically, an organisation might have operational, knowledge, management and strategic level systems for each functional area within the organisation. This would be based on the management model adopted by the organisation, so, while the most commonly-adopted systems structure would simply follow the standard functional model, structures reflecting bureaucratic, product and matrix models are also possible.



- As identified before, enterprise level information systems attempt to encompass the whole organisation in one system

Table 2.1 Characteristics of Information Processing Systems

Type of System	Information Inputs	Processing	Information Outputs	Users
ESS	Aggregate data; external, internal	Graphics; simulations; interactive	Projections; responses to queries	Senior managers
DSS	Low-volume data or massive databases optimized for data analysis; analytic models and data analysis tools	Interactive; simulations, analysis	Special reports; decision analyses; responses to queries	Professionals; staff managers
MIS	Summary transaction data; high-volume data; simple models	Routine reports; simple models; low-level analysis	Summary and exception reports	Middle managers
KWS	Design specifications; knowledge base	Modeling; simulations	Models; graphics	Professionals; technical staff
OAS	Documents; schedules	Document management; scheduling; communication	Documents; schedules; mail	Clerical workers
TPS	Transactions; events	Sorting; listing; merging; updating	Detailed reports; lists; summaries	Operations personnel; supervisors

Operational-level Systems

Transaction-Processing Systems (TPS)

- Basic business systems
- Perform daily routine transactions necessary for business functions
- At the operational level, tasks, resources and goals are predefined and highly structured
- Generally, five functional categories are identified, as shown in the diagram.

	TYPE OF TPS SYSTEM				
	Sales/ marketing systems	Manufacturing/ production systems	Finance/ accounting systems	Human resources systems	Other types (e.g., university)
Major functions of system	Sales management	Scheduling	Budgeting	Personnel records	Admissions
	Market research	Purchasing	General ledger	Benefits	Grade records
	Promotion	Shipping/receiving	Billing	Compensation	Course records
	Pricing	Engineering	Cost accounting	Labor relations	Alumni
	New products	Operations		Training	
Major application systems	Sales order information system	Materials resource planning systems	General ledger	Payroll	Registration system
	Market research system	Purchase order control systems	Accounts receivable/payable	Employee records	Student transcript system
	Pricing system	Engineering systems	Budgeting	Benefit systems	Curriculum class control systems
		Quality control systems	Funds management systems	Career path systems	Alumni benefactor system

Knowledge-level Systems

Office Automation Systems (OAS)

- Targeted at meeting the knowledge needs of *data workers* within the organisation
- Data workers tend to process rather than create information. Primarily involved in information use, manipulation or dissemination.
- Typical OAS handle and manage documents, scheduling and communication.

Knowledge Work Systems (KWS)

- Targeted at meeting the knowledge needs of *knowledge workers* within the organisation
- In general, knowledge workers hold degree-level professional qualifications (e.g. engineers, scientists, lawyers), their jobs consist primarily in creating new information and knowledge
- KWS, such as scientific or engineering design workstations, promote the creation of new knowledge, and its dissemination and integration throughout the organisation.

Management-level Systems

Management Information Systems (MIS)

- MIS provide managers with reports and, in some cases, on-line access to the organisations current performance and historical records
- Typically these systems focus entirely on internal events, providing the information for short-term planning and decision making.
- MIS summarise and report on the basic operations of the organisation, dependent on the underlying TPS for their data.

Decision-Support Systems (DSS)

- As MIS, these serve the needs of the management level of the organisation
- Focus on helping managers make decisions that are semi-structured, unique, or rapidly changing, and not easily specified in advance
- Use internal information from TPS and MIS, but also information from external sources
- Greater analytical power than other systems, incorporate modelling tools, aggregation and analysis tools, and support *what-if* scenarios
- Must provide user-friendly, interactive tools

Voyage-estimating Decision Support System

Strategic-level Systems Executive Support/Information Systems (ESS/EIS)

- Serve the strategic level of the organisation
- ESS/EIS address unstructured decisions and create a generalised computing and communications environment, rather than providing any fixed application or specific capability. Such systems are not designed to solve specific problems, but to tackle a changing array of problems

- ESS/EIS are designed to incorporate data about external events, such as new tax laws or competitors, and also draw summarised information from internal MIS and DSS
- These systems filter, compress, and track critical data, emphasising the reduction of time and effort required to obtain information useful to executive management
- ESS/EIS employ advanced graphics software to provide highly visual and easy-to-use representations of complex information and current trends, but they tend not to provide analytical models

http://www.cba.edu.kw/abo/pdf/SAD-chapter-1.ppt

Information systems analysis and design is a complex, challenging, and stimulating organizational process that a team of business and systems professionals uses to develop and maintain computer-based information systems. Although advances in information technology continually give us new capabilities, the analysis and design of information systems is driven from an organizational perspective.

Information systems analysis and design is therefore an organizational improvement process. Systems are built and rebuilt for organizational benefits. Benefits result from adding value during the process of creating, producing, and supporting the organization's products and services. Thus the analysis and design of information systems is based on your understanding of the organization's objectives, structure, and processes, as well as your knowledge of how to exploit information technology for advantage.

In the current business environment, the Internet, especially the World Wide Web, has been firmly integrated into an organization's way of doing business. Although you are probably most familiar with marketing done on the web and web-based retailing sites, such as eBay or Amazon.com, the overwhelming majority of business use of the web is business-to-business applications.

Methodologies are comprehensive, multiple-step approaches to systems development that will guide your work and influence the quality of your final product— the information system. A methodology adopted by an organization will be consistent with its general management style (e.g., an organization's orientation toward consensus management will influence its choice of systems development methodology). Most methodologies incorporate several development techniques.

Techniques are particular processes that you, as an analyst, will follow to help ensure that your work is well thought out, complete, and comprehensible to others on your project team. Techniques provide support for a wide range of tasks, including conducting thorough interviews to determine what your system should do, planning and managing the activities in a systems development project, diagramming the system's logic, and designing the reports your system will generate. Tools are typically computer programs that make it easy to use and benefit from techniques and to faithfully follow the guidelines of the overall development methodology.

To be effective, techniques and tools must both be consistent with an organization's systems development methodology. Techniques and tools must make it easy for systems developers to conduct the steps called for in the methodology. These three elements—methodologies, techniques, and tools—work together to form an organizational approach to systems analysis and design (see Figure). Fig: An organizational approach to systems analysis and design is driven by methodologies, techniques, and tools

DEVELOPING INFORMATION SYSTEMS AND THE SYSTEMS DEVELOPMENT LIFE CYCLE

Most organizations find it beneficial to use a standard set of steps, called a **systems development methodology**, to develop and support their information systems. Like many processes, the development of information systems often follows a life cycle. For example, a commercial product follows a life cycle in that it is created, tested, and introduced to the market. Its sales increase, peak, and decline. Finally, the product is removed from the market and replaced by something else. The **systems development life cycle (SDLC)** is a common methodology for systems development in many organizations; it features several phases that mark the progress of the systems analysis and design effort.

SDLC

Planning

SYSTEMS PLANNING The **systems planning phase** usually begins with a formal request to the IT department, called a **systems request**, which describes problems or desired changes in an information system or a business process. In many companies, IT systems planning is an integral part of overall business planning. When managers and users develop their business plans, they usually include IT requirements that generate systems requests. A systems request can come from a top manager, a planning team, a department head, or the IT department itself. The request can be very significant or relatively minor. A major request might involve a new information system or the upgrading of an existing system. In contrast, a minor request might ask for a new feature or a change to the user interface.

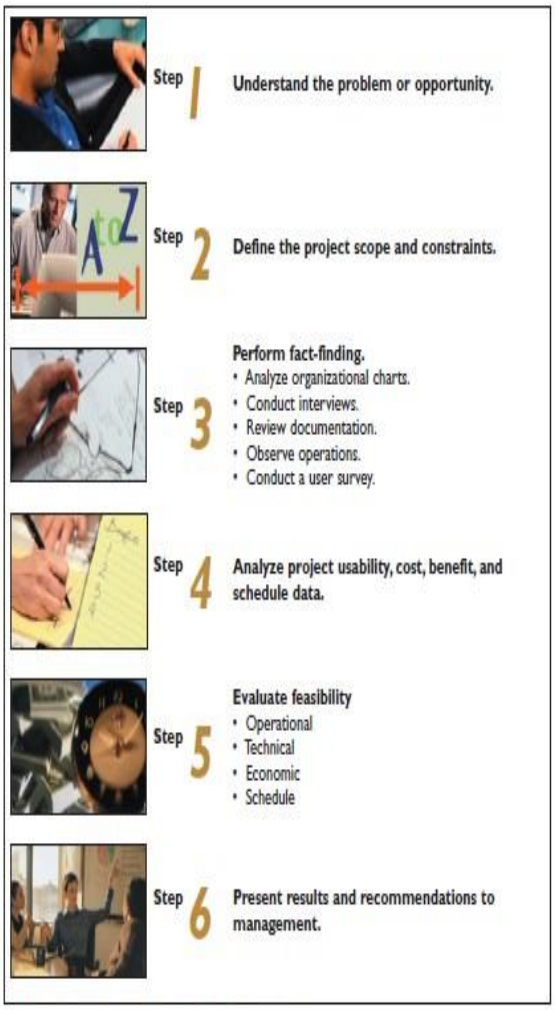


FIGURE 2-17 Six steps in a preliminary investigation.

The purpose of this phase is to perform a **preliminary investigation** to evaluate an ITrelated business opportunity or problem. The preliminary investigation is a critical step because the outcome will affect the entire development process. A key part of the preliminary investigation is a **feasibility study** that reviews anticipated costs and benefits and recommends a course of action based on operational, technical, economic, and time factors. Suppose you are a systems analyst and you receive a request for a system change or improvement. Your first step is to determine whether it makes sense to launch a preliminary investigation at all. Often you will need to learn more about business operations before you can reach a conclusion. After an investigation, you might find that the information system functions properly, but users need more training. In some situations, you might recommend a business process review, rather than an IT solution. In other cases, you might conclude that a full-scale systems review is necessary. If the development process continues, the next step is the systems analysis phase.

SYSTEMS ANALYSIS

The purpose of the **systems analysis phase** is to build a logical model of the new system. The first step is **requirements modeling**, where you investigate business processes and document what the new system must do to satisfy users. Requirements modeling continues the investigation that began during the systems planning phase. To understand the system, you perform fact-finding using techniques such as interviews, surveys, document review, observation, and sampling. You use the factfinding results to build business models, data and process models, and object models.

The deliverable for the systems analysis phase is the **system requirements document**. The system requirements document describes management and user requirements, costs and benefits, and outlines alternative development strategies.

THE SYSTEMS ANALYST

A systems analyst investigates, analyzes, designs, develops, installs, evaluates, and maintains a company's information systems. To perform those tasks, a systems analyst constantly interacts with users and managers within and outside the company. On large projects, the analyst works as a member of an IT department team; on smaller assignments, he or she might work alone.

Most companies assign systems analysts to the IT department, but analysts also can report to a specific user area such as marketing, sales, or accounting. As a member of a functional team, an analyst is better able to understand the needs of that group and how information systems support the department's mission. Smaller companies often use consultants to perform systems analysis work on an as-needed basis.

Responsibilities

The systems analyst's job overlaps business and technical issues. Analysts help translate business requirements into IT projects. When assigned to a systems development team, an analyst might help document business profiles, review business processes, select hardware and software packages, design information systems, train users, and plan ecommerce Web sites.
A systems analyst plans projects, develops schedules, and estimates costs. To keep managers and users informed, the analyst conducts meetings, delivers presentations, and writes memos, reports, and documentation.

Knowledge, Skills, and Education

A successful systems analyst needs technical knowledge, oral and written communication skills, an understanding of business operations, and critical thinking skills. Educational requirements vary widely depending on the company and the position. In a rapidly changing IT marketplace, a systems analyst must manage his or her own career and have a plan for professional development.

SYSTEMS DESIGN

The purpose of the **systems design phase** is to create a physical model that will satisfy all documented requirements for the system. At this stage, you design the user interface and identify necessary outputs, inputs, and processes. In addition, you design internal and external controls, including computer-based and manual features to guarantee that the system will be reliable, accurate, maintainable, and secure.

During the systems design phase, you also determine the application architecture, which programmers will use to transform the logical design into program modules and code. The deliverable for this phase is the **system design specification**, which is presented to management and users for review and approval. Management and user involvement is critical to avoid any misunderstanding about what the new system will do, how it will do it, and what it will cost. critical to avoid any misunderstanding about what the new system will do, how it will do it, and what it will cost.

SYSTEMS IMPLEMENTATION

During the **systems implementation phase,** the new system is constructed. Whether the developers use structured analysis or O-O methods, the procedure is the same — programs are written, tested, and documented, and the system is installed. If the system was purchased as a package, systems analysts configure the software and perform any necessary modifications. The objective of the systems implementation phase is to deliver a completely functioning and documented information system. At the conclusion of this phase, the system is ready for use. Final preparations include converting data to the new system's files, training users, and performing the actual transition to the new system. The systems implementation phase also includes an assessment, called a **systems evaluation**, to determine whether the system operates properly and if costs and benefits are within expectations.

SYSTEMS DEVELOPMENT GUIDELINES

Develop a Plan: Prepare an overall project plan and stick to it. Complete the tasks in a logical sequence. Develop a clear set of ground rules and be sure that everyone on the team understands them clearly.

Involve Users and Listen Carefully to Them: Ensure that users are involved in the development process, especially when identifying and modeling system requirements. When you interact with users, listen closely to what they are saying.

Use Project Management Tools and Techniques: Try to keep the project on track and avoid surprises. Create a reasonable number of checkpoints — too many can be burdensome, but too few will not provide adequate control.

Develop Accurate Cost and Benefit Information: Managers need to know the cost of developing and operating a system, and the value of the benefits it will provide.
You must provide accurate, realistic cost and benefit estimates, and update them as necessary.

Remain Flexible: Be flexible within the framework of your plan. Systems development is a dynamic process, and overlap often exists among tasks. The ability to react quickly is especially important when you are working on a system that must be developed rapidly.

What is methodology in SDLC?

A methodology is a comprehensive plan to be followed, multiple-step approach to system development that will guide your work and influence the quality of information system. It Include the model that needs to be followed, plus the tools and techniques that need to be used. It can be purchased or home-grown. Methodology is purchased because many information system organisations cannot afford to dedicate staff to the development and continuous improvement of a home-grown methodology. Whilst, Methodology vendors have a vested interest in keeping their methodologies current with the latest business and technology trends.

Methodology is written information in the form of books and other documents by detailing every activity, which needs to be implemented by system developers which includes documentation forms and reports that need to be provided by the project team.

Software Process

What is it? When you work to build a product or system, it's important to go through a series of predictable steps—a road map that helps you create a timely, high-quality result. The road map that you follow is called a "software process."
Who does it? Software engineers and their managers adapt the process to their needs and then follow it. In addition, the people who have requested the software have a role to play in the process of defining, building, and testing it.
Why is it important? Because it provides stability, control, and organization to an activity that can, if left uncontrolled, become quite chaotic. However, a modern software engineering approach must be "agile." It must demand only those activities, controls, and work products that are appropriate for the project team and the product that is to be produced.

What are the steps? At a detailed level, the process that you adopt depends on the software that you're building. One process might be appropriate for creating software for an aircraft avionics system, while an entirely different process would be indicated for the creation of a website.

What is the work product? From the point of view of a software engineer, the work products are the programs, documents, and data that are produced as a consequence of the activities and tasks defined by the process.

How do I ensure that I've done it right?

There are a number of software process assessment mechanisms that enable organizations to determine the "maturity" of their software process. However, the quality, timeliness, and long-term viability of the product you build are the best indicators of the efficacy of the process that you use.

Software process presents a description of a process from some particular perspective as:

- Specification.
- Design.
- Validation.
- Evolution.

List of General Software Process Models

The list of traditional software development models are:

1. Waterfall model
2. Prototype model
3. Rapid application development model
4. Incremental model.
5. Agile Model
6. Iterative model.
7. Spiral model.

The Waterfall Model

The waterfall model is the classical model of software engineering. This model is one of the oldest models and is widely used in government projects and in many major companies. As this model emphasizes planning in early stages, it ensures design flaws before they develop. In addition, its intensive document and planning make it work well for projects in which quality control is a major concern.

The pure waterfall lifecycle consists of several non-overlapping stages, as shown in the following figure. The model begins with establishing system requirements and software requirements and continues with architectural design, detailed design, coding, testing, and maintenance.

The waterfall model serves as a baseline for many other lifecycle models.

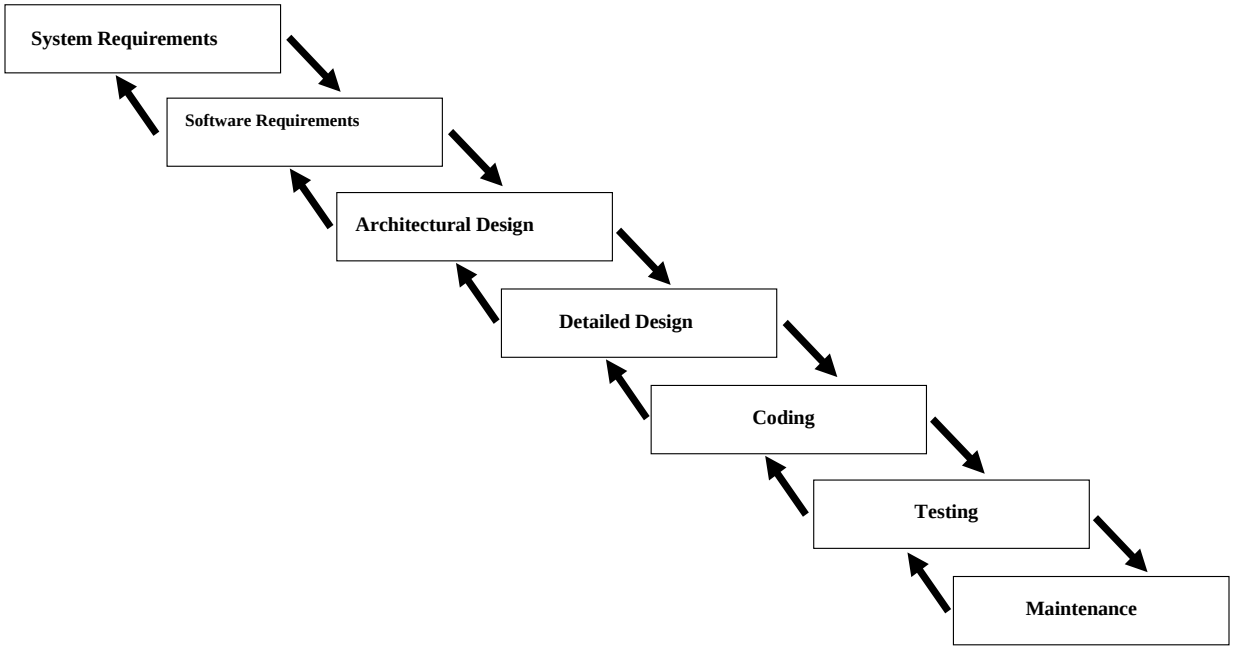


Fig. 2 Waterfall Model[4].

1. **System requirements**: Establishes the components for building the system, including the hardware requirements, software tools, and other necessary components. Examples include decisions on hardware, such as plug-in boards (number of channels, acquisition speed, and so on), and decisions on external pieces of software, such as databases or libraries.

2. **Software requirements**: Establishes the expectations for software functionality and identifies which system requirements the software affects. Requirements analysis includes determining interaction needed with other applications and databases, performance requirements, user interface requirements, and so on.

3. **Architectural design**: Determines the software framework of a system to meet the specific requirements. This design defines the major components and the interaction of those components, but it does not define the structure of each component. The external interfaces and tools used in the project can be determined by the designer.

4. **Detailed design**: Examines the software components defined in the architectural design stage and produces a specification for how each component is implemented.

   "Architecture is concerned with the selection of architectural elements, their interaction, and the constraints on those elements and their interactions…Design is concerned with the modularization and detailed interfaces of the design elements,

their algorithms and procedures, and the data types needed to support the architecture and to satisfy the requirements."

5. **Coding**: Implements the detailed design specification.
6. **Testing**: Determines whether the software meets the specified requirements and finds any errors present in the code.
7. **Maintenance:** Addresses problems and enhancement requests after the software releases.

In each stage, documents that explain the objectives and describe the requirements for that phase are created. At the end of each stage, a review to determine whether the project can proceed to the next stage is held. Your prototyping can also be incorporated into any stage from the architectural design and after.
The waterfall method does not prohibit returning to an earlier phase, for example, returning from the design phase to the requirements phase. However, this involves costly rework. Each completed phase requires formal review and extensive documentation development. Thus, oversights made in the requirements phase are expensive to correct later.

Because the actual development comes late in the process, one does not see results for a long time. This delay can be disconcerting to management and customers. Many people also think that the amount of documentation is excessive and inflexible.

Advantages:

1. Easy to understand and implement.
2. Widely used and known (in theory!).
3. Reinforces good habits: define-before- design, design-before-code.
4. Identifies deliverables and milestones.
5. Document driven, URD, SRD, … etc. Published documentation standards
6. Works well on mature products and weak teams.

Disadvantages:

1. Idealized, doesn't match reality well. 2. Doesn't reflect iterative nature of exploratory development.
3. Unrealistic to expect accurate requirements so early in project.
4. Software is delivered late in project, delays discovery of serious errors.
5. Difficult to integrate risk management. 6. Difficult and expensive to make changes to documents, "swimming upstream".
7. Significant administrative overhead, costly for small teams and projects

The 5 Essential Stages of a RAD Model

There are several stages to go through when developing a RAD model including analysis, designing, building, and the final testing phase. These steps can be divided to make them more easily understandable and achievable. The following describes the steps included in all RAD models:

Stage 1: Business Modeling
This step in the RAD model takes information gathered through many business related sources. The analysis takes all the pertinent information from the company. This info is then combined into a useful description of how the information can be used, when it is processed, and what is making this specific information successful for the industry.

Stage 2: Data Modeling
During the Data Modeling stage, all the information gathered during the Business Modeling phase is analyzed. Through the analysis, the information is grouped together into different groups that can be useful to the company. The quality of every group of information is carefully examined and given an accurate description. A relationship between these groups and their usefulness as defined in the Business Modeling step is also established during this phase of the RAD model.

Stage 3: Process Modeling
The Process Modeling phase is the step in the RAD model procedure where all the groups of information gathered during the Data Modeling step are converted into the required usable information. During the Process Modeling stage changes and optimizations can be done and the sets of data can be further defined. Any descriptions for adding, removing, or changing the data objects are also created during this phase.

Stage 4: Application Generation
The Application Generation step is when all the information gathered is coded, and the system that is going to be used to create the prototype is built. The data models created are turned into actual prototypes that can be tested in the next step.

Stage 5: Testing and Turnover
The Testing and Turnover stage allows for a reduced time in the overall testing of the prototypes created. Every model is tested individually so that components can quickly be identified and switched in order to create the most effective product. By this point in the RAD model, most of the components have already been examined, so major problems with the prototype are not likely.

Iterative Development:
The problems with the Waterfall Model created a demand or a new method of developing systems which could provide faster results, require less up-front information, and offer greater flexibility. With Iterative Development, the project is divided into small parts. This allows the development team to demonstrate results earlier on in the process and obtain valuable feedback from system users. Often, each iteration is actually a mini-Waterfall

process with the feedback from one phase providing vital information for the design of the next phase. In a variation of this model, the software products, which are produced at the end of each step (or series of steps), can go into production immediately as incremental releases.

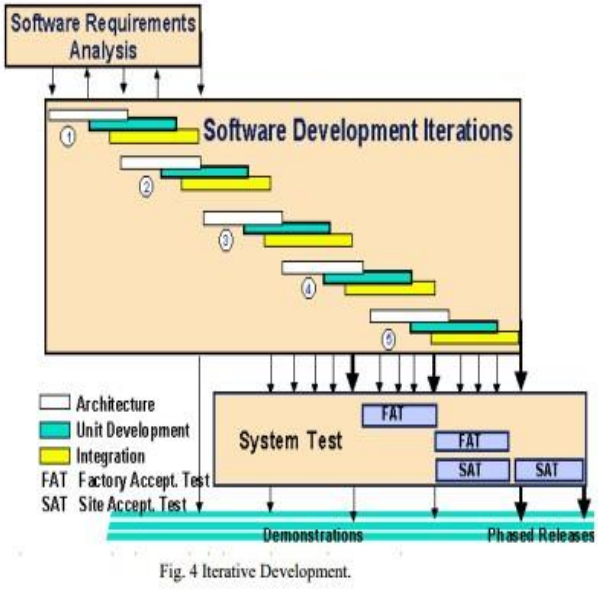


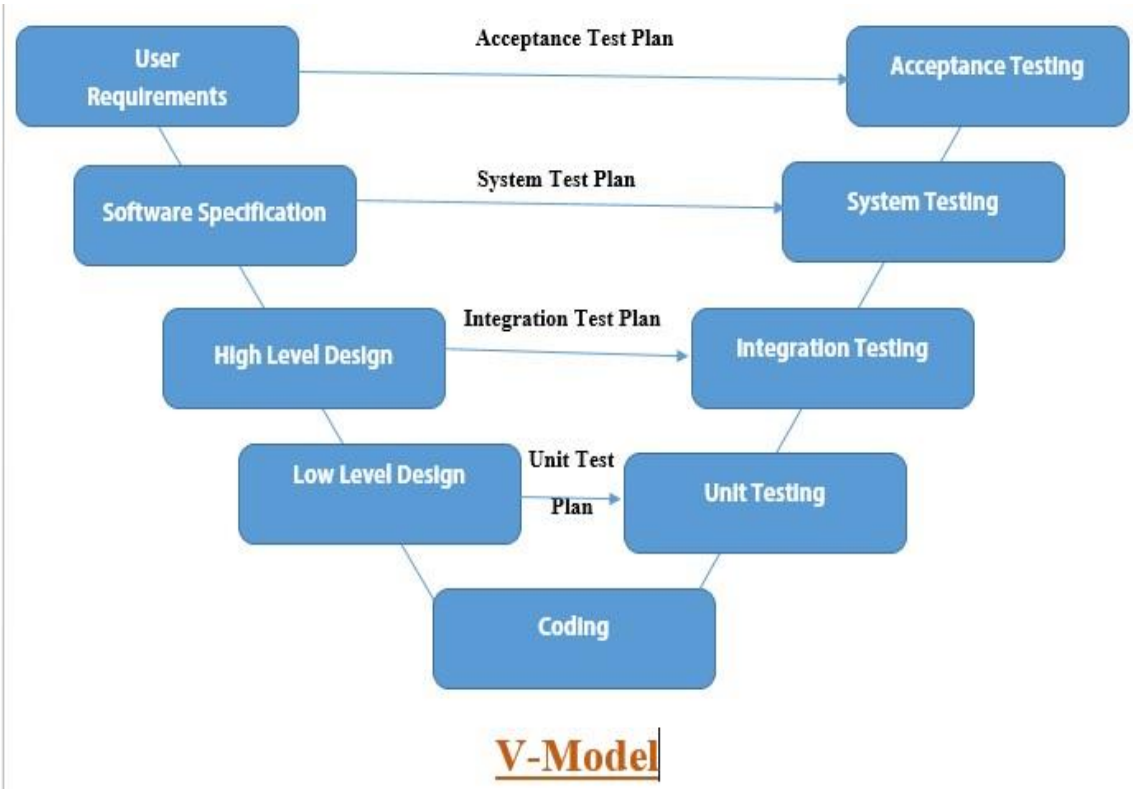
Fig. 4 Iterative Development.

V-Shaped Model: Just like the waterfall model, the V-Shaped life cycle is a sequential path of execution of processes. Each phase must be completed before the next phase begins. Testing is emphasized in this model more than the waterfall model. The testing procedures are developed early in the life cycle before any coding is done, during each of the phases preceding implementation. Requirements begin the life cycle model just like the waterfall model. Before development is started, a system test plan is created. The test plan focuses on meeting the functionality specified in requirements gathering. The high-level design phase focuses on system architecture and design. An integration test plan is created in this phase in order to test the pieces of the software systems ability to work together. However, the low-level design phase lies where the actual software components are designed, and unit tests are created in this phase as well. The implementation phase is, again, where all coding takes place. Once coding is complete, the path of execution continues up the right side of the V where the test plans developed earlier are now put to use.

Advantages

1. Simple and easy to use.
2. Each phase has specific deliverables.

3. Higher chance of success over the waterfall model due to the early development of test plans during the life cycle.
4. Works well for small projects where requirements are easily understood.



Disadvantages

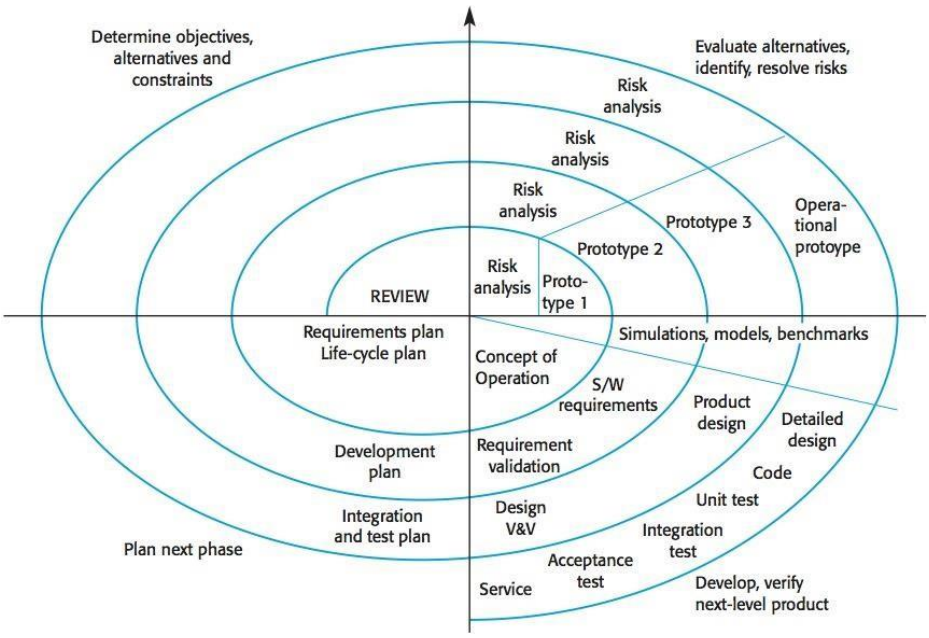
1. Very rigid like the waterfall model.
2. Little flexibility and adjusting scope is difficult and expensive.
3. Software is developed during the implementation phase, so no early prototypes of the software are produced.
4. This Model does not provide a clear path for problems found during testing phases.

Spiral Model The spiral model is similar to the incremental model, with more emphases 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 builds on the baseline spiral. Requirements are gathered during the planning phase. 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. Software is produced in the engineering phase, along with testing at the end of the phase. The evaluation phase

allows the customer to evaluate the output of the project to date before the project continues to the next spiral. In the spiral model, the angular component represents progress, and the radius of the spiral represents cost.

• Advantages
1. High amount of risk analysis.
2. Good for large and mission-critical projects.
3. Software is produced early in the software life cycle.

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



In Summary, spiral consists of:

Planning: Define objectives, constraints, and deliverables
Risk: analysis Identify risks and develop acceptable resolutions
Engineering: Develop a prototype that includes all deliverables
Evaluation: Perform assessment and testing to develop objectives for next iteration

Extreme Programming: An approach to development, based on the development and delivery of very small increments of functionality. It relies on constant code improvement, user involvement in the development team and pair wise programming. It can be difficult to keep the interest of customers who are involved in the process. Team members may be unsuited to the intense involvement that characterizes agile methods. Prioritizing changes can be difficult where there are multiple stakeholders. Maintaining simplicity requires extra work. Contracts may be a problem as with other approaches to iterative development.

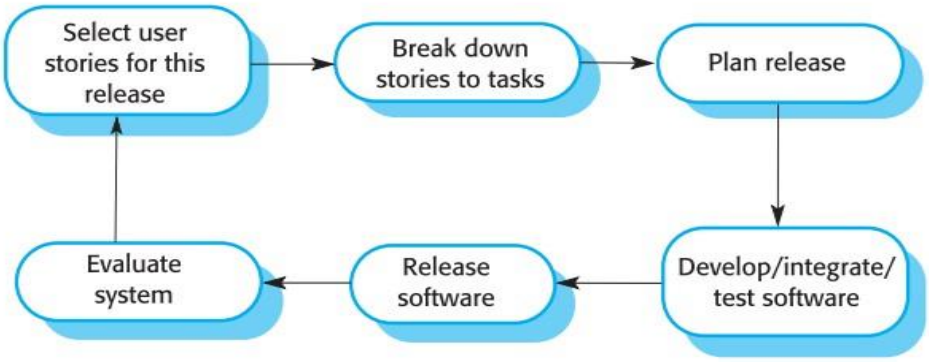


Fig. 8 The XP Release Cycle

Extreme Programming Practices Incremental planning: Requirements are recorded on Story Cards and the Stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development "Tasks".

Small Releases: The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.

Simple Design: Enough design is carried out to meet the current requirements and no more.

Test first development: An automated unit test framework is used to write tests for a new piece of functionality before functionality itself is implemented. Refactoring: All developers are expected to re-factor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.

Pair Programming: Developers work in pairs, checking each other's work and providing support to do a good job.

Collective Ownership: The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers own all the code. Anyone can change anything.

Continuous Integration: As soon as work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.

Sustainable pace: Large amounts of over-time are not considered acceptable as the net effect is often to reduce code quality and medium term productivity.

On-site Customer: A representative of the end-user of the system (the Customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

XP and agile principles

1. Incremental development is supported through small, frequent system releases.
2. Customer involvement means full-time customer engagement with the team.
3. People not process through pair programming, collective ownership and a process that avoids long working hours.
4. Change supported through regular system releases.
5. Maintaining simplicity through constant refactoring of code

• Advantages

1. Lightweight methods suit small-medium size projects.
2. Produces good team cohesion.
3. Emphasises final product.
4. Iterative.
5. Test based approach to requirements and quality assurance.

• Disadvantages

1. Difficult to scale up to large projects where documentation is essential.
2. Needs experience and skill if not to degenerate into code-and-fix.
3. Programming pairs is costly.
4. Test case construction is a difficult and specialized skill.

DIFFERENT APPROACHES TO IMPROVING DEVELOPMENT

In the continuing effort to improve the systems analysis and design process, several different approaches have been developed. Attempts to make systems development less of an art and more of a science are usually referred to as *systems engineering* or *software engineering*. As the names indicate, rigorous engineering techniques have been applied to systems development. One manifestation of the engineering approach is CASE tools.

CASE Tools

Other efforts to improve the systems development process have taken advantage of the benefits offered by computing technology itself. The result has been the creation

and fairly widespread use of **computer-aided software engineering (CASE) tools**. CASE tools have been developed for internal use and for sale by several leading firms, but the best known is the series of Rational tools made by IBM.

CASE tools are used to support a wide variety of SDLC activities. CASE tools can be used to help in multiple phases of the SDLC: project identification and selection, project initiation and planning, analysis, design, and implementation and maintenance. An integrated and standard database called a *repository* is the common method for providing product and tool integration, and has been a key factor in enabling CASE to more easily manage larger, more complex projects and to seamlessly integrate data across various tools and products. The idea of a central repository of information about a project is not new—the manual form of such a repository is called a *project dictionary* or *workbook*.

The general types of CASE tools are listed below:

- Diagramming tools enable system process, data, and control structures to be represented graphically.
- Computer display and report generators help prototype how systems “look and feel.” Display (or form) and report generators make it easier for the systems analyst to identify data requirements and relationships.
- Analysis tools automatically check for incomplete, inconsistent, or incorrect specifications in diagrams, forms, and reports.
- A central repository enables the integrated storage of specifications, diagrams, reports, and project management information.
- Documentation generators produce technical and user documentation in standard formats.
- Code generators enable the automatic generation of program and database definition code directly from the design documents, diagrams, forms, and reports.

SDLC Phase	Key Activities	CASE Tool Usage
Project identification and selection	Display and structure high-level organizational information	Diagramming and matrix tools to create and structure information
Project initiation and planning	Develop project scope and feasibility	Repository and documentation generators to develop project plans
Analysis	Determine and structure system requirements	Diagramming to create process, logic, and data models
Logical and physical design	Create new system designs	Form and report generators to prototype designs; analysis and documentation generators to define specifications
Implementation	Translate designs into an information system	Code generators and analysis, form and report generators to develop system; documentation generators to develop system and user documentation
Maintenance	Evolve information system	All tools are used (repeat life cycle)

Table 1: EXAMPLES OF CASE USAGE WITHIN THE SDLC

AGILE METHODOLOGIES

Many approaches to systems analysis and design have been developed over the years. In February 2001, many of the proponents of these alternative approaches met in Utah and reached a consensus on several of the underlying principles their various approaches contained. This consensus turned into a document they called “The Agile Manifesto”.

According to Fowler (2003), the Agile Methodologies share three key principles:
(1) a focus on adaptive rather than predictive methodologies, (2)
a focus on people rather than roles, and
(3) a focus on self-adaptive processes.

The Agile Methodologies group argues that software development methodologies adapted from engineering generally do not fit with real-world software development (Fowler, 2003). In engineering disciplines, such as civil engineering, requirements tend to be well understood. Once the creative and difficult work of design is completed, construction becomes very predictable. In addition, construction may account for as much as 90 percent of the total project effort. For software, on the other hand, requirements are rarely well understood, and they change continually during the lifetime of the project. Construction may account for as little as 15 percent of the total project effort, with design

constituting as much as 50 percent. Applying techniques that work well for predictable, stable projects, such as bridge building, tend not to work well for fluid, design-heavy projects such as writing software, say the Agile Methodology proponents. What is needed are methodologies that embrace change and that are able to deal with a lack of predictability. One mechanism for dealing with a lack of predictability, which all Agile Methodologies share, is iterative development (Martin, 1999). Iterative development focuses on the frequent production of working versions of a system that have a subset of the total number of required features. Iterative development provides feedback to customers and developers alike.

The Agile Methodologies' focus on people is an emphasis on individuals rather than on the roles that people perform (Fowler, 2003). The roles that people fill, of systems analyst or tester or manager, are not as important as the individuals who fill those roles. Fowler argues that the application of engineering principles to systems development has resulted in a view of people as interchangeable units instead of a view of people as talented individuals, each bringing something unique to the development team. The Agile Methodologies promote a self-adaptive software development process. As software is developed, the process used to develop it should be refined and improved. Development teams can do this through a review process, often associated with the completion of iterations. The implication is that, as processes are adapted, one would not expect to find a single monolithic methodology within a given corporation or enterprise. Instead, one would find many variations of the methodology, each of which reflects the particular talents and experience of the team using it.

Agile Methodologies are not for every project. Fowler (2003) recommends an agile or adaptive process if your project involves

- unpredictable or dynamic requirements,
- responsible and motivated developers, and
- customers who understand the process and will get involved.

The Manifesto for Agile Software Development, *Seventeen anarchists agree:*
We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Responding to change over following a plan. That is, while we value the items on the right, we value the items on the left more.

We follow the following principles:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Businesspeople and developers work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Continuous attention to technical excellence and good design enhances agility.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Simplicity—the art of maximizing the amount of work not done—is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

[Note: The name of seventeen change maker]: Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C.
Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas (*www.agileAlliance.org*)

TABLE 1-4 Five Critical Factors That Distinguish Agile and Traditional Approaches to Systems Development

Factor	Agile Methods	Traditional Methods
Size	Well matched to small products and teams. Reliance on tacit knowledge limits scalability.	Methods evolved to handle large products and teams. Hard to tailor down to small projects.
Criticality	Untested on safety-critical products. Potential difficulties with simple design and lack of documentation.	Methods evolved to handle highly critical products. Hard to tailor down to products that are not critical.
Dynamism	Simple design and continuous refactoring are excellent for highly dynamic environments but a source of potentially expensive rework for highly stable environments.	Detailed plans and Big Design Up Front, excellent for highly stable environment but a source of expensive rework for highly dynamic environments.
Personnel	Requires continuous presence of a critical mass of scarce experts. Risky to use non-agile people.	Needs a critical mass of scarce experts during project definition but can work with fewer later in the project, unless the environment is highly dynamic.
Culture	Thrives in a culture where people feel comfortable and empowered by having many degrees of freedom (thriving on chaos).	Thrives in a culture where people feel comfortable and empowered by having their roles defined by clear practices and procedures (thriving on order).

Many different individual methodologies come under the umbrella of Agile Methodologies. Fowler (2003) lists the Crystal family of methodologies, Adaptive Software Development, Scrum, Feature Driven Development, and others as Agile Methodologies. Perhaps the best known of these methodologies, however, is eXtreme Programming.

Capability Maturity Model (CMM)

The Capability Maturity Model (CMM) is a methodology used to develop and refine an organization's software development process. The model describes a five-level evolutionary path of increasingly organized and systematically more mature processes. CMM was developed and is promoted by the Software Engineering Institute (SEI), a research and development center sponsored by the U.S. Department of Defense (DoD). SEI was founded in 1984 to address software engineering issues and, in a broad sense, to advance software engineering methodologies. More specifically, SEI was established to optimize the process of developing, acquiring, and maintaining heavily software-reliant systems for the DoD. Because the processes involved are equally applicable to the software industry as a whole, SEI advocates industry-wide adoption of the CMM.

- At the *initial* level, processes are disorganized, even chaotic. Success is likely to depend on individual efforts, and is not considered to be repeatable, because processes would not be sufficiently defined and documented to allow them to be replicated.
- At the *repeatable* level, basic project management techniques are established, and successes could be repeated, because the requisite processes would have been made established, defined, and documented.
- At the *defined* level, an organization has developed its own standard software process through greater attention to documentation, standardization, and integration.
- At the *managed* level, an organization monitors and controls its own processes through data collection and analysis.
- At the *optimizing* level, processes are constantly being improved through monitoring feedback from current processes and introducing innovative processes to better serve the organization's particular needs.

Alternate development methodologies

Development Strategies

Analysts and organisations are increasingly faced with a make, buy or outsource decision when assessing development strategies for information systems project. As an analyst, part of the expertise you are developing is to make sound judgments regarding developing software versus purchase of software or outsourcing for new and existing system. Each alternative comes with its own strengths and weaknesses. Therefore, one alternative is better than others in different situations.

Create Custom Software

Custom Software is also known as in-house software, is a type of software that is developed from the beginning for a specific organisation or function. Most of the project groups consider custom development as the best approach to develop a system because the team has complete control over the functions and shape of the system.

There are several situations that call for the custom development of a new software or software components. The most likely instance is when commercial off-the-shelf (COTS) software does not exist or cannot be identified for desired application. Alternatively, the software may exist but is unaffordable or cannot easily be purchased or licensed.

Custom software development should be done when the organisation is attempting to gain a competitive advantage through the leveraged use of information systems. This is often the case when an organisation is creating ecommerce or other innovative applications where none existed. It is also possible that the organisation is a “first pioneer”

in the use of a particular technology or in its particular industry. Organisations that have highly specialised requirements or exist in niche industries can also benefit from custom development. Given Table summarises the advantages and disadvantages of custom development.

Advantages	Disadvantages
• Specific response to specialised business needs. • Innovation may give firm a competitive advantage. • In-house staff available to maintain software. • Pride of ownership.	• May be significantly higher initial cost compared to COTS software or Application Service Provider. • Necessity of hiring or working with a development team. • Ongoing maintenance.

Purchasing COTS Packages

Commercial Off-The-Shelf (COTS) software is a ready-made software product that can be easily obtained. COTS include such products as Microsoft Office suite which includes Word for word processing, Excel for spreadsheets, Access for building databases and other applications. Other types of COTS software from Enterprise Resource Planning (ERP) packages such as Oracle and SAP.

Consider using COTS software when you can easily integrate the applications or packages into existing or planned systems, and when you have identified no necessity to immediately or continuously change or customise them for users. There are some advantages to purchasing COTS software that you should keep in mind as you weigh alternative. One advantage is that these products have been refined through the process of commercial use and distribution, so that often there are additional functionalities offered. Another advantage is that packaged software is typically extensively tested, and thus extremely reliable. Table given below summarises the advantages and disadvantages of purchasing COTS packages

Most of the COTS packages allow customisation or manipulation of system parameters for changing certain functions. These changes are good in creating functions that are needed, but the said functions might not be present inside the software package.

System Integration refers to the process of developing a new system by integrating or combining a software package with an existing legacy system. Many companies are skilful in system integration. It is not impossible for a company to select COTS packages,and then to outsource the job of integrating various packages to the consulting companies.

Outsourcing is contracting goods or services with another company or person to do a particular function such as to pay an external supplier to develop or to provide services for creating a system. This third option is to outsource some of the organisation's software needs to an Application Service Provider (ASP) that specialises in IT applications. Almost every organisation outsources in some way. Typically, the function being outsourced is considered non-core to the business.

There are specific benefits to outsourcing applications to an ASP. For example organisations that desire to retain their strategic focus and do what they are best at, may want to outsource the production of information systems applications. Additionally outsourcing one's software needs means that the organisation doing the outsourcingmay be able to sidestep the need to hire, train and retain a large IT staff.

- Will the new system result in a workforce reduction? If so, what will happen to affected employees?

Advantages	Disadvantages
• Refined in the commercial world. • Increased reliability. • Increased functionality. • Often lower initial cost. • Already in use by other firms. • Help and training comes with software.	• Programming focused; not business focused. • Must live with the existing features. • Limited customisation. • Uncertain financial future of vendor. • Less ownership and commitment.

- Will the new system require training for users? If so, is the company prepared to provide the necessary resources for training current employees?
- Will users be involved in planning the new system right from the start?
- Will the new system place any new demands on users or require any operating changes? For example, will any information be less accessible or produced less

frequently? Will performance decline in any way? If so, will an overall gain to the organization outweigh individual losses?

- Will customers experience adverse effects in any way, either temporarily or permanently?
- Will any risk to the company's image or goodwill result?
- Does the development schedule conflict with other company priorities?
- Do legal or ethical issues need to be considered?

UNIT II

FEASIBILITY ANALYSIS AND THE SYSTEM PROPOSAL

Feasibility Study

A Feasibility Study which is conducted by the project manager involves determining if the information system makes sense for the organisation from an economic and operational standpoint. The study takes place before the system is constructed.

Technical Feasibility

Technical feasibility refers to the technical resources needed to develop, purchase, install, or operate the system. When assessing technical feasibility, an analyst must consider the following points:

a) Does the company have the necessary hardware, software, and network resources? If not, can those resources be acquired without difficulty?
b) Does the company have the needed technical expertise? If not, can it be acquired?
c) Does the proposed platform have sufficient capacity for future needs? If not, can it be expanded?
d) Will a prototype be required?
e) Will the hardware and software environment be reliable? Will it integrate with other company information systems, both now and in the future? Will it interface properly with external systems operated by customers and suppliers?
f) Will the combination of hardware and software supply adequate performance? Do clear expectations and performance specifications exist?
g) Will the system be able to handle future transaction volume and company growth?

Economic Feasibility

Economic feasibility means that the projected benefits of the proposed system outweigh the estimated costs usually considered the **total cost of ownership** (**TCO**), which includes ongoing support and maintenance costs, as well as acquisition costs. To determine TCO, the analyst must estimate costs in each of the following areas:

• People, including IT staff and users
• Hardware and equipment

• Software, including in-house development as well as purchases from vendors
• Formal and informal training
• Licenses and fees
• Consulting expenses
• Facility costs

• The estimated cost of not developing the system or postponing the project
In addition to costs, you need to assess tangible and intangible benefits to the company. The systems review committee will use those figures, along with your cost estimates, to decide whether to pursue the project beyond the preliminary investigation phase.

Tangible benefits are benefits that can be measured in dollars. Tangible benefits result from a decrease in expenses, an increase in revenues, or both. Examples of tangible benefits include the following:

a) A new scheduling system that reduces overtime
b) An online package tracking system that improves service and decreases the need for clerical staff
c) A sophisticated inventory control system that cuts excess inventory and eliminates production delays

Intangible benefits are advantages that are difficult to measure in dollars but are important to the company. Examples of intangible benefits include the following: a) A user-friendly system that improves employee job satisfaction

b) A sales tracking system that supplies better information for marketing decisions
c) A new Web site that enhances the company's image

You also must consider the development timetable, because some benefits might occur as soon as the system is operational, but others might not take place until later.

Schedule Feasibility

Schedule feasibility means that a project can be implemented in an acceptable time frame. When assessing schedule feasibility, a systems analyst must consider the interaction between time and costs. For example, speeding up a project schedule might make a project feasible, but much more expensive.

Other issues that relate to schedule feasibility include the following:

• Can the company or the IT team control the factors that affect schedule feasibility?
• Has management established a firm timetable for the project?
• What conditions must be satisfied during the development of the system? • Will an accelerated schedule pose any risks? If so, are the risks acceptable?
• Will project management techniques be available to coordinate and control the project?
• Will a project manager be appointed?

6.2.1 Schedule Feasibility

Schedule feasibility measure of how reasonable a project timetable is. We ask the technical expertise, is the project can be completed within the deadlines? During the schedule feasibility analysis, it's important to define the deadlines for every phase if there is any.

6.2.2 Cultural Feasibility

Cultural feasibility also refers as political feasibility. This is related with operational feasibility. But, the operational feasibility deals more on how well the solution meets the system requirement but the cultural feasibility deals with how the end users feel about the proposed system. We can say that operational feasibility concerns with whether the system can work in solving the problem and cultural feasibility concerns with whether the system can be adapted in the organizational environment.

The following questions can help the team when doing cultural feasibility :

- Does the management support the system?
- How do the end users feel about their role in the proposed system?
- What end users or managers may resist or not use the system?
- How will the working environment of the end users can change with the proposed system?

6.2.3 Legal Feasibility

Information system has a legal impact. Legal feasibility is a measure of how well a solution can be implemented within existing legal and organization's policy. It's also regarding copyright issues. For example, if we need certain software in develop the system; we have to make sure that we use the licensed software.

COST-BENEFITS ANALYSIS TECHNIQUES

Economic feasibility has been defined as a cost benefit analysis. It's a way to estimate the benefits and costs in the system development. In order to determine this, we need to do a comparison between this.

Costs

An information system can have tangible costs and intangible costs. Tangible costs refer to items that you can easily measure in terms of money and with certainty. During the system development, tangible cost includes hardware costs, labor costs, and others.

Intangible costs refers to cost derived from the system, but it can't be measured in terms of money and with certainty. Examples of intangible costs are loss of customer goodwill, and employee morale.

In system development life cycle, there are two types of costs; costs of developing the system and costs of operating the system. The cost of developing the system refers to one time costs which it will not recur after the project has been completed. Several categories of costs that need to be considered such as :

- Personnel costs – salaries for the person involved such as system analyst, programmer, system designer, data entry personnel, manager and others.
- Computer usage – computer time for the activities such as programming, installation, data loading, storage and others.
- Training – the cost of training that will be provided for users who will use the system.
- Supply, duplication and equipment costs
- New hardware and software costs

There are two types of costs, fixed cost and variable cost. Fixed cost is a cost that occurs at a regular interval and at a relatively fixed rate such as salaries. Variable cost is a cost that occurs in proportion to some usage factors such as printer toner, paper and others.

Benefits

Similar with costs, an information system can provide a lot of benefits to an organization. It can be divided as tangible benefits and intangible benefits. A tangible benefit refers to benefits that can be measured in terms of money and with certainty. An example of tangible benefits are reduces the cost, increase the profits. An intangible benefit refers to benefits derived from the system, but it can't be measured in terms of money and with certainty. An example of intangible benefits are competitive necessity, improved the organizational reputation, faster decision making. Normally, benefits can increase the organizations' profit and can decrease the costs, either in short term or long term.

6.3 TECHNIQUES FOR ASSESSING ECONOMIC FEASIBILITY

There are three popular techniques can be used for assessing the economic feasibility; payback analysis, return on investment and net present value. We will discuss these three techniques in this following section.

6.3.1 Payback Analysis

Payback analysis technique is a popular technique for determining if and when an investment will pay for itself. As mentioned earlier, system development needs more costs during the earlier phases we need more time to get the benefits to overtake the costs. At the early phases of system development, most of the costs spend on analysis, design and implementation. After implementation, the cost is needed for the operating expenses that must be recovered. Payback analysis determines how much time needed before benefits overtake the costs needed. This period is refer payback period. Figure 6-1 shows that before entered the year three, the benefits can overtake the costs spend.

R

	A	B	C	D	E	F
1	**Cash Flow Description**	**Year 1**	**Year 2**	**Year 3**	**Year 4**	
2	Development Cost	-9500				
3	Operation and Maintenance Cost	-2000	-7500	-9000	-8000	
4	Time-adjusted costs	-5500	-5000	-4500	-5000	
5	Cumulative time-adjusted costs	*-17000*	*-12500*	*-13500*	*-13000*	
6						
7	Benefits from using a new system	2500	5000	7000	12000	
8	Time-adjusted benefits	3000	4500	7000	10000	
9	Cumulative time-adjusted benefits	*5500*	*9500*	*14000*	*22000*	
10						
11	**Cumulative time adjusted Cost+Benefits**	**-11500**	**-3000**	**500**	**9000**	

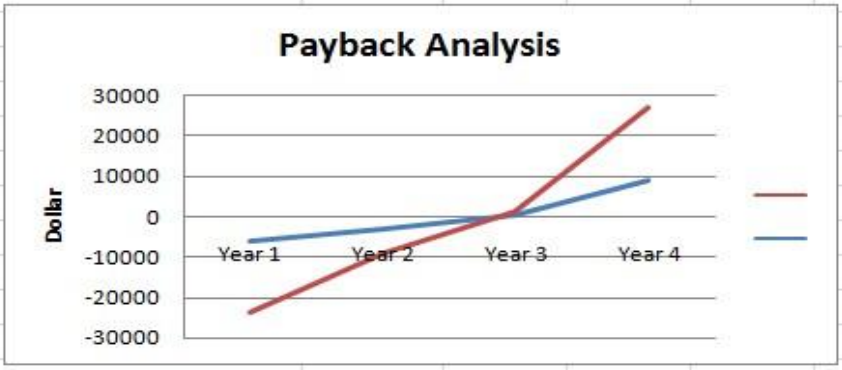


Figure 6-1: Payback Analysis

6.3.2 Return On Investment

Return on Investment (ROI) analysis technique compares the lifetime profitability of alternative solutions. The ROI of the solution is a percentage rate that measures the relationship between the amounts the business get back from the investment and the amount invested.

Lifetime ROI = (Estimated lifetime benefits – Estimated lifetime costs)/ Estimated lifetime costs

As for example from Figure 6-1 above:

22000 – 13000 = 9000

Therefore the lifetime ROI is

Lifetime ROI = 9000/13000 = .692 = 70%

6.3.3 Net Present Value

Net present value is a technique that compares the annual costs and benefits of alternative solutions. We define the costs and benefits for each years of the system's lifetime. Using a same example as Figure 6-1, we can determine the net present value as in Figure 6-2 below.

	A	B	C	D	E	F
1	**Cash Flow Description**	**Year 1**	**Year 2**	**Year 3**	**Year 4**	**Total**
2	Development Cost	-9500				
3	Operation and Maintenance Cost	-2000	-7500	-9000	-8000	
4	Time-adjusted costs	-5500	-5000	-4500	-5000	
5	Present value of costs	*-17000*	*-12500*	*-13500*	*-13000*	
6	Total present value of lifetime costs					*-13000*
7						
8	Benefits from using a new system	2500	5000	7000	12000	
9	Time-adjusted benefits	3000	4500	7000	10000	
10	Present value of benefits	*5500*	*9500*	*14000*	*22000*	
11	Total present value of lifetime benefits					***22000***
12						
13	**Net present value**	**-11500**	**-3000**	**500**	**9000**	**9000**
14						

Figure 6-2: Net Present Value

6.5 FEASIBILITY ANALYSIS OF CANDIDATE SYSTEM

At the third checkpoint of the feasibility analysis, the team identifies all candidate system solutions and then analyzes each of them. We do a comparison for each of the candidate to choose which candidate is the best solution to be applied. There are two can be used to make a comparison and make a recommendation. These two matrices are candidate system matrix and feasibility analysis matrix.

6.5.1 Candidate System Matrix

Candidate system matrix allows us to make a comparison between all the candidate systems that available. It is a tool used to compare the similarities and differences between candidate systems based on certain characteristic. Table 6-1 shows an example of candidate systems matrix.

Table 6-1: Candidate Systems Matrix Template

Characteristics	Candidate 1	Candidate 2	Candidate 3
Portion of the system			
Benefits			
Software needed			
Input devices			
Output devices			

Candidate Systems Matrix (Sample)

Characteristics	Candidate 1	Candidate 2	Candidate 3
Brief Description	Do nothing: continue current business processes	Purchase a PC and a DBMS that can be used to maintain an inventory database	Candidate 2 plus acquire internet service and develop a web site.
Portion of System Computerized: Brief description of the portion of system processes computerized by the	N/A	Inventory record storing and processing	Candidate 2 plus Order/customer record storing and processing, and marketing
Technology - Servers & workstations	N/A	PC	PC w/ high-speed Internet connection ISP contract
- Output device	N/A	Laser printer	Same as candidate 2
- Input device	N/A	Keyboard and mouse	Same as candidate 2
- Storage devices	N/A	PC hard disk (at least 200 GB)	Same as candidate 2
- Software tools needed	N/A	MS Access	MS Access Visual Studio 2010 (for ASP.Net)
- Application software	N/A	Designed in MS Access	Web Browser (package) ASP.Net plus MS Access
Interfaces	N/A	PC monitor (Access Interfaces)	Same as candidate 2 Plus Web pages
Processes	Same as current processes	Inventory maintenance process will be changed	Customer order processing, inventory maintenance processes will be changed
- Method of data processing	N/A	Batch	On-line/real time
Geography (Network)	N/A	N/A	N/A

6.5.2 Feasibility Analysis Matrix

Feasibility analysis matrix is similar with candidate system matrix, but the different is it comes with an analysis and ranking of the candidate systems. It has same columns as in candidate system matrix with an additional column named ranking column. Table 6-2 shows an example of feasibility analysis matrix.

Table 6-2: Feasibility Analysis Matrix Template

	Weighting	Candidate 1	Candidate 2	Candidate 3
Description				
Operational Feasibility				
Technical Feasibility				
Economic Feasibility				
Schedule Feasibility				
Legal Feasibility				
Cultural Feasibility				
Weighted score				

6.6 THE SYSTEM PROPOSAL

Notice that during this decision analysis task, it involves identifying all candidate solutions, analyzing all candidate solutions using the matrix discussed earlier, ended up with selecting the best candidate to be implemented. Next, the team will continue with the preparing the system proposal. System proposal can be in a format of a report or a formal presentation of a recommended solution. Normally, both methods; report and formal presentation will be used.

6.6.1 Report

A report about the recommended system is prepared by the team to the several categories of audiences such as top management, clerk, manager and end users. The report consists of both primary and secondary elements. Primary element is the actual information that the report is intended to convey while secondary elements is information about the

recommended system. Main basic and important element in this report is introduction, methods and procedures and conclusion. It's important to write the report in a proper way, well organized, using simple but meaningful words, so that it can deliver effectively to the target audience.

6.6.2 Formal Presentation

Formal presentation is another way in how the team presents the findings of the recommended system to solve the problem to the audience. The way how to conduct the presentation will effect the audience' confident towards the system. Effective and successful presentation requires significant preparation. Using presentation is good because the audience can respond during the session, and a good body language will convey the message which is difficult to show in written report. Visual aids can be used to support and convey the ideas.

UNIT III
Determining System Requirements

- **Requirements engineering processes • Requirements elicitation and analysis • Requirements validation • Requirements management**

Business Requirements: Business requirements represent high-level objectives of the organisation or customer requesting the system or product. (Wiegers, 1999) **COTS:** COTS (commercial off-the-shelf) is a ready-made software product, which is supplied by a vendor and it has specific functionality as part of a system (Morisio et al., 2000).

Requirement - (1) A condition or capability needed by a user to solve a problem or achieve an objective.

- (2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.
- (3) A documented representation of a condition or capability as in (1) or (2). (IEEE Std 610.12.1990)

Requirements Engineering (RE) : Activities that cover discovering, analysing, documenting and maintaining a set of requirements for a system. (Sommerville & Sawyer, 1997)

Requirements Elicitation: Elicitation is one of the first phases in requirements engineering and purpose is to discover requirements for the system being developed.

Requirements are elicited from customers, end-users and other stakeholders such as system developers. (Kotonya & Sommerville, 1998)

Requirements analysis: Analysis is one of the first phases in requirements engineering and purpose is to analyse the elicited requirements. When requirements are discovered, conflicts, overlaps, omissions and inconsistencies should be analysed. (Kotonya & Sommerville, 1998; Sommerville & Sawyer, 1997)

Requirements Validation : Requirements validation is concerned with the process of examining the requirements document to ensure that it defines the right system (i.e. the system that the user expects). (Kotonya & Sommerville, 1998)

Requirements Verification: The process of ensuring, that requirements statements are accurate, complete and that they demonstrate the desired quality characteristics. (Wiegers, 1999)

Specification A document that fully describes a design element or its interfaces in terms of requirements (functional, performance, constraints, and design characteristics) and the qualification conditions and procedures for each requirement. (IEEE Std 1220.1998)

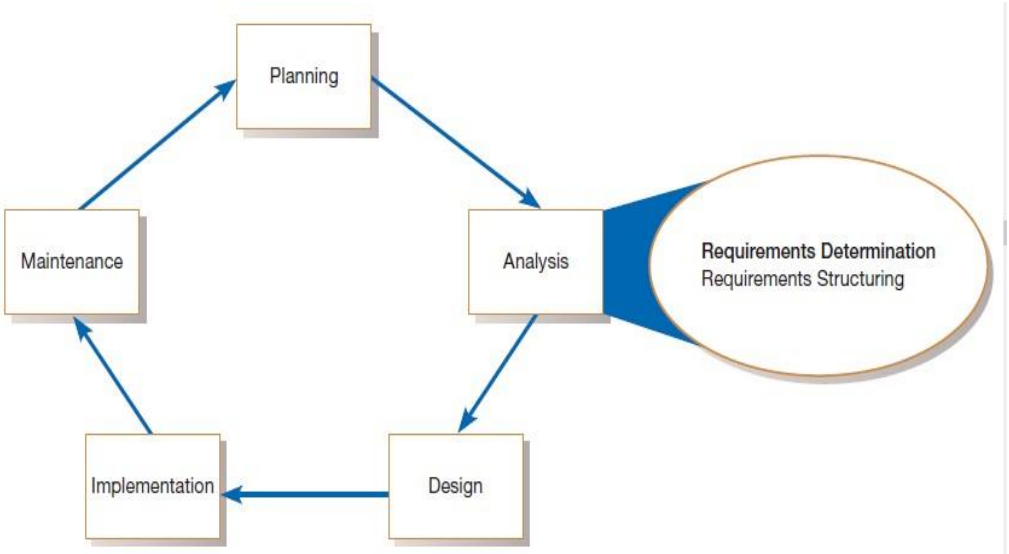


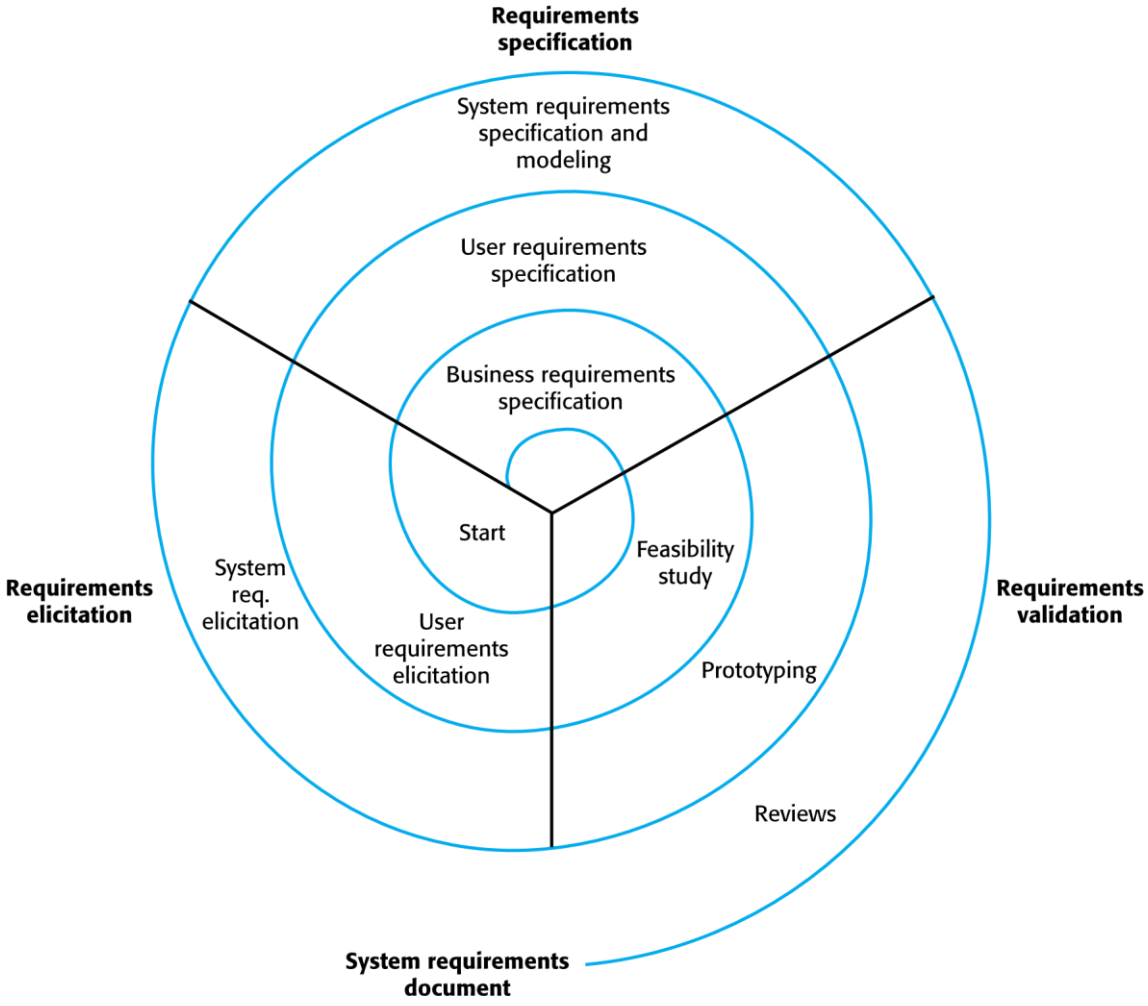
FIGURE 6-1
Systems development life cycle with analysis phase highlighted.

Requirements Engineering (RE) is a set of activities concerned with identifying and communicating the purpose of a software-intensive system, and the contexts in which it will be used. Hence, RE acts as the bridge between the real-world needs of users, customers, and other constituencies affected by a software system, and the capabilities and opportunities afforded by software-intensive technologies

Requirements engineering processes

- The processes used for RE vary widely depending on the application domain, the people involved and the organisation developing the requirements.
- However, there are a number of generic activities common to all processes
  - Requirements elicitation;
  - Requirements analysis;
  - Requirements validation;
  - Requirements management.
- In practice, RE is an iterative activity in which these processes are interleaved.

A spiral view of the requirements engineering process



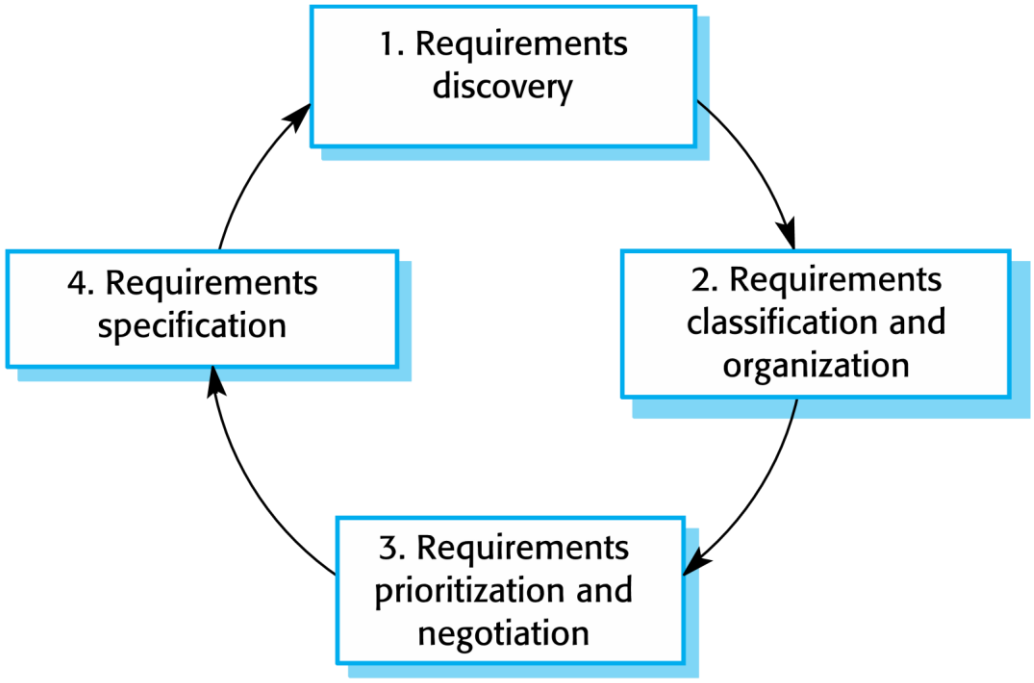
Requirements elicitation and analysis

- Sometimes called requirements elicitation or requirements discovery.
- Involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system's operational constraints.
- May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called *stakeholders.*

Requirements elicitation and analysis

- Software engineers work with a range of system stakeholders to find out about the application domain, the services that the system should provide, the required system performance, hardware constraints, other systems, etc.
- Stages include:
  - Requirements discovery,
  - Requirements classification and organization, – Requirements prioritization and negotiation, – Requirements specification.

The requirements elicitation and analysis process



Process activities

- Requirements discovery
  - Interacting with stakeholders to discover their requirements. Domain requirements are also discovered at this stage.
  - The process of gathering information about the required and existing systems and distilling the user and system requirements from this information.
  - Interaction is with system stakeholders from managers to external regulators.
  - Systems normally have a range of stakeholders.

- Requirements classification and organisation
  - Groups related requirements and organises them into coherent clusters.
- Prioritisation and negotiation

  - Prioritising requirements and resolving requirements conflicts.
- Requirements specification
  - Requirements are documented and input into the next round of the spiral.

Problems of requirements elicitation

- Stakeholders don't know what they really want.
- Stakeholders express requirements in their own terms.
- Different stakeholders may have conflicting requirements.
- Organisational and political factors may influence the system requirements.
- The requirements change during the analysis process. New stakeholders may emerge and the business environment change.

Requirements validation

- Concerned with demonstrating that the requirements define the system that the customer really wants.
- Requirements error costs are high so validation is very important
  - Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.

Requirements checking

- Validity. Does the system provide the functions which best support the customer's needs?
- Consistency. Are there any requirements conflicts?
- Completeness. Are all functions required by the customer included?
- Realism. Can the requirements be implemented given available budget and technology
- Verifiability. Can the requirements be checked?

Requirements validation techniques

- Requirements reviews
  - Systematic manual analysis of the requirements.
- Prototyping
  - Using an executable model of the system to check requirements. Covered in Chapter 2.
- Test-case generation
  - Developing tests for requirements to check testability.

Requirements reviews

- Regular reviews should be held while the requirements definition is being formulated.
- Both client and contractor staff should be involved in reviews.
- Reviews may be formal (with completed documents) or informal. Good communications between developers, customers and users can resolve problems at an early stage.

Review checks

- Verifiability
  - Is the requirement realistically testable?
- Comprehensibility
  - Is the requirement properly understood?
- Traceability
  - Is the origin of the requirement clearly stated?
- Adaptability
  - Can the requirement be changed without a large impact on other requirements?

Requirements management

- Requirements management is the process of managing changing requirements during the requirements engineering process and system development.
- New requirements emerge as a system is being developed and after it has gone into use.
- You need to keep track of individual requirements and maintain links between dependent requirements so that you can assess the impact of requirements changes. You need to establish a formal process for making change proposals and linking these to system requirements.

UNIT IV

DBMS

Database management system is abbreviated as DBMS. A DBMS is a set of computer program that control the certain maintenance and its end users. The need of such system is growing rapidly.

In summary

- A DBMS is a software to create and maintain the database.
- Enables business organization to extract data.
- Independent of specific computer program

Some example of DBMS

dBASE, faxpax,(is based on CUI (character user interface)) Sybase, Informix, Ms.Access, file maker program, oracle.

Application of DBMS:-

University:- for student information, course registration and grade Airline:-
for reservation and flight scheduling.
Finance:- for storing information data porches, sales, inventory stock and bonds.
Sales:- for customer products and porches information.
Telecommunication:- for keeping records of call made, generating bills, maintaining balance in prepaid calling cards.
HR:- maintaining information about employed payroll, tax etc..
Banking:- for customer information balance, loans and transaction.

Data model:-

Collection of tools for describing :

I. Data
II. Relationship
III. Data semantic
IV. Data constrains

Data model:-

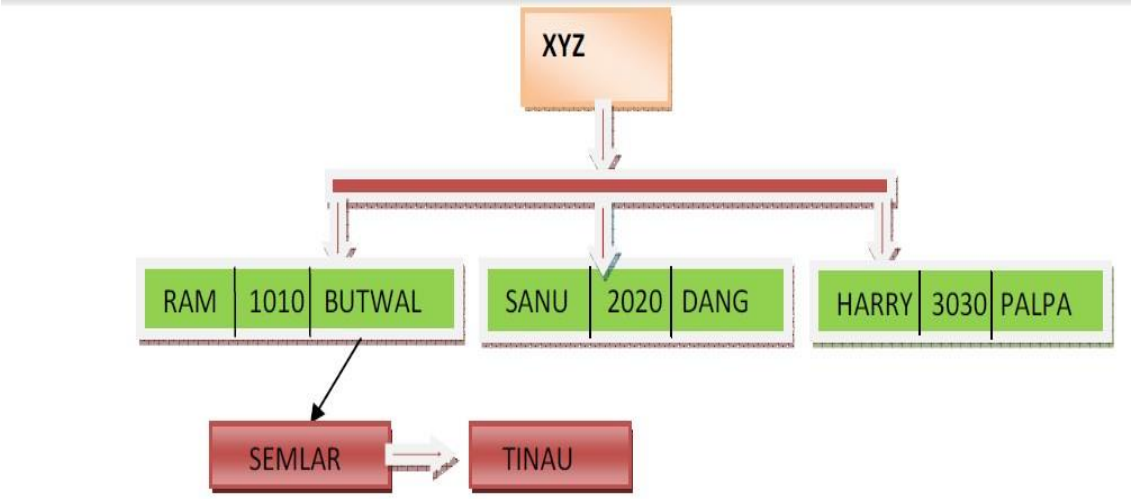
Collection of tools for describing :

I. Data
II. Relationship
III. Data semantic

IV. Data constrains

Purpose of designing:- models helps us to communicate with peoples mind. They can be used to do following a. Communication
b. Categorize
c. Describe
d. Specify

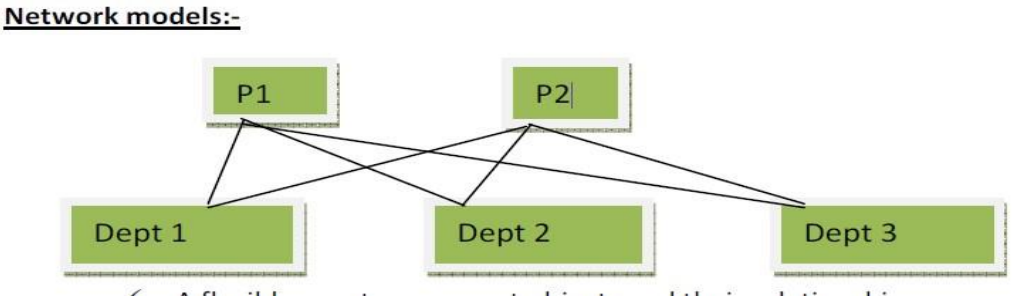
Hierarchical models :-
information are organized in the form of the like structure using parents/child relationship such as that it can but have two many relationship.
Data are represented by collection of records and data in a records have set of values attached to it.
Relationship among the records are represented by links like a pointer
Collection of all instances of specifies records together from a record type. These records are equivalent to table in a relational database and with the individuals records being equivalent to row.



Advantages of Hierarchical models:_- A.
Easiest data models.
B. Database is more secured because nobody can see and modify child database without consulting it's parents.
C. Searching faster and easy, if parents are known.
D. Efficient in handling the relationship of one to many. **Disadvantages of Hierarchical models:- A.** Traditional method.
B. Works only on one to many models .

Network models:-

- A flexible way to represent objects and their relationship.
- It allows many to many relationship among records.
- this model is similar to hierarchical database mode on sense that data and their relationship, represented by records and links.



Advantages of network model.

- Popular of mainframe and large volume of application.

Disadvantages of network model.

- Very complex database model.
- Needs complex program to handle database.
- Pointer in the database model to increase the overhead of storage. Less
- secured due to many to many relationship.

Relational database model:- Modern

- database model.
- Allows to organized the data in the form of rows and column.
- Supports only types of relationship such as 1 to many and many to many.

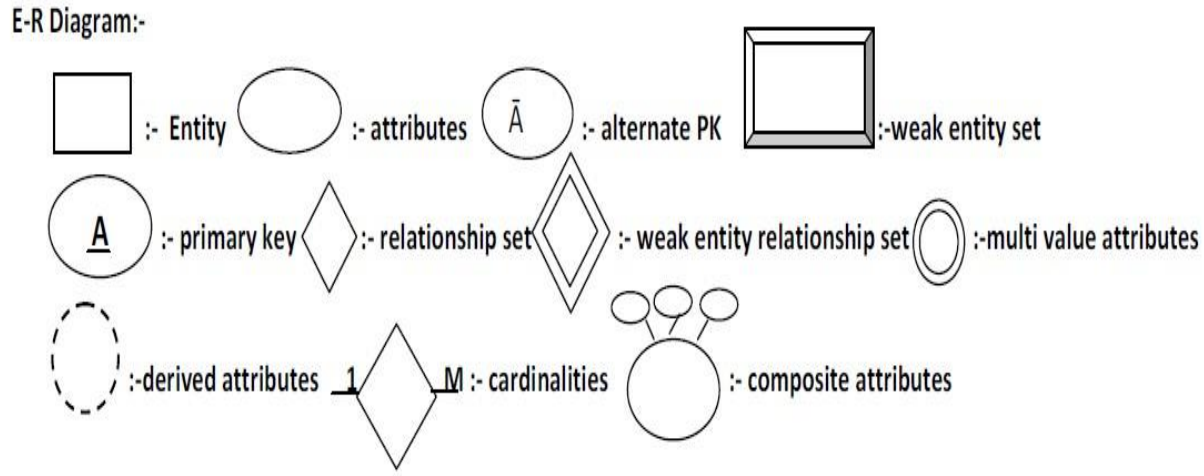
s.n.	Name	Address	Phone
1	Harry	Kathmandu	9804568236
2	Ram	Pokhara	9845896543

Entity relationship (ER) Model

- ER model is a higher level model.
- It based on the perception of real world that consist of a collection of basic object called entities and relationship among these entities.
- It consist of entities, attributes of entities and relationship among these entities.
- Helps to communicate with non-technical personnel's.

Entities:-
An entity represents an object i.e. (place, event, person) of the real world that has an independent existence. It is represented by rectangle graphically. E.g. peter, maya, ktm, us are entities that exist physically University, course, account, are entities that exist conceptually.
Relationship :-
Association between two or more than two entities.
e.g. customer, borrows loans.
Peter marry maya.
Attributes :-it is the property of an entity i.e. characteristics of the entity. It also describe about the entity. It is represented by oval entity is characterized by their attributes such as university has established data and university building has a color.

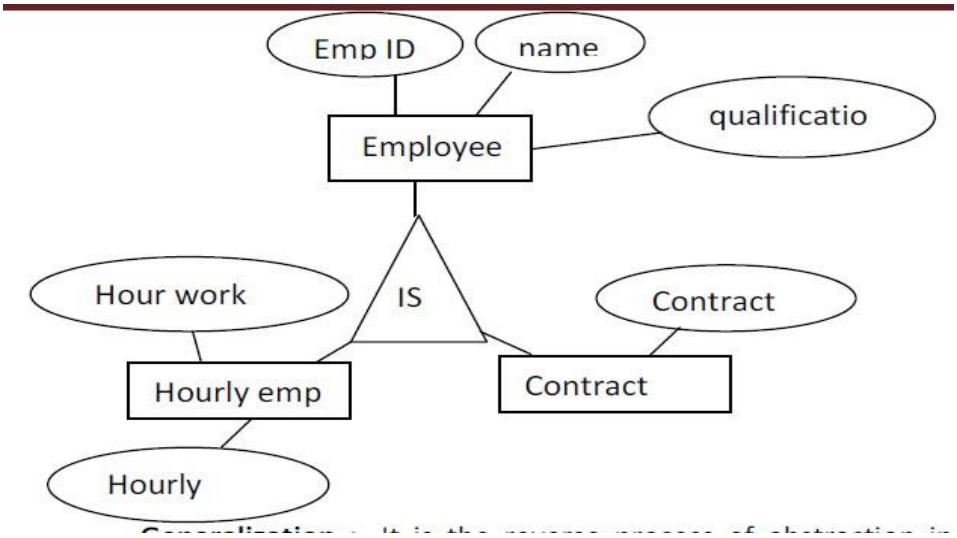
Attributes type:-
a. Simple vs composite:- simple attributes are these, which are divided into subpart e.g. emp.ID. composite attributes are these which are divided into sub-part for name and person (first name middle name, last name) address –streets district, world
b. Single value vs multiple attributes:- single value is described by one value eg. Roll no. A multi value attributes is described by many values (phone no of a person, color combination in graphic design, qualification of person).
c. Store vs derived attributes:- An attributes where value can be extracted from a value of another attributer is called derived attributes for example, age can be derived from, date of birth and current date. If this case isn't applicable then that is stored attributes.
d. Null value:- Attributes that don't have applicable values are said to be null values. It is differ from zero (ie. Unknown or missing values).



Extended ER features:-

a) Specialization (top to bottom):- It is a process of defining a set of sub classes of an entity set (ie the super class) that share some distinguishing characteristics.

Establish additional specific attributes with each sub classes, in terms of ER diagram, specialization is depicted by triangle components labeled 'ISA' the labeled ISA stand for 'is a '. ISA relationship is referred to a super class relationship.



Generalization :- It is the reverse process of abstraction in which we suppress the differences among several entities identities their common features and generalized them into a single super class of which the original entities are special sub-classes. Therefore designing top to bottom is (specialization) and designing bottom to top is a generalization (general feature).

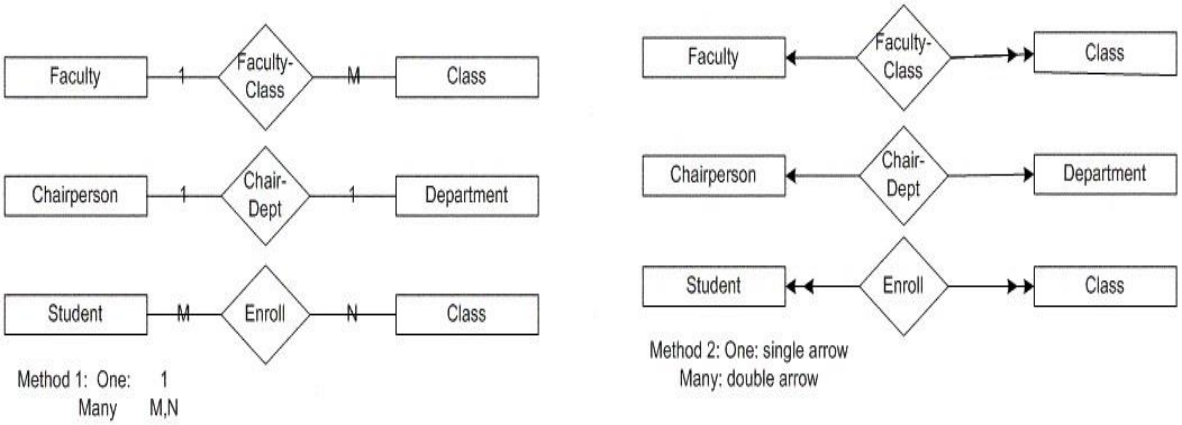
1.1 Cardinality of Relationships

Cardinality is the number of entity instances to which another entity set can map under the relationship. This does not reflect a requirement that an entity has to participate in a relationship. Participation is another concept.

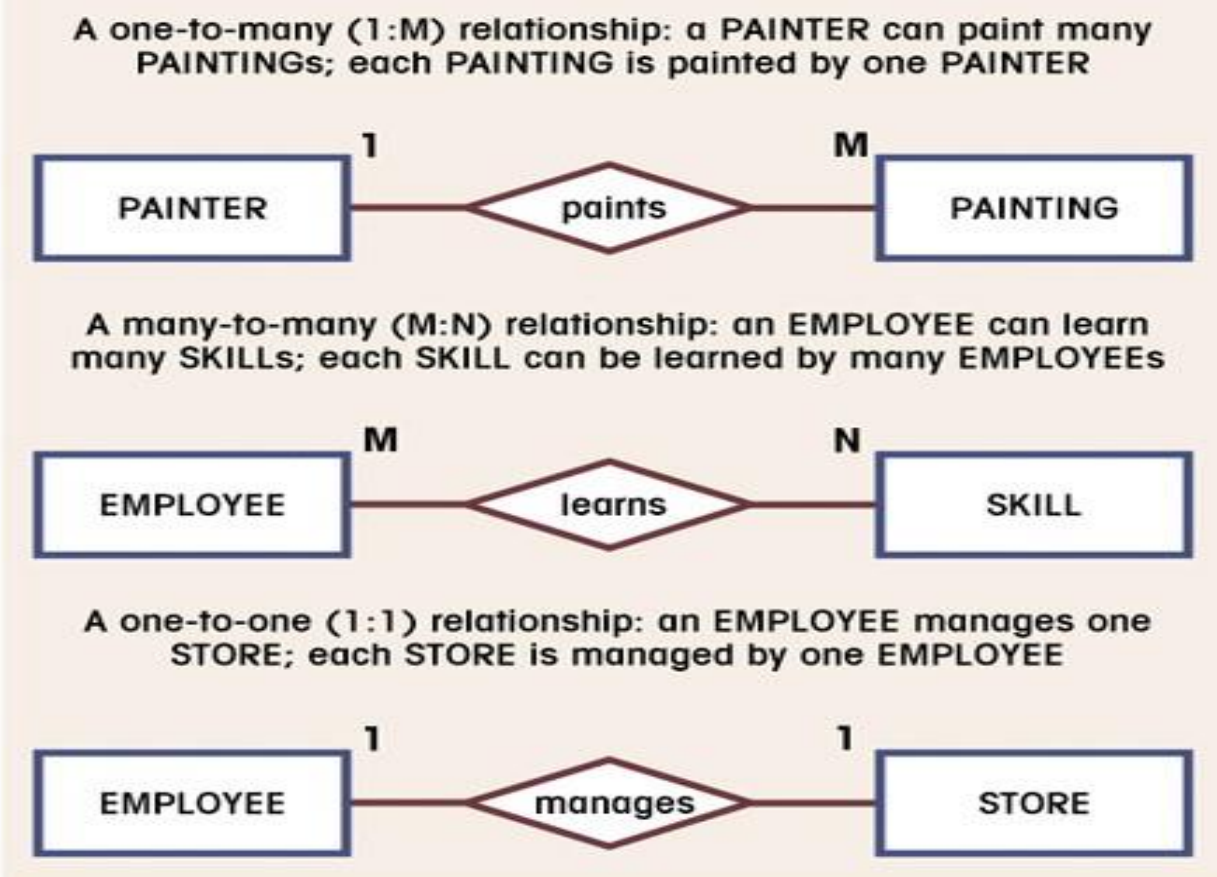
One-to-one: X-Y is 1:1 when each entity in X is associated with at most one entity in Y, and each entity in Y is associated with at most one entity in X.

One-to-many: X-Y is 1:M when each entity in X can be associated with many entities in Y, but each entity in Y is associated with at most one entity in X.

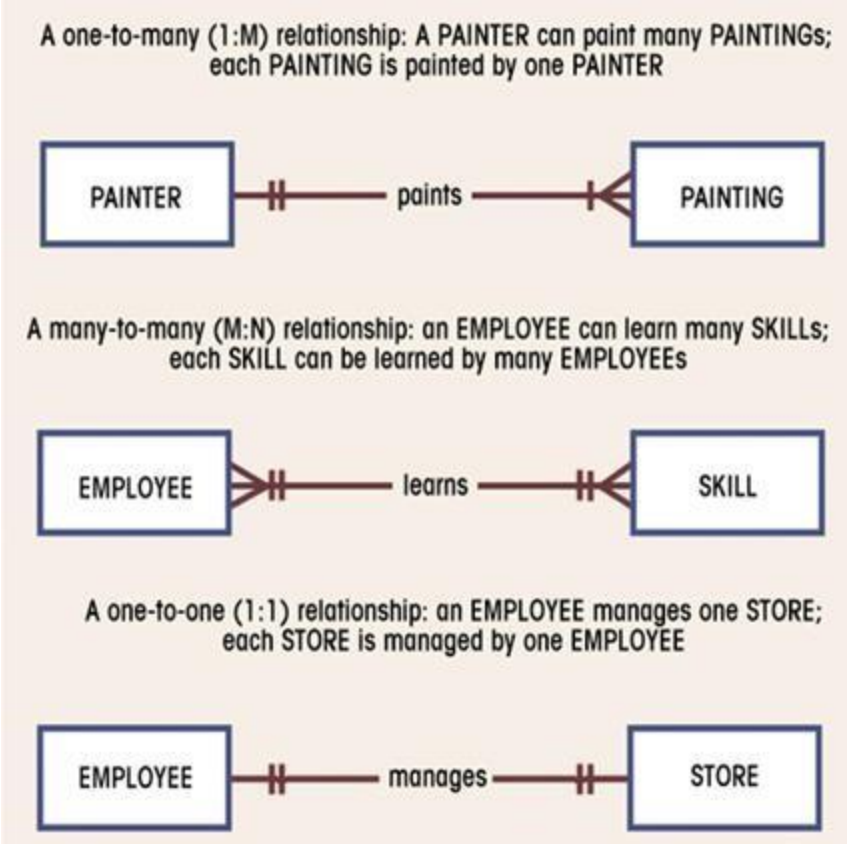
Many-to-many: X:Y is M:M if each entity in X can be associated with many entities in Y, and each entity in Y is associated with many entities in X ("many" =>one or more and sometimes zero)



E-R Diagram: Chen Model

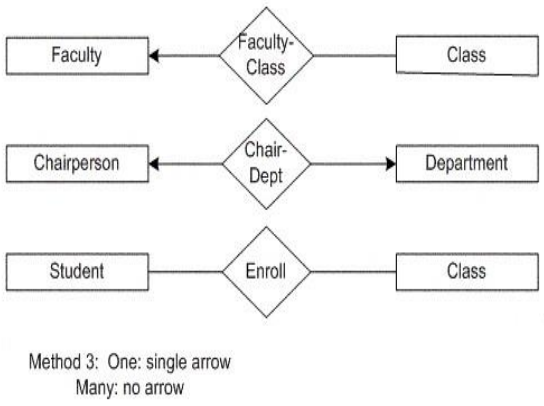


- Entity
  – represented by a rectangle with its name in capital letters.
- Relationships
  – represented by an active or passive verb inside the diamond that connects the related entities.
- Connectivities
  – i.e., types of relationship
  – written next to each entity box.

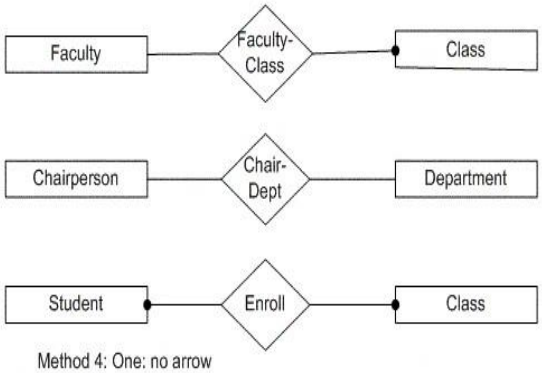


- Entity
  - represented by a rectangle with its name in capital letters.
- Relationships
  - represented by an active or passive verb that connects the related entities.
- Connectivities
  - indicated by symbols next to entities.
    - 2 vertical lines for 1
    - “crow’s foot” for M

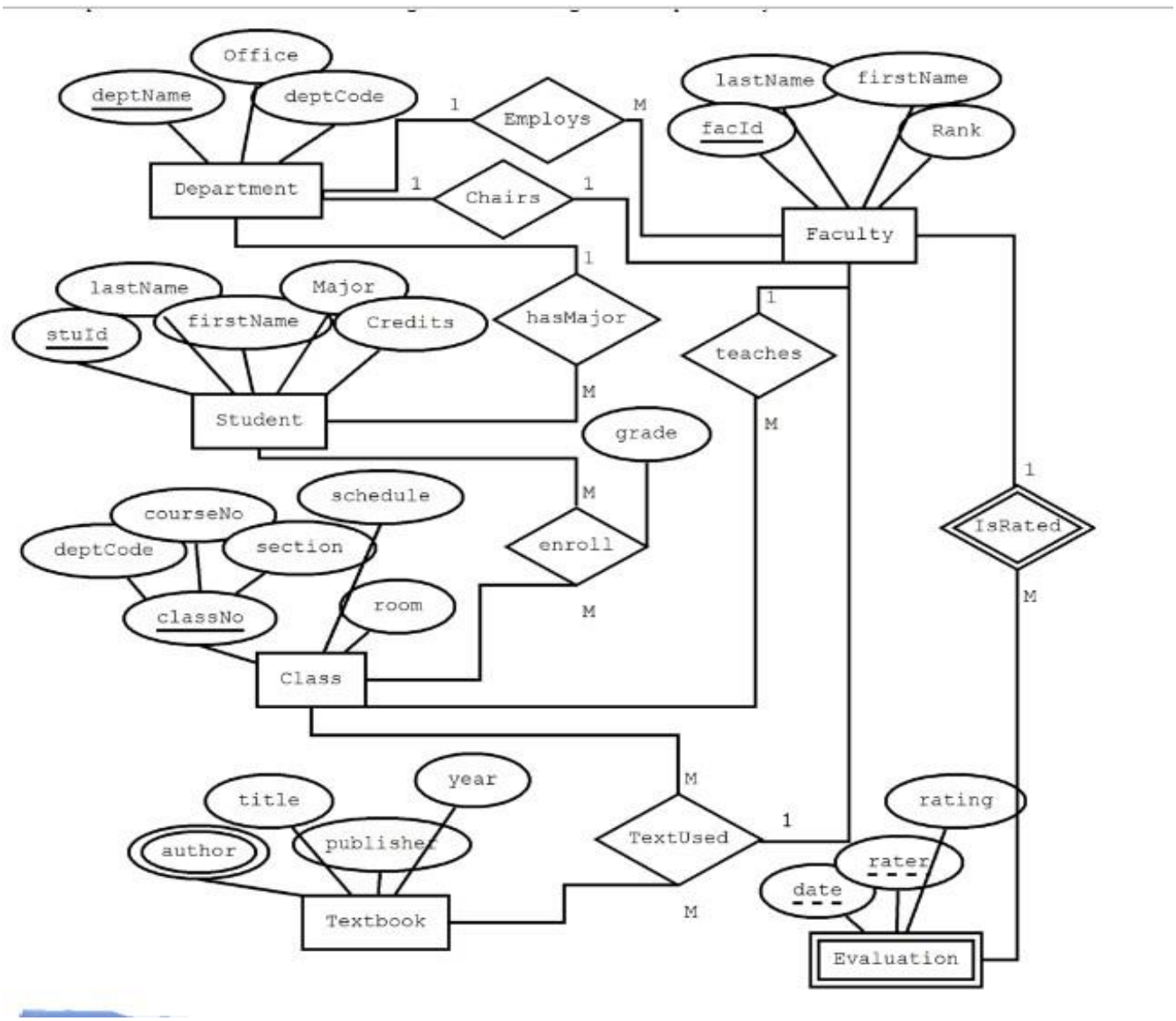
It is good to use for many part



Method 3: One: single arrow
Many: no arrow



Method 4: One: no arrow
Many: big dot



UNIT V Process Modelling

Software system requirements are often classified as functional requirements or nonfunctional requirements:

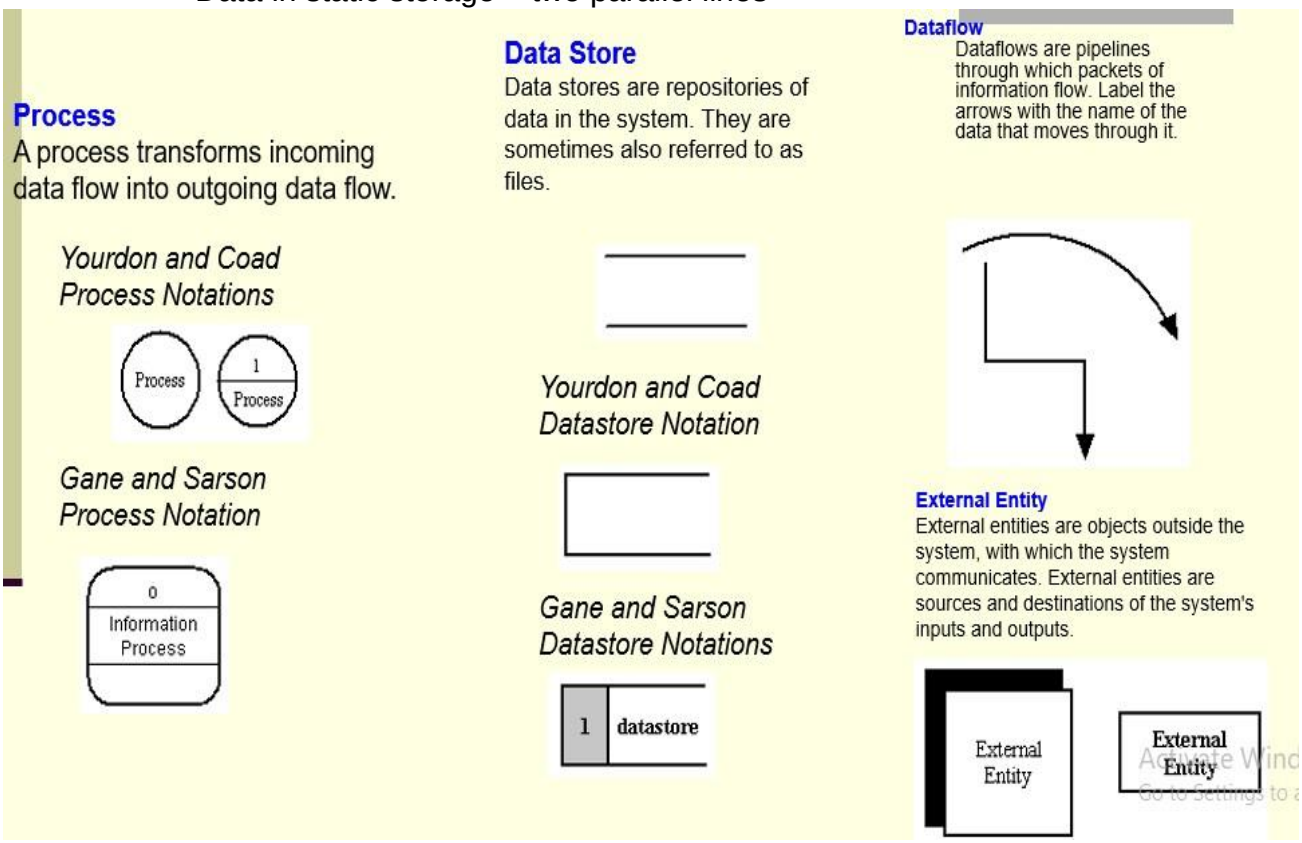
Functional requirements are statements of services the system should provide, how the system should react to particular inputs, and how the system should behave in particular situations. In some cases, the functional requirements may also explicitly state what the system should not do.

Non-functional requirements are constraints on the services or functions offered by the system. They include timing constraints, constraints on the development process, and constraints imposed by standards. Non-functional requirements often apply to the system as a whole, rather than individual system features or services.

Now, What is process modelling?
Process modelling describes the logic that programmers use to write code modules. It is typically used in structured analysis and design methods. Also called a data flow diagram (DFD). The process modelling shows the flow of information through a system, each process transforms inputs into outputs.
The models might appear to be overlapped, but they actually work together to describe the same environment from different points of view.

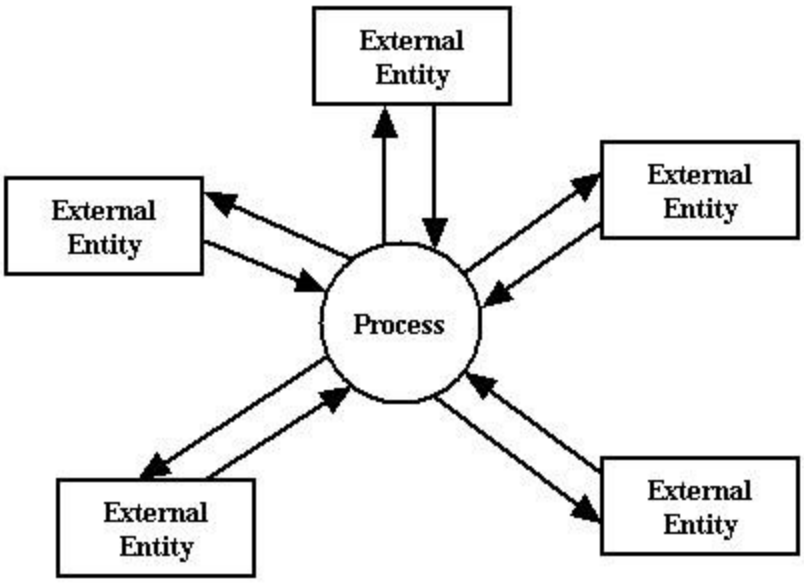
Data Flow Diagrams(DFD)
DFD existed long before computers and show the flow of data through a system Icons

- Data on the move – named arrows
- Transformations of data – named bubbles
- Sources and destinations of data – named rectangles (terminators)
- Data in static storage – two parallel lines



Context Diagrams

- A context diagram is a top level (also known as Level 0) data flow diagram. It only contains one process node (process 0) that generalizes the function of the entire system in relationship to external entities.



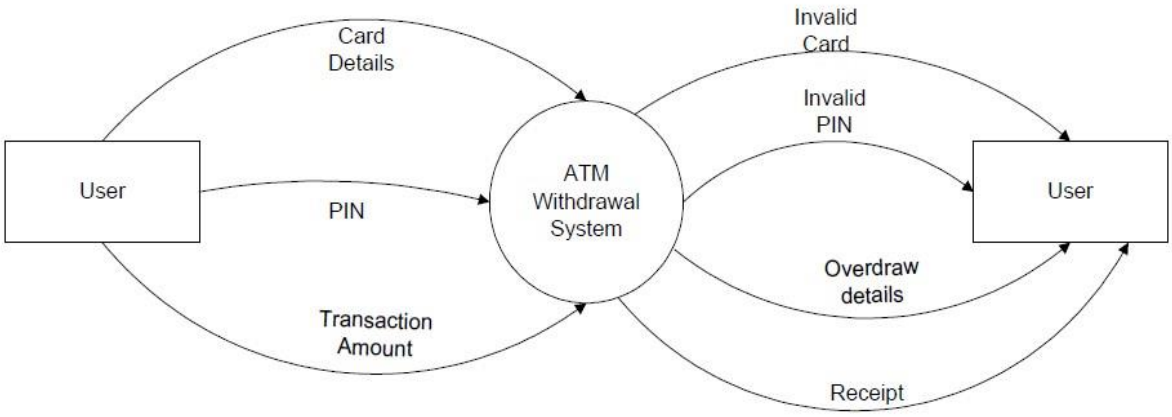


Fig: Context Diagram (0 Level DFD) of ATM Withdrawal system.

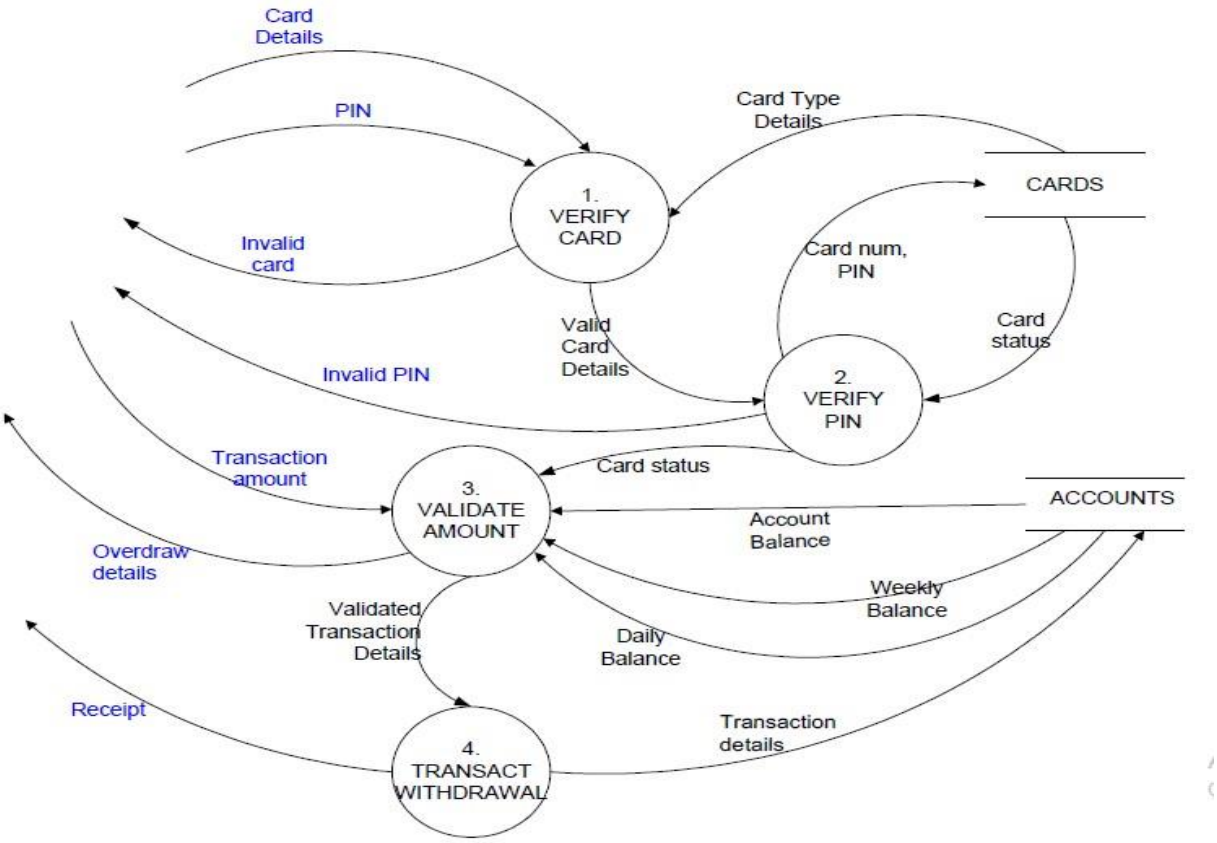


Fig : Level 1 DFD ATM cash Withdrawal

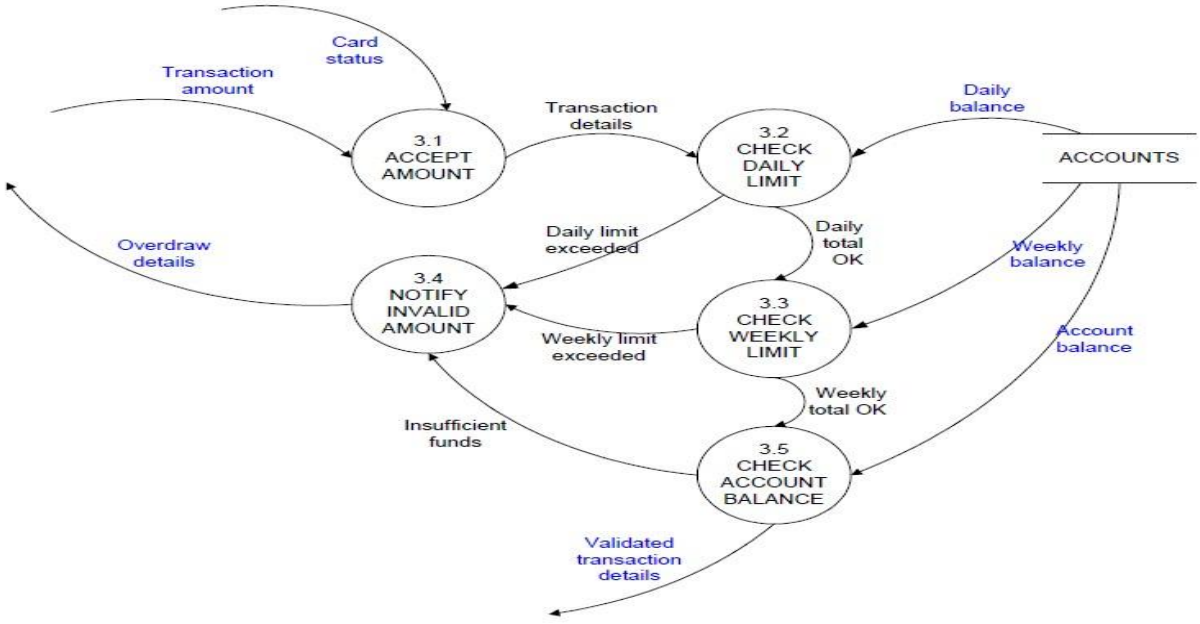
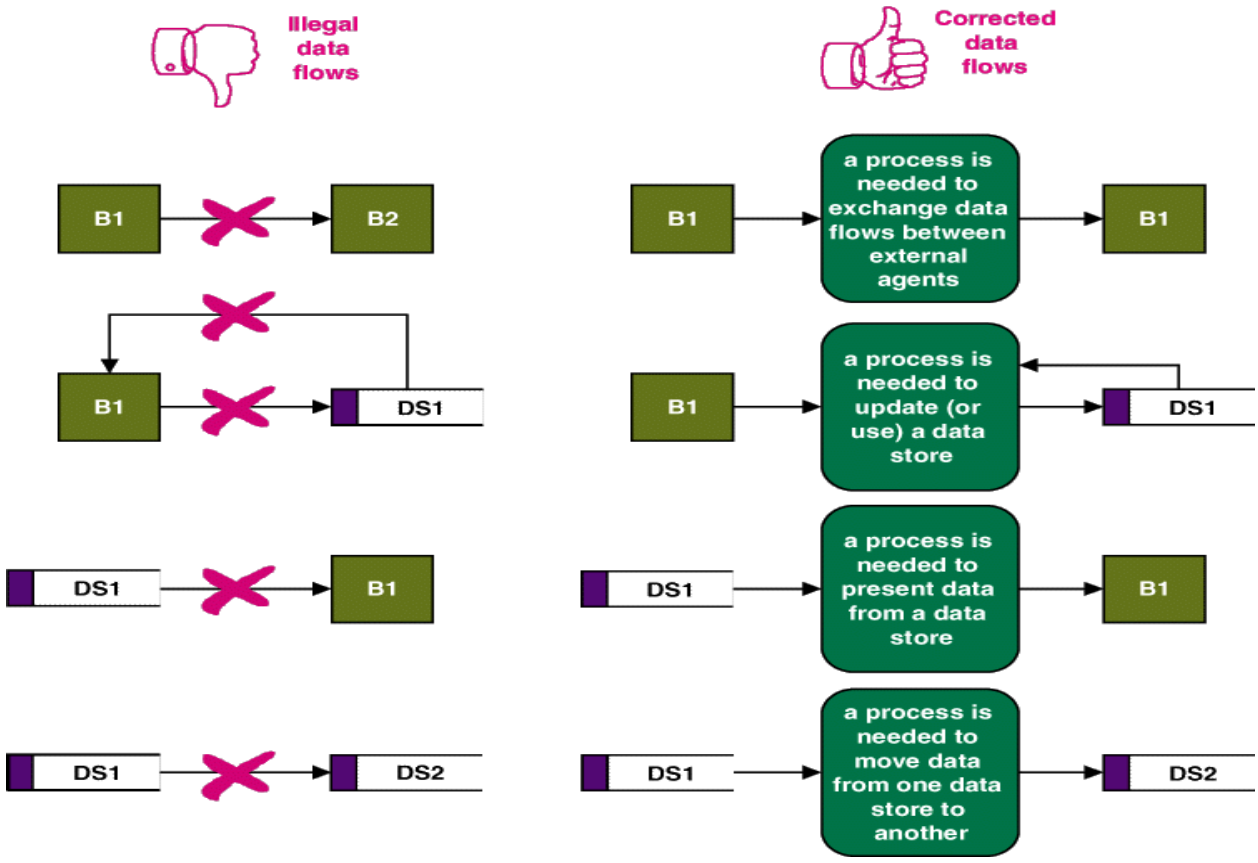


Fig: Level 2 DFD ATM cash Withdrawal (Validate Amount) Illegal Vs Legal Data Flows



Logical modelling

As good as data flow diagrams, they are not very good at showing the logic inside the processes. This is because data flow diagrams are not really designed to show the detailed logic of processes, you must model process logic using logic modeling techniques.

This topic introduces you to techniques used for modeling process decision logic. Logic modeling involves representing internal structure and functionality of processes depicted on a DFD. Logic modeling can also be used to show when processes on a DFD occur. Processes must be clearly described before they can be translated into a programming language. In this topic you will be introduced to techniques that you can use during the analysis phase to model the logic within processes: Structured English, Decision Tables and Decision Tree. This topic concludes with explanation on development strategies in creating new software which are by: Custom software, purchase commercial off-the-shelf software package and outsourced to an application service provider.

In the next section, we will learn in detail how to model the logic within processes using several techniques that are Structured English, Decision Tables and Decision Tree.

Structured English

Structured English is a modified form of English language used to specify the logic of information system processes.

The language structure is simple and clear. It describes the process logic precisely in sequence. Normally, Structured English uses:

- Statements in the form of sequence, selection and repetition.
- Indentation, which provides the structure, so that it is easy to read and understand.
- Use a limited vocabulary, including standard terms used in the data dictionary and specific words that describe the processing rules.

Structured English might look familiar to programming students because it resembles pseudo-codes, which are used in program design. Although the techniques are similar, the primary purpose of structured English is to describe the underlying business logic, while programmers, who are concerned with coding, mainly use pseudo codes as a shorthand notation for the actual code.

Examples of the Structured English are shown in Figure 7.2. After you study the sales promotion policy, notice that the structured English version describes the processing logic that the system must apply. Following these structured English rules ensures that your process descriptions are understandable to users who must confirm that the process is correct, as well as to other analysts and programmers who must design the information system from your descriptions.

To validate the system design

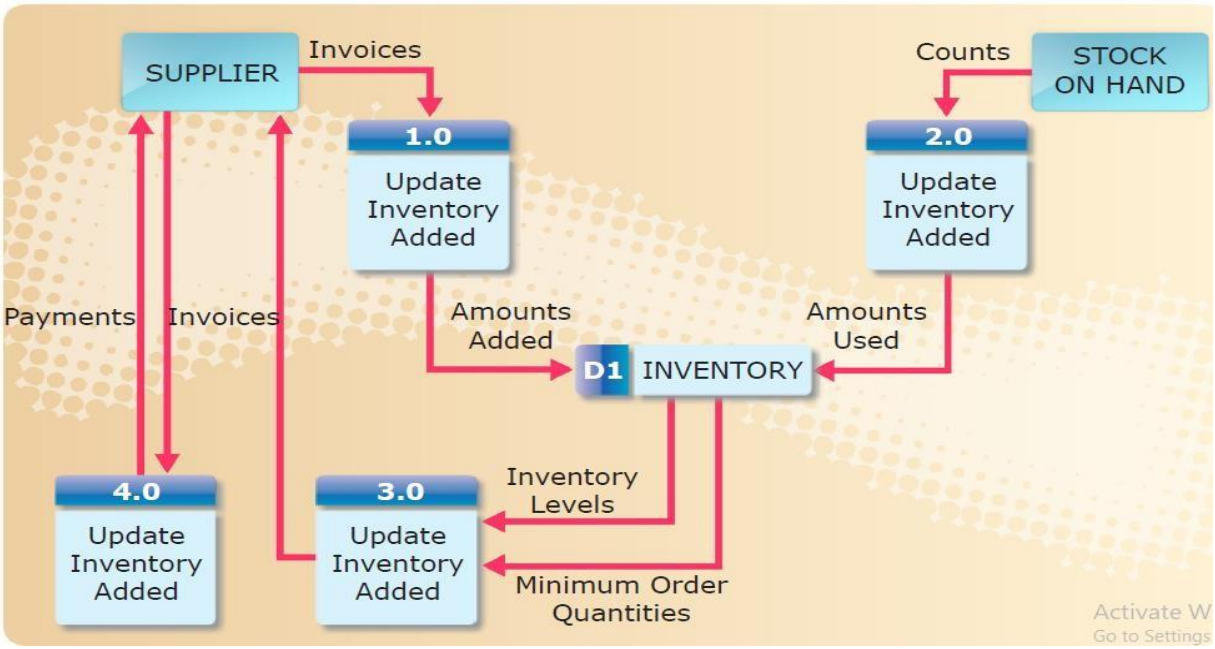
Prefered customers who order more than $1,000 are entitled to a 5% discount, and an additional 5% discount if they used our charge card.
Prefered customers who do not order more than $1,000 receive a 25% bonus coupon. All other customers receive a 5% bonus coupon.

STRUCTURED ENGLISH VERSION OF THE SALES PROMOTION POLICY

```
IF customer is a prefered customer, and  IF
        customer orders more than $1,000 then
        Apply a 5% discount, and
                IF customer uses our charge card, then  Apply
                        an additional 5% discount
                ELSE
                        Award a $25 bonus coupon
ELSE
                Award a $5 bonus coupon
```

Note:: Sample of policy and structured English version of policy Source: Adapted from Shelly et al (2006)

Let's look at another example of how structured English would represent the logic of some of the processes identified in the current logical Inventory Control System as



shown in Figure:

Figure: DFD for Inventory Control System Source: Adapted from Hoffer et al (2008)

Process 1.0: Update Inventory Added

DO

READ next Invoice-item-record
FIND matching Inventory-record
ADD Quantity-added from Invoice-item-record to Quantity-in-stock on Inventory-record

UNTIL End-of-file

Process 1.0: Update Inventory Added

DO

READ next Stock-item-record
FIND matching Inventory-record
SUBTRACT Quantity-used on Stock-item-record from Quantity-in-stock on Inventory-record

UNTIL End-of-file

Process 3.0: Generate Orders

DO

READ next Inventory-record
BEGIN IF
If Quantity-in-stock is less than Minimum-ordeer-quantity
THEN GENERATE Order
END IF
UNTIL End-of-file

Process 4.0: G**enerate Payments**
READ Today's-date
DO
SORT Invoice-records by Date
READ next Invoice-record
BEGIN IF
IF Date is 30 days greater than Today's-date
THEN GENERATE Payments
END IF
UNTIL End-of-file

Figure: Structured English representations of four processes depicted in Figure 7.3
Source: Adapted from Hoffer et al (2008)

Besides the obvious advantage of clarifying the logic and relationships found in human languages, Structured English has another important advantage: It is a communication tool. Structured English can be taught to and hence understood by users in the organisation, so if communication is important, Structured English is a viable alternative for decision analysis.

Decision Table

Decision Table shows a logical structure consisting of a combination of all process conditions and actions. Analysts often use decision table, in addition to structured English, to describe a logical process and ensure that they have not overlooked any logical possibility.

In certain situations, it provides the logic in a form that is shorter when compared with Structured English. A decision table is more suitable for representing very complex logic. It is sometimes used to check the accuracy of logic that is written in Structured English.

A decision table is a table of rows and columns, separated into four quadrants, as shown in Table 7.1. The upper left quadrant contains the condition(s); the upper right quadrant contains the condition alternatives. The lower half of the table contains the actions to be taken on the left and the rules for executing the actions on the right.

Table 7.1: Standard Format Used for Presenting a Decision Table

Conditions and Actions	Rules
Conditions	Condition Alternatives
Action	Action Entries

Figure: shows a decision table for Sales Promotion Policy that is equivalent to Figure above. The company needs to categorise its customers before deciding what kind of discount to give. Here three condition exist; was the customer a preferred customer, did the customer order more than 1,000 and did the customer use our charge card? Based on these three conditions, four possible actions can occur, as shown in the Figure.

		1	2	3	4	5	6	7	8
CONDITION	Preferred customer	Y	Y	Y	Y	N	N	N	N
	Order more than $1,000	Y	Y	N	N	Y	Y	N	N
	Used our charge card	Y	N	Y	N	Y	N	Y	N
ACTION	5% discount	X	X						
	Additional 5% discount	X							
	$25 bonus coupon			X					
	$5 bonus coupon	X	X	X					

Figure: Sample decision table for Sales Promotion Policy Source: Adapted from Shelly et al (2006)

The following steps can be adopted in building a decision table:

1. Place all the process conditions in the first column at the left of the table with one condition at each row.
2. Place all the process outcomes after the last line of conditions at the first column on the left of the table with one outcome on each line.
3. Enter all the combinations Y (for yes) or N (for no) for each condition. The columns after the condition column represent all the probabilities that are called the rules.
4. Mark with an X at each column in the outcome row for each condition concerned.

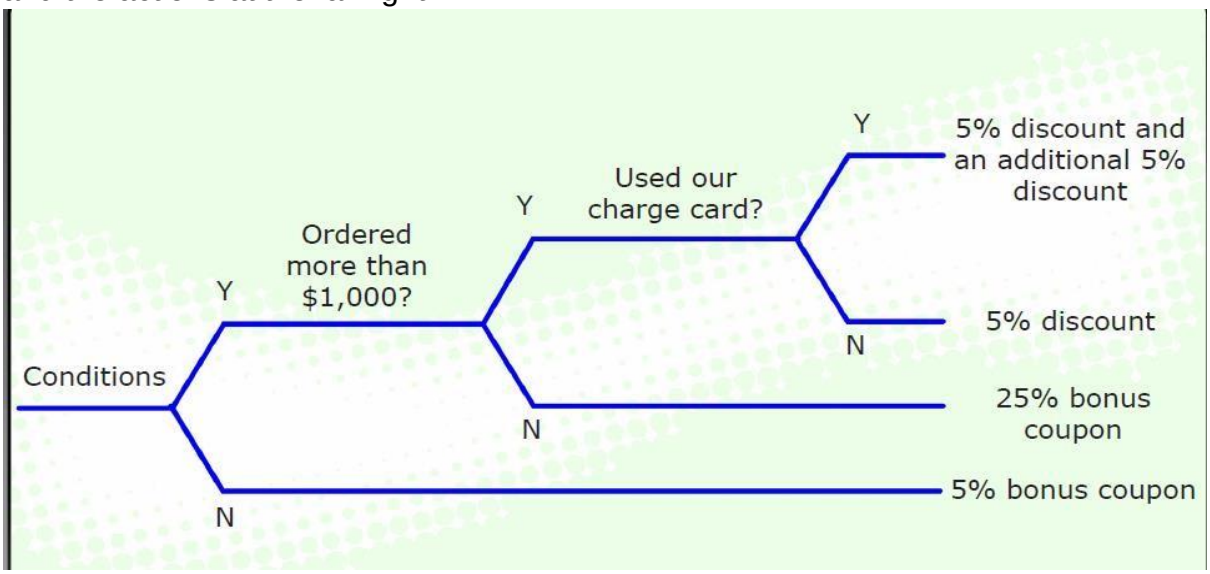
Decision tables are an important technique in the analysis of structured decisions. One major advantage of using decision tables over other methods is that tables help the analyst ensure completeness. When using decision tables, it is also easy to check for possible errors, such as impossible situations, contradictions and redundancy.

7.1.3 Decision Tree

Decision Tree is quite similar to the decision table. Both represent the same elements (condition, action and rules) but in different forms - one in the table form, while the other in graphic form.

Decision tree is a graphical representation of the conditions, actions, and rules found a decision table. It shows a logical structure, which appears like a tree. Beginning with root and stem on the left, it expands into branches and leaves towards the right.

A decision tree is read from left to right, with the conditions along the various branches and the actions at the far right.



Whether to use a decision table or a decision tree often is a matter of personal preference. A decision table might be a better way to handle complex combinations of conditions. On the other hand, a decision tree is an effective way to describe a relatively simple process.

Commercial Off-The-Shelf (COTS)

Commercial Off-The-Shelf (COTS) software is a ready-made software product that can be easily obtained. COTS include such products as Microsoft Office suite which includes Word for word processing, Excel for spreadsheets, Access for building databases and other applications. Other types of COTS software from Enterprise Resource Planning (ERP) packages such as Oracle and SAP.

Consider using COTS software when you can easily integrate the applications or packages into existing or planned systems, and when you have identified no necessity to immediately or continuously change or customise them for users. There are some advantages to purchasing COTS software that you should keep in mind as you weigh alternative. One advantage is that these products have been refined through the process of commercial use and distribution, so that often there are additional functionalities offered. Another advantage is that packaged software is typically extensively tested, and thus

extremely reliable. Table 7.3 summarises the advantages and disadvantages of purchasing COTS packages.

Outsourcing is contracting goods or services with another company or person to do a particular function such as to pay an external supplier to develop or to provide services for creating a system. This third option is to outsource some of the organisation's software needs to an Application Service Provider (ASP) that specialises in IT applications. Almost every organisation outsources in some way. Typically, the function being outsourced is considered non-core to the business.

There are specific benefits to outsourcing applications to an ASP. For example organisations that desire to retain their strategic focus and do what they are best at, may want to outsource the production of information systems applications. Additionally outsourcing one's software needs means that the organisation doing the outsourcingmay be able to sidestep the need to hire, train and retain a large IT staff. This can result in significant saving.

UNIT V
System Implementation

Implementation & Maintenance

System implementation and maintenance are the last two phases of the systems development life cycle. The purpose of implementation is to build a properly working system, test and install it in the organisation, finalise documentation and train the users.

The implementation phase of the systems development life cycle (SDLC) is the most expensive and time-consuming phase of the entire life cycle. Implementation is expensive because so many people are involved in the process; it is time consuming because of all the work that has to be completed. Regardless of methodology used, once coding and

testing are complete and the system is ready to “go live,” it must be installed (or put into production), user sites must be prepared for the new system, and users rely on the new system rather than the existing one to get their work done.
Implementing a new information system into an organizational context is not a mechanical process. The organizational context has been shaped and reshaped by the people who work in the organization. The work habits, beliefs, interrelationships, and personal goals of an organization’s members all affect the implementation process.
The purpose of maintenance is to fix and enhance the system to respond to problems and changing business conditions.
System implementation is made up of many activities. The six major activities we are concerned with in this chapter are coding, testing, installation, documentation, training, and support. The purpose of these steps is to convert the physical system specifications into working and reliable software and hardware, document the work that has been done, and provide help for current and future users and caretakers of the system.

Deliverables and outcomes from Documenting the System, training Users, and Supporting Users

At the very least, the development team must prepare user documentation. For most modern information systems, documentation includes any online help designed as part of the system interface. The development team should think through the user training process: Who should be trained? How much training is adequate for each training audience? What do different types of users need to learn during training? The training plan should be supplemented by actual training modules, or at least outlines of such modules, that at a minimum address the three questions stated previously. Finally, the development team should also deliver a user support plan that addresses issues such as how users will be able to find help once the information system has become integrated into the organization. The development team should consider a multitude of support mechanisms and modes of delivery.

Software Testing

Software testing begins early in the SDLC, even though many of the actual testing activities are carried out during implementation. During analysis, you develop a master test plan. During design, you develop a unit test plan, an integration test plan, and a system test plan. During implementation, these various plans are put into effect and the actual testing is performed.
The purpose of these written test plans is to improve communication among all the people involved in testing the application software. The plan specifies what each person’s role will be during testing. The test plans also serve as checklists you can use to determine whether the master test plan has been completed. The master test plan is not just a single document, but a collection of documents. Each of the component documents represents a complete test plan for one part of the system or for a particular type of test. Presenting a complete master test plan is far beyond the scope of this book. To give you an idea of what a master test plan involves, we present an abbreviated table of contents of one in Table given below.

1. Introduction
   a. Description of system to be tested
   b. Objectives of the test plan
   c. Method of testing
   d. Supporting documents
2. Overall Plan
   a. Milestones, schedules, and locations
   b. Test materials
      i. Test plans
      ii. Test cases
      iii. Test scenarios
      iv. Test log
3. Testing Requirements
   a. Hardware
   b. Software
   c. Personnel
4. Procedure Control
   a. Test initiation
   b. Test execution
   c. Test failure
   d. Access/change control
   e. Document control
5. Test-Specific or Component-Specific Test Plans
   a. Objectives
   b. Software description
   c. Method
   d. Milestones, schedule, progression, and locations
   e. Requirements
   f. Criteria for passing tests
   g. Resulting test materials
   h. Execution control
   i. Attachments

(*Source*: Adapted from Mosley, 1993.)

Unit testing, sometimes called *module testing*, is an automated technique whereby each module is tested alone in an attempt to discover any errors that may exist in the module's code. But because modules coexist and work with other modules in programs and the system, they must also be tested together in larger groups.
Combining modules and testing them is called **integration testing**. Integration testing is gradual. First you test the coordinating module and only one of its subordinate modules. After the first test, you add one or two other subordinate modules from the same level. Once the program has been tested with the coordinating module and all of its immediately subordinate modules, you add modules from the next level and then test the program. You continue this procedure until the entire program has been tested as a unit. **System testing** is a similar process, but instead of integrating modules into programs for testing, you integrate programs into systems. System testing follows the same incremental logic that integration testing does. Under both integration and system testing, not only do individual modules and programs get tested many times, so do the interfaces between modules and programs.
There are two important things to remember about testing information systems:
1. The purpose of testing is to confirm that the system satisfies requirements.
2. Testing must be planned.

Acceptance testing by Users

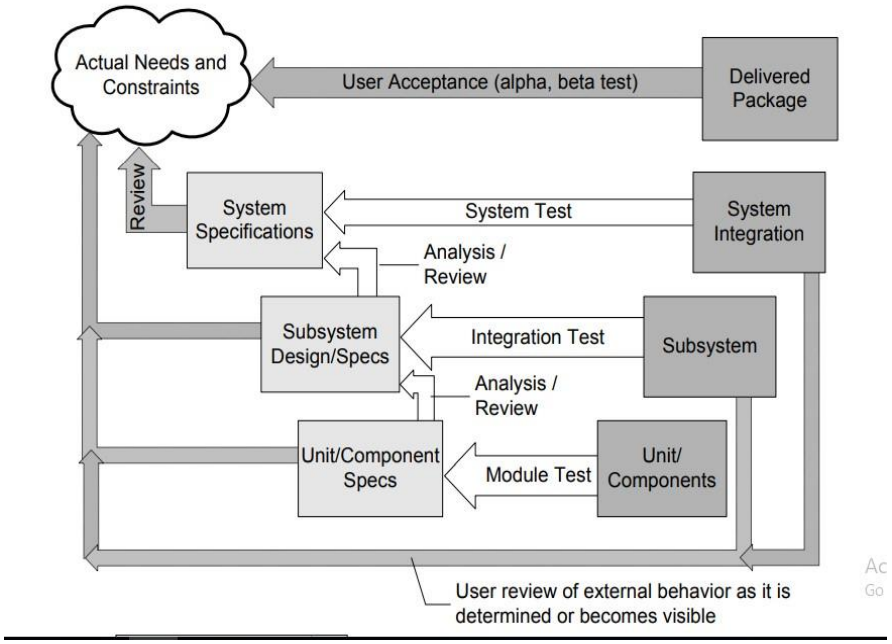
Once the system tests have been satisfactorily completed, the system is ready for **acceptance testing**, which is testing the system in the environment where it will eventually be used. Acceptance refers to the fact that users typically sign off on the system and "accept" it once they are satisfied with it. The purpose of acceptance testing is for users to determine whether the system meets their requirements. The extent of acceptance testing will vary with the organization and with the system in question. The most complete acceptance testing will include **alpha testing**, in which simulated but

typical data are used for system testing; **beta testing**, in which live data are used in the users' real working environment; and a system audit conducted by the organization's internal auditors or by members of the quality assurance group.

During alpha testing, the entire system is implemented in a test environment to discover whether the system is overtly destructive to itself or to the rest of the environment. The types of tests performed during alpha testing include the following:

- Recovery testing—forces the software (or environment) to fail in order to verify that recovery is properly performed.
- Security testing—verifies that protection mechanisms built into the system will protect it from improper penetration.
- Stress testing—tries to break the system (e.g., what happens when a record is written to the database with incomplete information or what happens under extreme online transaction loads or with a large number of concurrent users).
- Performance testing—determines how the system performs in the range of possible environments in which it may be used (e.g., different hardware configurations, networks, operating systems, and so on); often the goal is to have the system perform with similar response time and other performance measures in each environment.

In beta testing, a subset of the intended users runs the system in the users' own environments using their own data. The intent of the beta test is to determine whether the software, documentation, technical support, and training activities work as intended. In essence, beta testing can be viewed as a rehearsal of the installation phase. Problems uncovered in alpha and beta testing in any of these areas must be corrected before users can accept the system. Systems analysts can tell many stories about long delays in final user acceptance due to system bugs.



	Module test	Integration test	System test
Specification:	Module interface	Interface specs, module breakdown	Requirements specification
Visible structure:	Coding details	Modular structure (software architecture)	— none —
Scaffolding required:	Some	Often extensive	Some
Looking for faults in:	Modules	Interactions, compatibility	System functionality

Integration versus Unit Testing

- Unit (module) testing is a necessary foundation – Unit level has maximum controllability and visibility.
- Integration testing can never compensate for inadequate unit testing.

Integration testing

- **It** may serve as a process check – If module faults are revealed in integration testing, they signal inadequate unit testing – If integration faults occur in interfaces between correctly implemented modules, the errors can be traced to module breakdown and interface specifications

INSTALLATION

The process of moving from the current information system to the new one is called **installation**. All employees who use a system, whether they were consulted during the development process or not, must give up their reliance on the current system and begin to rely on the new system. Four different approaches to installation have emerged over the years: direct, parallel, single-location, and phased. The approach an organization decides to use will depend on the scope and complexity of the change associated with the new system and the organization's risk aversion.

Direct installation

Changing over from the old information system to a new one by turning off the old system when the new one is turned on.

Parallel installation

Running the old information system and the new one at the same time until management decides the old system can be turned off.

Single-location installation

Trying out a new information system at one site and using the experience to decide if and how the new system should be deployed throughout the organization.

Phased installation

Changing from the old information system to the new one incrementally, starting with one or a few functional components and then gradually extending the installation to cover the whole new system.

TRAINING AND SUPPORTING USERS

Training and **support** are critical for the success of an information system. As the person whom the user holds responsible for the new system, you and other analysts on the project team must ensure that high-quality training and support are available. Although training and support can be talked about as if they are two separate things, in organizational practice the distinction between the two is not all that clear because the two sometimes overlap. After all, both deal with learning about computing. training

Information Systems Users

Computer use requires skills, and training people to use computer applications can be expensive for organizations. Training of all types is a major activity in American corporations, but information systems training is often neglected. Many organizations tend to underinvest in computing skills training. It is true that some organizations institutionalize high levels of information system training, but many others offer no systematic training at all.

The type of training needed will vary by system type and user expertise. The list of potential topics from which you will determine if training will be useful includes the following:

- Use of the system (e.g., how to enter a class registration request)
- General computer concepts (e.g., computer files and how to copy them)
- Information system concepts (e.g., batch processing)
- Organizational concepts (e.g., FIFO inventory accounting)
- System management (e.g., how to request changes to a system)
- System installation (e.g., how to reconcile current and new systems during phased installation)

System Maintenace

The systems maintenance is the largest systems development expenditure for many organizations. In fact, more programmers today work on maintenance activities than work on new development. There is no single reason why software is maintained; however, most reasons relate to a desire to evolve system functionality in order to overcome internal processing errors or to better support changing business needs. Thus, maintenance is a fact of life for most systems. This means that maintenance can begin soon after the system is installed.

A question many people have about maintenance relates to how long organizations should maintain a system. Five years? Ten years? Longer? There is no simple answer to this question, but it is most often an issue of economics. In other words, at what point does it make financial sense to discontinue evolving an older system and build or purchase a new one? The focus of a great deal of upper IS management attention is devoted to assessing the trade-offs between maintenance and new development.

MAINTAINING INFORMATION SYSTEMS

Once an information system is installed, the system is essentially in the maintenance phase of the systems development life cycle (SDLC). When a system is in the maintenance phase, some person within the systems development group is responsible for collecting maintenance requests from system users and other interested parties, such as system auditors, data center and network management staff, and data analysts. Once collected, each request is analyzed to better understand how it will alter the system and what business benefits and necessities will result from such a change. If the change request is approved, a system change is designed and then implemented. As with the initial development of the system, implemented changes are formally reviewed and tested before installation into operational systems.

The four major activities occur within maintenance:

1. Obtaining maintenance requests
2. Transforming requests into changes
3. Designing changes
4. Implementing changes

Obtaining maintenance requests requires that a formal process be established whereby users can submit system change requests. Earlier in this book, we presented a user request document called a System Service Request (SSR). Most companies have some sort of document like an SSR to request new development, to report problems, or to request new features within an existing system.

When developing the procedures for obtaining maintenance requests, organizations must also specify an individual within the organization to collect these requests and manage their dispersal to maintenance personnel.

Once a request is received, analysis must be conducted to gain an understanding of the scope of the request. It must be determined how the request will affect the current system and how long such a project will take. As with the initial development of a system, the size of a maintenance request can be analyzed for risk and feasibility. Next, a change request can be transformed into a formal design change, which can then be fed into the maintenance implementation phase. Thus, many similarities exist between the SDLC and the activities within the maintenance process.

Deliverables and outcomes

Because the maintenance phase of the SDLC is basically a subset of the activities of the entire development process, the deliverables and outcomes from the process are the development of a new version of the software and new versions of all design documents developed or modified during the maintenance process. This means that all documents created or modified during the maintenance effort, including the system itself, represent the deliverables and outcomes of the process. Those programs and documents that did not change may also be part of the new system. Because most organizations archive prior versions of systems, all prior programs and documents must be kept to ensure the proper versioning of the system. This enables prior versions of the system to be recreated if needed. A more detailed discussion of configuration management and change control is presented later in this chapter. Because of the similarities among the steps, deliverables, and outcomes of new development and maintenance, you may be

wondering how to distinguish between these two processes. One difference is that maintenance reuses most existing system modules in producing the new system version. Other distinctions are that a new system is developed when there is a change in the hardware or software platform or when fundamental assumptions and properties of the data, logic, or process models change.

Types of Maintenance

Corrective maintenance

Corrective maintenance is changes made to a system to repair flaws in its design, coding or implementation.

For example, if you had recently purchased a new home, corrective maintenance would involve repairs made to things that had never worked as designed, such as a faulty electrical outlet or a misaligned door. Most corrective maintenance problems surface soon after installation. When corrective maintenance problems surface, they are typically urgent and need to be resolved to curtail possible interruptions in normal business activities.

Of all types of maintenance, corrective maintenance takes as much as 75 percent of all maintenance activity. This is unfortunate because corrective maintenance adds little or no value to the organisation, it simply focuses on removing defects from an existing system without adding new functionality.

In Summary, Changes made to a system to repair flaws in its design, coding, or implementation.

Adaptive maintenance

Adaptive maintenance is changes made to a system to evolve its functionality to changing business needs or technologies.

Within a home, adaptive maintenance might be adding an air conditioner to improve air circulation. Adaptive maintenance is usually less urgent than corrective maintenance because business and technical changes typically occur over some period of time. Contrary to corrective maintenance, adaptive maintenance is generally a small part of an organisation's maintenance effort, but it adds value to the organisation.

In summary, Changes made to a system to evolve its functionality to changing business needs or technologies.

Perfective maintenance

Perfective maintenance is changes made to a system to add new features or to improve performance.

It involves making enhancements to improve processing performance or interface usability or to add desired, but not necessarily required. In our home example, perfective maintenance would be adding a new room. Many systems professionals feel that perfective maintenance is not really maintenance but rather new development. In summary, Changes made to a system to add new features or to improve performance.

Preventive maintenance

Preventive maintenance is changes made to a system to avoid possible future problems. An example of preventive maintenance might be to increase the number of records that a system can process far beyond what is currently needed or to generalise how a system

sends report information to a printer so that the system can easily adapt to changes in printer technology. In our home example, preventive maintenance could be painting the exterior to better protect the home from severe weather conditions. As with adaptive maintenance, both perfective and preventive maintenance are typically a much lower priority than corrective maintenance. Over the life of a system, corrective maintenance is most likely to occur after initial system installation or after major system changes. This means that adaptive, perfective, and preventive maintenance activities can lead to corrective maintenance activities if not carefully designed and implemented. In summary, Changes made to a system to avoid possible future problems.

The Cost of Maintenance

Information systems maintenance costs are a significant expenditure. For some organizations, as much as 60 to 80 percent of their information systems budget is allocated to maintenance activities (Kaplan, 2002). These huge maintenance costs are due to the fact that many organizations have accumulated more and more older so-called legacy systems that require more and more maintenance. More maintenance means more maintenance work for programmers.

reverse engineering

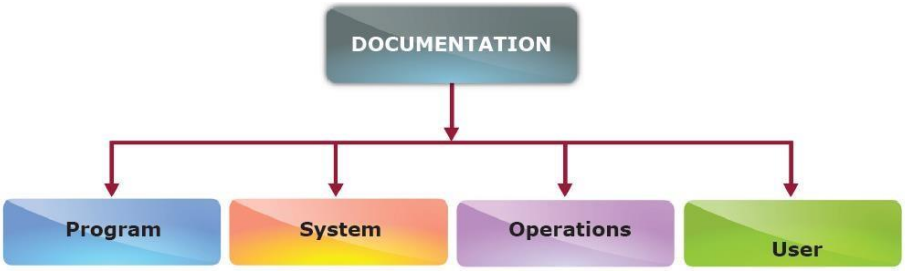
Automated tools that read program source code as input and create graphical and textual representations of design-level information such as program control structures, data structures, logical flow, and data flow.

reengineering

Automated tools that read program source code as input; perform an analysis of the program's data and logic; and then automatically, or interactively with a systems analyst, alter an existing system in an effort to improve its quality or performance.

Documentation is an activity of recording all the specifications and facts about an information system as a reference for the future.

Accurate documentation can reduce system downtime, cut costs and speed up maintenance tasks. A system that has completely gone into operation will not remain permanent just like that. From time to time, there will to be changes being made. The programmer who does the change on the program requires accurate documentation. Besides that, the operators and users working on the system throughout the system's life also need to refer to the documentation as shown in Figure.



Program documentation describes the inputs, outputs, and processing logic for all program modules. The program documentation process starts in the systems analysis phase and continues during systems implementation. Systems analysts prepare overall documentation, such as process descriptions and report layouts, early in the SDLC. This documentation guides programmers, who construct modules that arc well supported by internal and external comments and descriptions that can be understood and maintained easily. A systems analyst usually verifies that program documentation is complete and accurate.

System documentation describes the system's functions and how they are implemented. System documentation includes data dictionary entries, data flow diagrams, object models, screen layouts, source documents, and the systems request that initiated the project. System documentation is necessary reference material for the programmers and analysts who must support and maintain the system.
Most of the system documentation is prepared during the systems analysis and systems design phases. During the systems implementation phase, an analyst must review prior documentation to verify that it is complete, accurate, and up-to-date, including any changes made during the implementation process.

For example;
If a screen or report has been modified, the analyst must update the documentation. Updates to the system documentation should be made in a timely manner to prevent oversights.

Operations Documentation contains all the information needed for processing and distributing online and printed output. Typical operations documentation includes:

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

Operations documentation should be clear, concise, and available online if possible. If the IT department has an operations group, you should review the documentation with

them, early and often, to identify any problems. If you keep the operations group informed at every phase of the SDLC, you can develop operations documentation as you go along.

User documentation consists of instructions and information to users who will interact with the system such as user manuals, help screens, Frequently Asked Questions (FAQs) and tutorials. User documentation needs to be written in a language that is easily understood by users. The use of special terms that are too technical need to be avoided because users do not know anything about system development.

An excerpt of online user documentation for Microsoft Word appears in Figure 10.2. Notice how the documentation is organised by topic, each of which has several subtopics underneath. This particular page is devoted to provide assistance and the menu at the left provides links to other relevant pages like training and templates. Such presentation methods have become standard for help files in online personal computer documentation.

UNIT VII
Object Oriented System Analysis and Design

Object

- Object is an abstraction of something in a problem domain, reflecting the capabilities of the system to keep information about it, interact with it, or both.
- Objects are entities in a software system which represent instances of real-world and system entities

Object	**Identity**	**Behaviors**	**State**
An employee	“Mr. John”	Join(), Retire()	Joined, Retired.
A book	“Book with title Object Oriented Analysis Design”	AddExemplar,	Rent, available, reserved

A sale	"Sale no 0015, 15/12/98"	SendInvoiced(), Cancel().	Invoiced, cancelled.

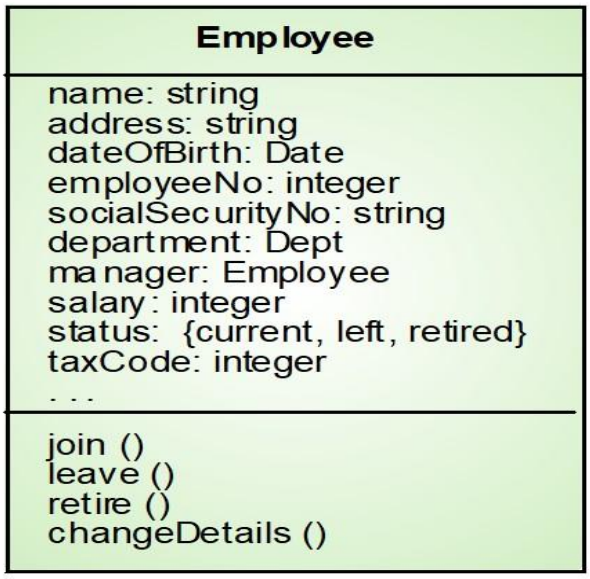
Term of objects

- Attribute: data items that define object.
- Operation: function in a class that combine to form behavior of class.
- Methods: the actual implementation of procedure (the body of code that is executed in response to a request from other objects in the system).

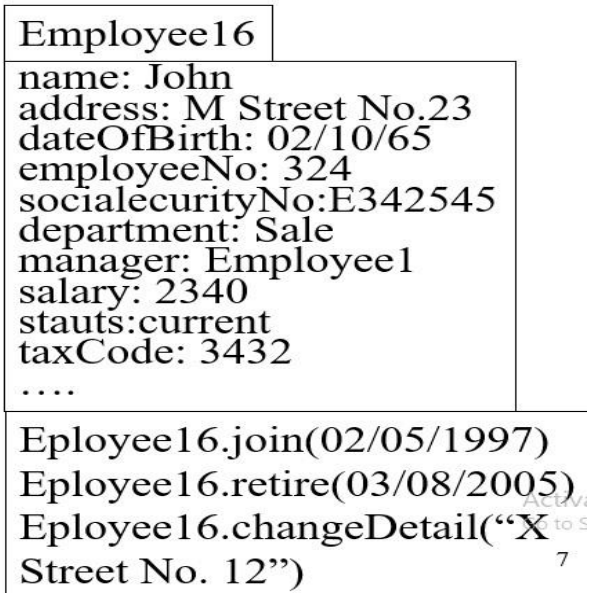
Difference between class and object

Employee object & class

Class



Object



7

Identifying Object •

Objects can be:

- External Entity (e.g., other systems, devices, people) that produce or consume information to be used by system
- Things (e.g., reports, displays, letters, signals) that are part of information domain for the problem
- Places (e.g., book's room) that establish the context of the problem and the overall function of the system.
- Organizational units (e.g., division, group, team, department) that are relevant to an application,
- Transaction (e.g., loan, take course, buy, order).

OO Analysis - examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain. In other words, the world (of the system) is modelled in terms of objects and classes.
OO Design - OO decomposition and a notation for depicting models of the system under development. Structures are developed whereby sets of objects collaborate to provide the behaviours that satisfy the requirements of the problem.

Software Engineering Development Activities
In this section, we give an overview of the technical activities associated with objectoriented software engineering. Development activities deal with the complexity by constructing and validating models of the application domain or the system.
Development activities include
• Requirements Elicitation
• Analysis
• System Design
• Object Design
• Implementation
• Testing

Requirements Elicitation
During **requirements elicitation**, the client and developers define the purpose of the system. The result of this activity is a description of the system in terms of actors and use cases. Actors represent the external entities that interact with the system. Actors include roles such as end users, other computers the system needs to deal with (e.g., a central bank computer, a network), and the environment (e.g., a chemical process). Use cases are general sequences of events that describe all the possible actions between an actor and the system for a given piece of functionality. Figure 1-3 depicts a use case for the TicketDistributor example we discussed previously.

Use case name: PurchaseOneWayTicket
Participating actor: Initiated by Traveler
Flow of events 1. The Traveler selects the zone in which the destination station is located.
2. The TicketDistributor displays the price of the ticket.
3. The Traveler inserts an amount of money that is at least as much as the price of the ticket.
4. The TicketDistributor issues the specified ticket to the Traveler and returns any change.
Entry condition The Traveler stands in front of the TicketDistributor, which may be located at the station of origin or at another station.
Exit condition The Traveler holds a valid ticket and any excess change. *Quality requirements* If the transaction is not completed after one minute of inactivity, the TicketDistributor returns all inserted change.

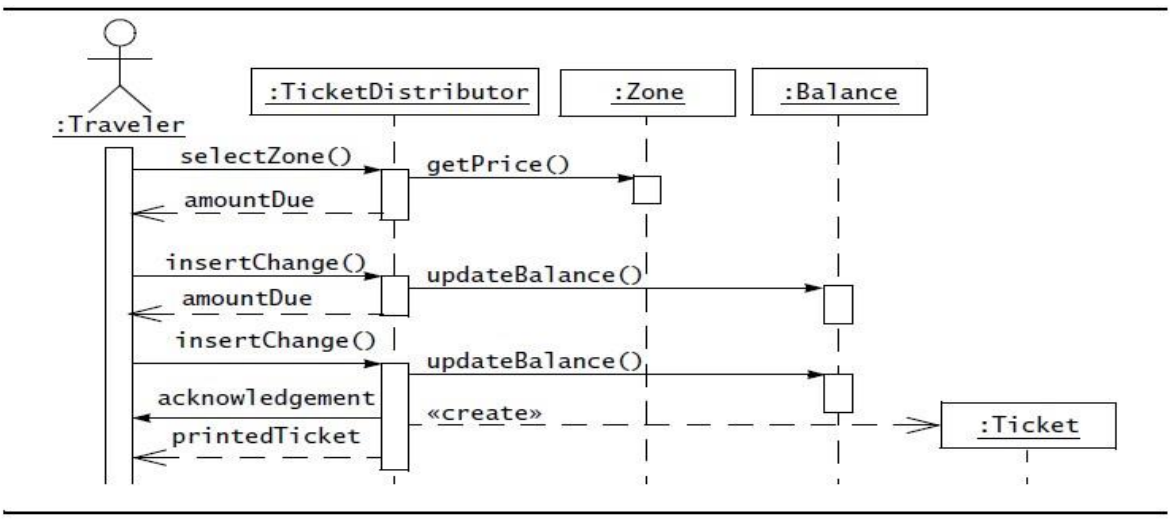


Figure 1-4 A dynamic model for the `TicketDistributor` (UML sequence diagram). This diagram depicts the interactions between the actor and the system during the `PurchaseOneWayTicket` use case and the objects that participate in the use case.

During **system design**, developers define the design goals of the project and decompose the system into smaller subsystems that can be realized by individual teams. Developers also select strategies for building the system, such as the hardware/software platform on which the system will run, the persistent data management strategy, the global control flow, the access control policy, and the handling of boundary conditions. The result of system design is a clear description of each of these strategies, a subsystem decomposition, and a deployment diagram representing the hardware/software mapping of the system. Whereas both analysis and system design produce models of the system under construction, only analysis deals with entities that the client can understand. System design deals with a much more refined model that includes many entities that are beyond the comprehension (and interest) of the client. Figure above depicts an example of system decomposition for the TicketDistributor.

During **object design**, developers define solution domain objects to bridge the gap between the analysis model and the hardware/software platform defined during system design. This includes precisely describing object and subsystem interfaces, selecting offthe-shelf components, restructuring the object model to attain design goals such as extensibility or understandability, and optimizing the object model for performance. The result of the object design activity is a detailed object model annotated with constraints and precise descriptions for each element.

During **implementation**, developers translate the solution domain model into source code. This includes implementing the attributes and methods of each object and integrating all the objects such that they function as a single system. The implementation activity spans the gap between the detailed object design model and a complete set of source code files that can be compiled. We describe the mapping of UML models to code in Chapter 10, *Mapping Models to Code*. We assume the reader is already familiar with programming concepts and knows how to program data structures and algorithms using an object-oriented language such as Java or C++.

UNIFIED MODELLING LANGUAGE (UML)

UML is a notation that resulted from the unification of OMT (Object Modeling Technique [Rumbaugh et al., 1991]), Booch [Booch, 1994], and OOSE (Object-Oriented Software Engineering [Jacobson et al., 1992]). UML has also been influenced by other objectoriented notations, such as those introduced by Mellor and Shlaer [Mellor & Shlaer, 1998], Coad and Yourdon [Coad et al., 1995], Wirfs-Brock [Wirfs-Brock et al., 1990], and Martin and Odell [Martin & Odell, 1992].

The goal of UML is to provide a standard notation that can be used by all object-oriented methods and to select and integrate the best elements of precursor notations. For example, UML includes the use case diagrams introduced by OOSE and uses many features of the OMT class diagrams. UML also includes new concepts that were not present in other major methods at the time, such as extension mechanisms and a constraint language. UML has been designed for a broad range of applications. Hence, it provides constructs for a broad range of systems and activities (e.g., distributed systems, analysis, system design, deployment). System development focuses on three different models of the system (see Figure 1-2):

- The **functional model**, represented in UML with use case diagrams, describes the functionality of the system from the user's point of view.

- The **object model**, represented in UML with class diagrams, describes the structure of the system in terms of objects, attributes, associations, and operations. During requirements and analysis, the object model starts as the *analysis object model* and describes the application concepts relevant to the system. During system design, the object model is refined into the *system design object model* and includes descriptions of the subsystem interfaces. During object design, the object model is refined into the *object design model* and includes detailed descriptions of solution objects.

- The **dynamic model**, represented in UML with interaction diagrams, state machine

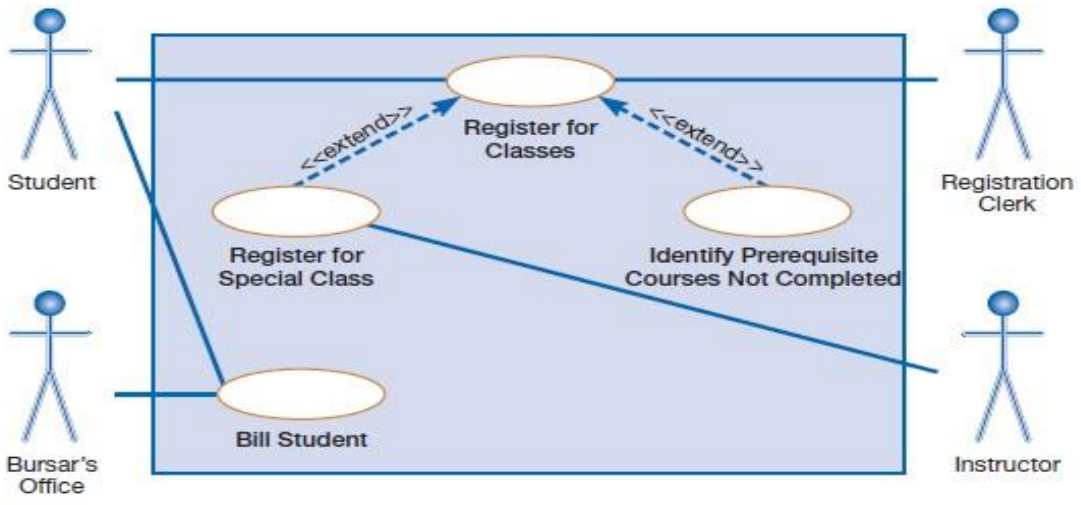
USE CASES

As we discussed in unit V DFDs are powerful modeling tools that you can use to show a system's functionality and the flow of data necessary for the system to perform its

functions. DFDs are not the only way to show functionality, of course. Another way is use case modeling. Use case modeling helps analysts analyze the functional requirements of a system. Use case modeling helps developers understand the functional requirements of the system without worrying about how those requirements will be implemented. The process is inherently iterative—analysts and users work together throughout the model development process to further refine their use case models. Although use case modeling is most often associated with object-oriented systems analysis and design, the concept is flexible enough that it can also be used within more traditional. Now, we will be discussing about Use Cases.

What is a Use case?

A **use case** shows the behavior or functionality of a system (see Figure given bwlow). It consists of a set of possible sequences of interactions between a system and a user in a particular environment, possible sequences that are related to a particular goal. A use case describes the behavior of a system under various conditions as the system responds to requests from principal actors. A principal actor initiates a request of the system, related to a goal, and the system responds. A use case can be stated as a present-tense verb phrase, containing the verb (what the system is supposed to do) and the object of the verb (what the system is to act on). For example, use case names would include Enter Sales Data, Compute Commission, Generate Quarterly Report. As with DFDs, use cases do not reflect all of the system requirements; they must be augmented by documents that detail requirements, such as business rules, data fields and formats, and complex formulas.



A use case model consists of actors and use cases. An **actor** is an external entity that interacts with the system. It is someone or something that exchanges information with the system. For the most part, a use case represents a sequence of related actions initiated

by an actor to accomplish a specific goal; it is a specific way of using the system (Jacobson et al., 1992). Note that there is a difference between an actor and a user. A user is anyone who uses the system. An actor, on the other hand, represents a role that a user can play. The actor's name should indicate that role. An actor is a type or class of users; a user is a specific instance of an actor class playing the actor's role. Note that the same user can play multiple roles. For example, if William Alvarez plays two roles, one as an instructor and the other as an adviser, we represent him as an instance of an actor called Instructor and as an instance of another actor called Adviser. Because actors are outside the system, you do not need to describe them in detail. The advantage of identifying actors is that it helps you to identify the use cases they carry out.

For identifying use cases, Jacobson et al. (1992) recommend that you ask the following questions:

- What are the main tasks performed by each actor?
- Will the actor read or update any information in the system?
- Will the actor have to inform the system about changes outside the system?
- Does the actor have to be informed of unexpected changes?

Definitions And Symbols Used In Use Cases

Use case diagramming is relatively simple because it involves only a few symbols. However, like DFDs and other relatively simple diagramming tools, these few symbols can be used to represent quite complex situations. Mastering use case diagramming takes a lot of practice. The key symbols in a use case diagram are illustrated in Figure above and explained below:

a) *Actor:* As explained earlier, an actor is a role, not an individual. Individuals are instances of actors. One particular individual may play many roles simultaneously. An actor is involved with the functioning of a system at some basic level. **Actors are represented by stick figures**.

b) *Use case:* Each **use case is represented as an ellipse**. Each use case represents a single system function. The name of the use case can be listed inside the ellipse or just below it.

c) *System boundary:* The system boundary is represented as a box that includes all of the relevant use cases. Note that actors are outside the system boundary.

Connections. In Figure above, note that the actors are connected to use cases with lines, and that use cases are connected to each other with arrows. A solid line connecting an actor to a use case shows that the actor is involved in that particular system function. The solid line does not mean that the actor is sending data to or receiving data from the use case. Note that all of the actors in a use case diagram are not involved in all the use cases in the system.

The dotted-line arrows that connect use cases also have labels (there is an <<extend>> label on the arrows in Figure). These use case connections and their labels are explained

next. Note that use cases do not have to be connected to other use cases. The arrows between use cases do not illustrate data or process flows.

- *Extend relationship.* An **extend relationship** extends a use case by adding new behaviors or actions. It is shown as a dotted-line arrow pointing toward the use case that has been extended and labeled with the <<extend>> symbol.

An **activity diagram**

An **activity diagram** shows the conditional logic for the sequence of system activities needed to accomplish a business process. An individual activity may be manual or automated. Further, each activity is the responsibility of a particular organizational unit. The basic activity diagram notation contains only a few symbols (Figure 7-36). Each activity is represented with a rounded rectangle, with the action performed by that activity written inside it. The diagram itself represents an overall process that is made up of a series of activities.

The beginning of the process is indicated with a filled in circle. An arrow connects the circle with the first activity. The end of the process is also indicated with a filled in circle, but it is surrounded by another circle. An activity diagram is designed to show conditional logic. The symbol that illustrates a choice that must be made is a diamond, and it is called a branch. The diamond follows one activity, so there is an arrow coming into it. Two activities follow it, so an arrow leaves the diamond for each possible course of action. These arrows are labeled with the conditions that cause each branch to be followed. The different courses of possible action end at some point, so they join together, and the overall process continues. The point at which they join is called a merge, and the symbol for it is also a diamond. For a merge diamond, two arrows come into the diamond but only one leaves. An activity diagram can also show parallel activities. We show where the parallel activities begin with a fork. The fork is shown with a horizontal line, which has two arrows coming out of it. When the parallel activities are done, they come back together in a join. The join is shown as a horizontal line with two arrows coming in and one arrow leaving. After the join, the overall process continues.

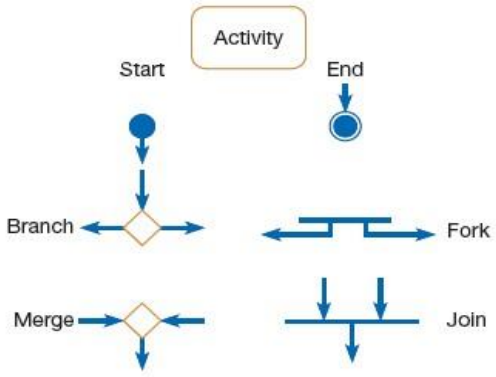


FIGURE 7-36
Basic notation for activity diagrams.

Figure given below shows a simple activity diagram that illustrates conditional logic. A user wants to log in to a system, which could be a website or some other information system. If the user has already registered, then the flow of activities shifts to the left-hand side. If the user has not registered, the flow shifts to the right-hand side. First, the user has to click on the 'Register' button. A registration form will then open, which the user must complete. The system then checks to see if the form has been completed correctly. If so, the action proceeds to the merge point. If not, the user must complete the registration form again. Once the user has successfully logged in, the flow of the action moves to the

end of the activity diagram. In practice, this activity diagram would illustrate only one small

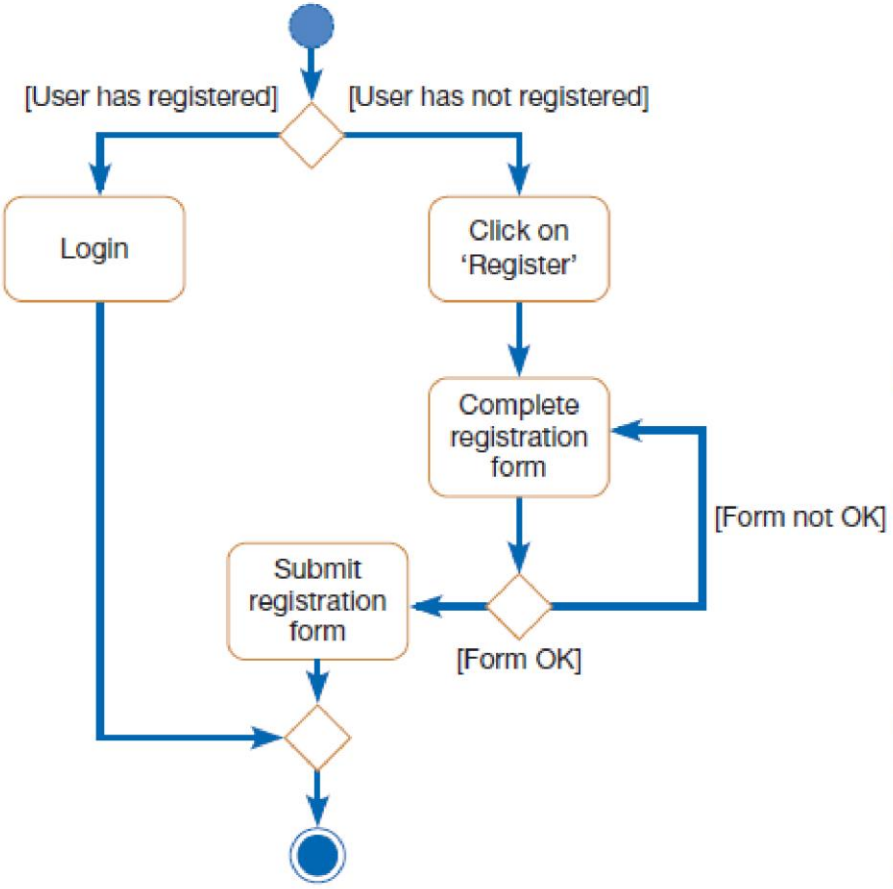


FIGURE 7-37
Simple activity diagram showing conditional logic.

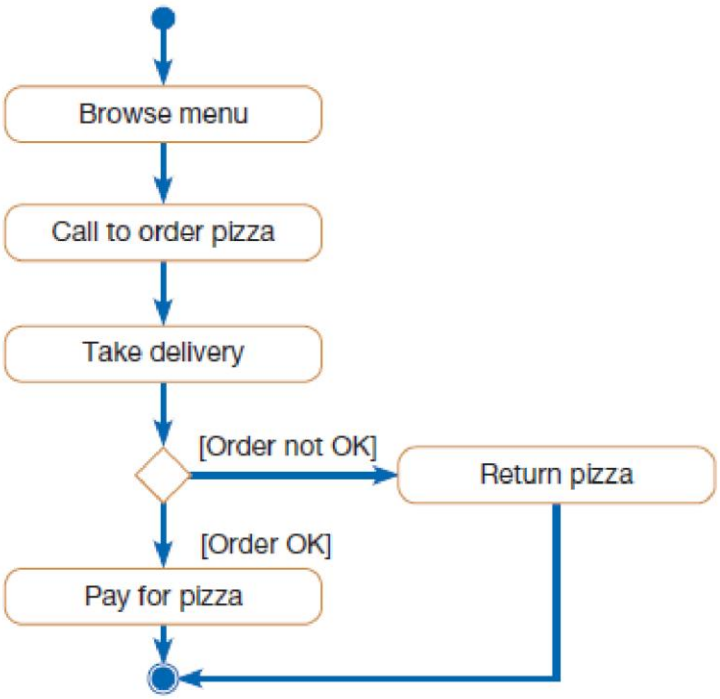


FIGURE 7-38
A simple activity diagram for ordering pizza.

part of a larger process involving a system.
There would typically be activities before the log-in branch, and there would be other activities following the merge at the bottom of the diagram. Another simple activity diagram is shown in Figure 7-38. This activity diagram shows how you might order a pizza. The process starts with the first activity, browsing the menu of your favorite pizza restaurant or take-out place. The next step is to call and order the pizza. The third step is to take delivery. But if the pizza is not what you ordered, you don't want it. So there is a conditional branch included. If the order is correct, then you pay for the pizza. If the pizza is not what you wanted, then you return it. Note how both of the activities that follow the branch lead to the end of the activity diagram.

This activity diagram shows how you might order a pizza. The process starts with the first activity, browsing the menu of your favorite pizza restaurant or take-out place. The next step is to call and order the pizza. The third step is to take delivery. But if the pizza is not what you ordered, you don't want it. So there is a conditional branch included. If the order is correct, then you pay for the pizza. If the pizza is not what you wanted, then you return it.

When to use Activity Diagram?

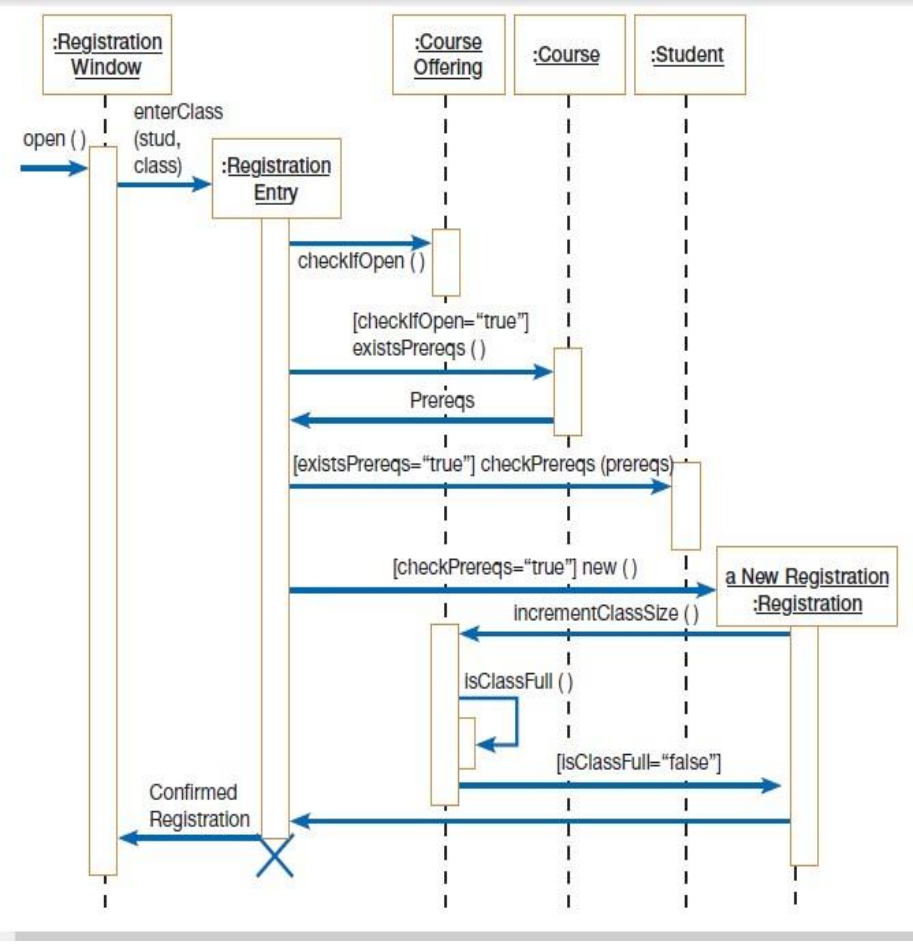
An activity diagram is a flexible tool that can be used in a variety of situations. It can be used at a high level as well as at a low level of abstraction. It should be used only when it adds value to the project. Our recommendation is to use it sparingly. Ask the following question: Does it add value or is it redundant? Specifically, an activity diagram can be used to accomplish the following tasks:

1. Depict the flow of control from activity to activity.
2. Help in use case analysis to understand what actions need to take place.
3. Help in identifying extensions in a use case.
4. Model work flow and business processes.
5. Model the sequential and concurrent steps in a computation process.

The interpretation of the term *activity* depends on the perspective from which one is drawing the diagram. At a conceptual level, an activity is a task that needs to be done, whether by a human or a computer (Fowler and Scott, 1999). At an implementation level, an activity is a method or a class.

DYNAMIC MODELING: SEQUENCE DIAGRAMS

A **sequence diagram** depicts the interactions among objects during a certain period of time. Because the pattern of interactions varies from one use case to another, each sequence diagram shows only the interactions pertinent to a specific use case. It shows the participating objects by their lifelines and the interactions among those objects—arranged in time sequence—by the messages they exchange with one another.



The vertical axis of the diagram represents time, and the horizontal axis represents the various participating objects. Time increases as we go down the vertical axis. The diagram has six objects, from an instance of Registration Window on the left to an instance of Registration called "a New Registration" on the right. The ordering of the objects has no significance. However, you should try to arrange the objects so that the diagram is easy to read and understand. Each object is shown as a vertical dashed line called the lifeline;

the lifeline represents the object's existence over a certain period of time. An object symbol—a box with the object's name underlined—is placed at the head of each lifeline.

A thin rectangle superimposed on the lifeline of an object represents an activation of the object. An **activation** shows the time period during which an object performs an operation, either directly or through a call to some subordinate operation. The top of the rectangle, which is at the tip of an incoming message, indicates the initiation of the activation; the bottom indicates its completion.
Objects communicate with one another by sending messages. A message is shown as a solid arrow from the sending object to the receiving object. For example, the checkIfOpen message is represented by an arrow from the Registration Entry object to the Course Offering object. When the arrow points directly into an object box, a new instance of that object is created. Normally the arrow is drawn horizontally, but in some situations (discussed later), you may have to draw a sloping message line.

In Summary,
Sequence diagram: Depicts the interactions among objects during a certain period of time.
Activation: The time period during which an object performs an operation.
Synchronous message: A type of message in which the caller has to wait for the receiving object to finish executing the called operation before it can resume execution itself
Simple message: A message that transfers control from the sender to the recipient without describing the details of the communication.
Asynchronous message: A message in which the sender does not have to wait for the recipient to handle the message.

Class Diagrams

Class diagrams describe the structure of the system in terms of classes and objects. Classes are abstractions that specify the attributes and behavior of a set of objects. A class is a collection of objects that share a set of attributes that distinguish the objects as members of the collection. Objects are entities that encapsulate state and behavior. Each object has an identity: it can be referred individually and is distinguishable from other objects.
In UML, classes and objects are depicted by boxes composed of three compartments. The top compartment displays the name of the class or object. The center compartment displays its attributes, and the bottom compartment displays its operations. The attribute and operation compartments can be omitted for clarity. Object names are underlined to indicate that they are instances. By convention, class names start with an uppercase letter. Objects in object diagrams may be given names (followed by their class) for ease of reference. In that case, their name starts

Scenario name warehouseOnFire
Participating actor instances bob, alice:FieldOfficer, john:Dispatcher *Flow of events*

1. Bob, driving down main street in his patrol car, notices smoke coming out of a warehouse. His partner, Alice, activates the "Report Emergency" function from her FRIEND laptop.
2. Alice enters the address of the building, a brief description of its location (i.e., northwest corner), and an emergency level. In addition to a fire unit, she requests several paramedic units on the scene given that area appear to be relatively busy. She confirms her input and waits for an acknowledgment.
3. John, the Dispatcher, is alerted to the emergency by a beep of his workstation. He reviews the information submitted by Alice and acknowledges the report. He allocates a fire unit and two paramedic units to the Incident site and sends their estimated arrival time (ETA) to Alice.
4. Alice receives the acknowledgment and the ETA.

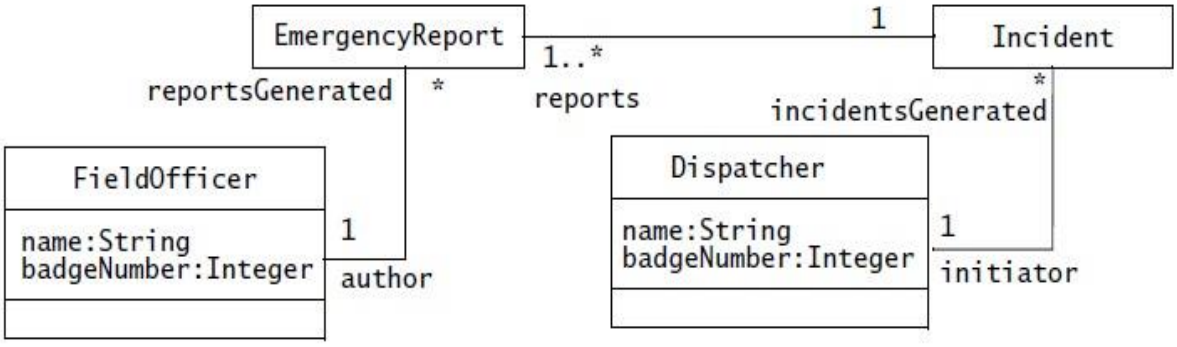


Fig: An example of a UML class diagram: classes that participate in the ReportEmergency use case

State Machine Diagrams

A UML **state machine** is a notation for describing the sequence of states an object goes through in response to external events. UML state machines are extensions of the finite state machine model. On one hand, state machines provide notation for nesting

states and state machines (i.e., a state can be described by a state machine). On the other hand, state machines provide notation for binding transitions with message sends and conditions on objects. UML state machines are largely based on Harel's statecharts [Harel, 1987] and have been adapted for use with object models [Douglass, 1999]. UML state machines can be used to represent any Mealy or Moore state machine. A **state** is a condition satisfied by the attributes of an object. For example, an Incident object in FRIEND can exist in four states: Active, Inactive, Closed, and Archived A **transition** represents a change of state triggered by events, conditions, or time. For example, Figure 2-37 depicts three transitions: from the Active state into the Inactive state, from the Inactive state to the Closed state, and from the Closed state to the Archived state. A state is depicted by a rounded rectangle. A transition is depicted by an open arrow connecting two states. States are labeled with their name. A small solid black circle indicates the initial state. A circle surrounding a small solid black circle indicates a final state.

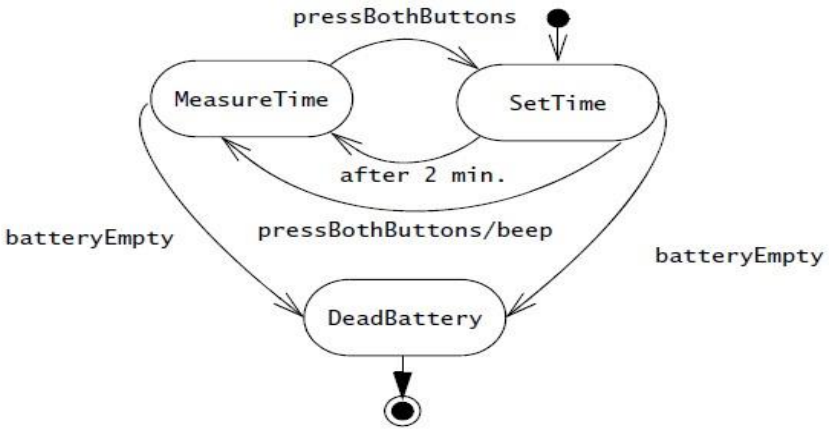


Figure 2-38 State machine diagram for 2Bwatch set time function.

Let's have a look of Automated Teller Machine (ATM) & Unified Modeling Language (UML)

Design Case Study: Automated Teller Machine (ATM)
Statement of purpose: An ATM is an electronic device designed for automated dispensing of money. A user can withdraw money quickly and easily after authorization. The user interacts with the system through a card reader and a numerical keypad. A small display screen allows messages and information to be displayed to the user. Bank members can access special functions such as ordering a statement.

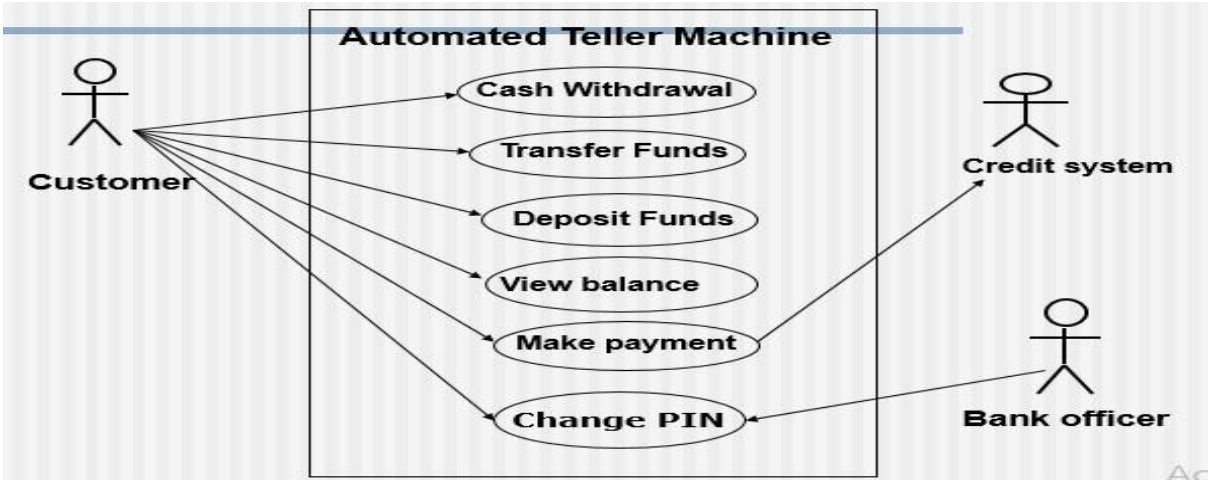


Fig: Use Cases in ATM

Use Case: Cash Withdrawal A Use Case Description:

- **When a customer inserts a card in the ATM, the machine reads the code from the card and checks if it is a valid card. If the card is valid then the machine queries the customer for a PIN number, else the card is ejected.**
- **When the machine matches customer coded in the PIN number, the machine checks the validity of the PIN number. If the PIN number is correct and matches the card number then the machine asks for the desired transaction the customer wishes to perform.**
- **When the customer selects cash withdrawal the machine asks for the desired amount with a warning indicating only multiple of $10 is allowed.**
- **When ....**

Scenarios

- Use case represents a *generic* case of interaction
- A particular course that a use-case instance might take is called as *scenario*
- For example, there may be many Withdraw Cash scenarios where no cash is withdrawn!
- Card holder has insufficient funds on account
- ATM cannot connect to bank's system
- ATM does not contain enough cash
- Customer cancels the transaction for some reason

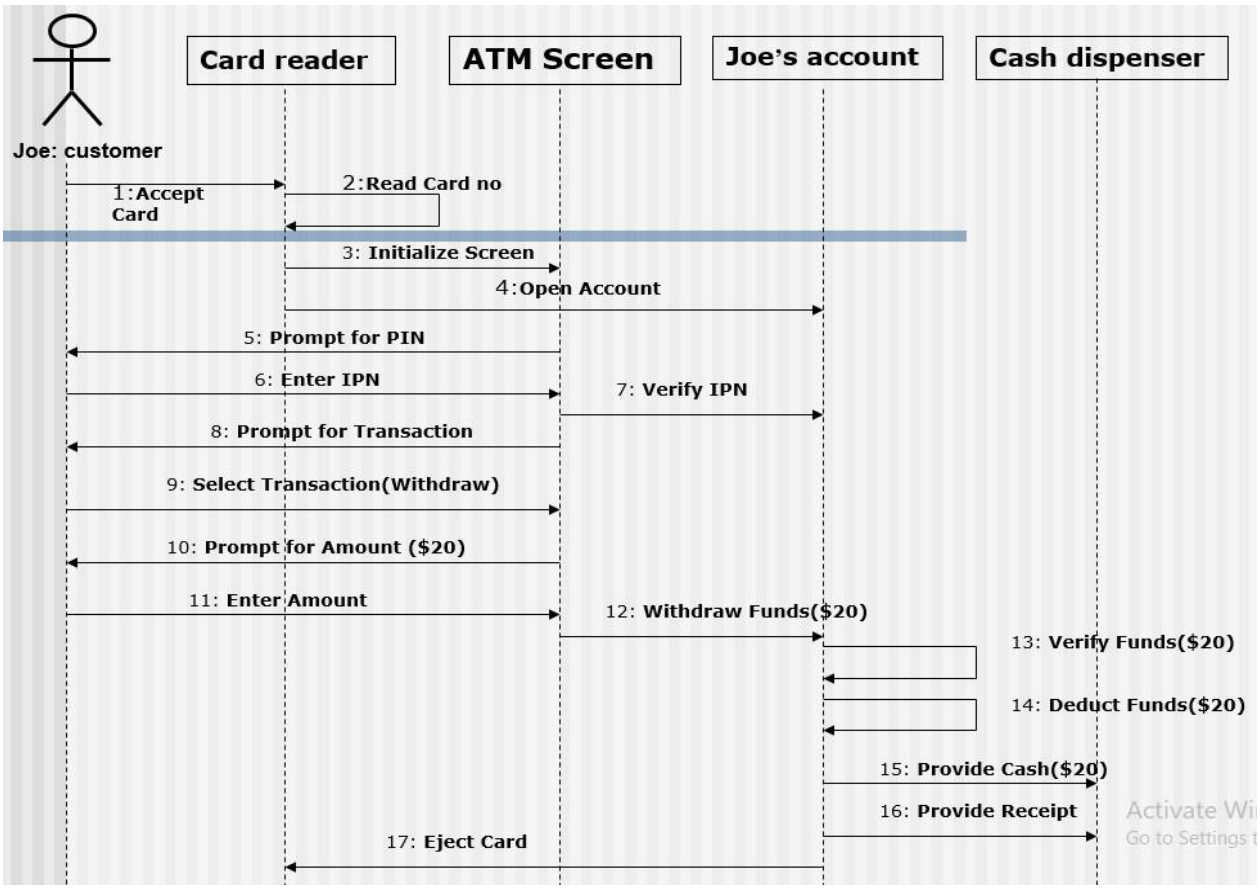


Fig: Interaction diagram

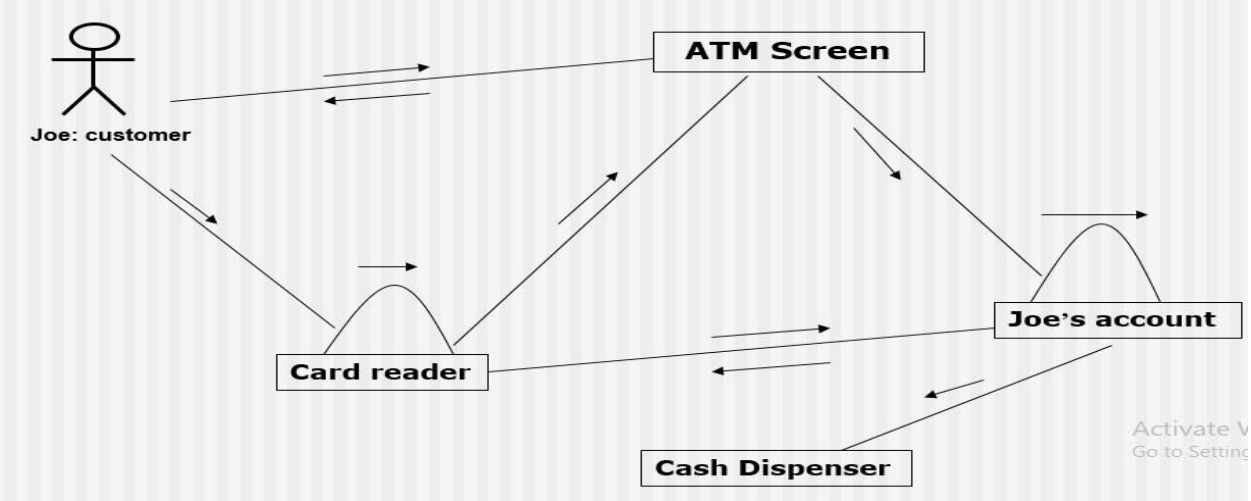
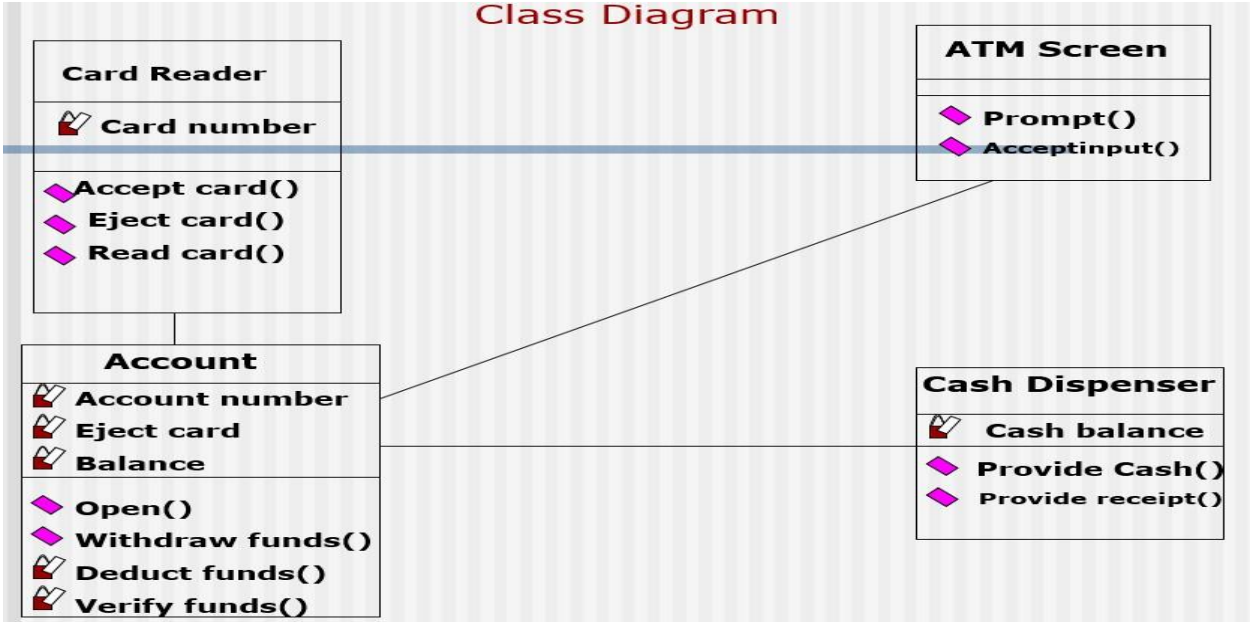
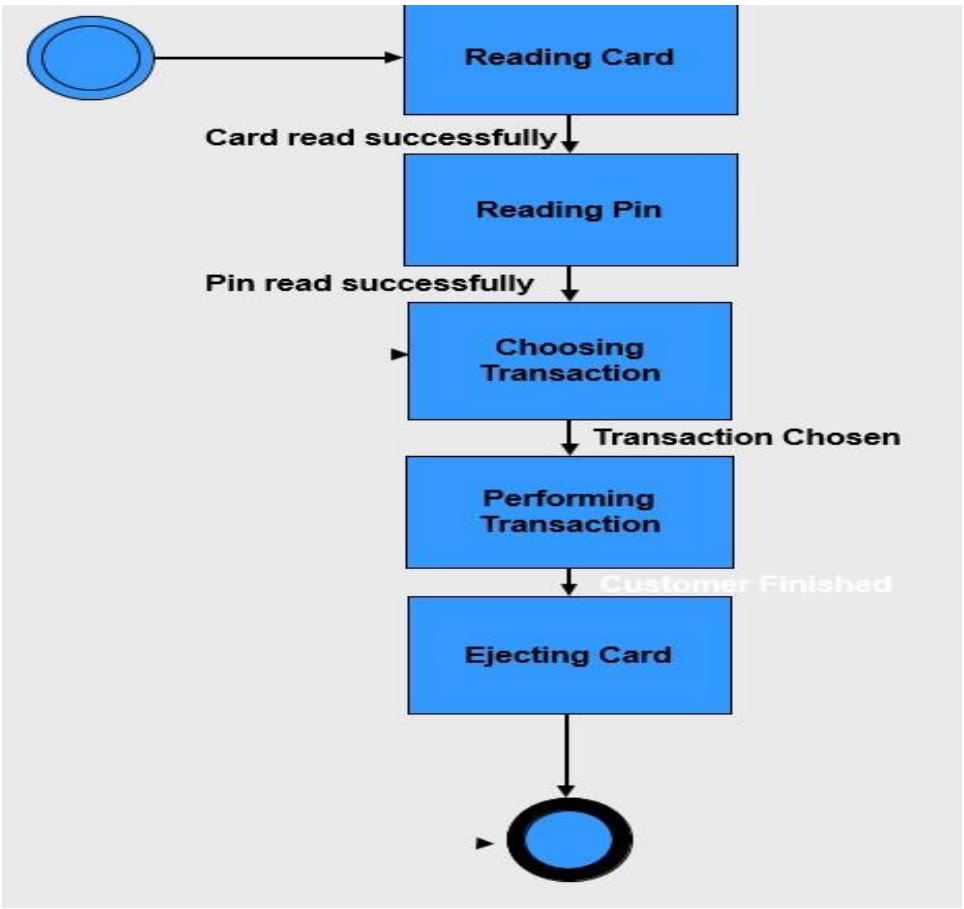


Fig: Collaboration Diagram


Class Diagram
Card Reader
Card number
Accept card()
Eject card()
Read card()
ATM Screen
Prompt()
Acceptinput()
Account
Account number
Eject card
Balance
Open()
Withdraw funds()
Deduct funds()
Verify funds()
Cash Dispenser
Cash balance
Provide Cash()
Provide receipt()


Reading Card
Card read successfully
Reading Pin
Pin read successfully
Choosing Transaction
Transaction Chosen
Performing Transaction
Customer Finished
Ejecting Card