Question Bank

System Analysis and Design

BBA-306-B

Unit-1

Q1. Give Introduction to Analysis and Design?

Q2. What is SDLC. Explain with the help of Diagram?

Q3. Describe Case tools for Analyst?

Q4.Specify role of System Analyst?

Q5. Explain ER data Models?

Q6. What is Feasibility Study?

Q7. What is Economic Feasibility?

Q8. What is Technical Feasibility?

Q9. What is Operational Feasibility?

Unit-2

Q1. What are DFD's?

Q2. What is Form Design?

Q3. What is Screen Design?

Q4. What is Report Design?

Q5. What is structure Chart?

Q6. What is Data Base Definitions?

Q7. What is Equipment Specification?

Q8. What are Personnel Estimates?

Q9. What is I-O Design?

Unit-3

Q1. What is Data Dictionary?

Q2.What are decision tables?

Q3. What are decision Trees?

Q4. What are Education and Training System?

Q5. What is Software Testing

Q6. What is Maintenance and Review?

Unit-4

Q1. What is distributed data Processing?

Q2. What is Real Time System?

Q3. What is the Evaluation of Distributing system?

Q4. How to design distributed data base?

Q5.What is Event based Real time analysis tools?

SYSTEM ANALYSIS AND DESIGN

BBA-306-B

UNIT-1

Introduction to analysis and design

Systems Analysis

It is a process of collecting and interpreting facts, identifying the problems, and decomposition of a system into its components.

System analysis is conducted for the purpose of studying a system or its parts in order to identify its objectives. It is a problem solving technique that improves the system and ensures that all the components of the system work efficiently to accomplish their purpose.

Analysis specifies **what the system should do**.

Systems Design

It is a process of planning a new business system or replacing an existing system by defining its components or modules to satisfy the specific requirements. Before planning, you need to understand the old system thoroughly and determine how computers can best be used in order to operate efficiently.

System Design focuses on **how to accomplish the objective of the system**.

System Analysis and Design (SAD) mainly focuses on −

- Systems
- Processes
- Technology

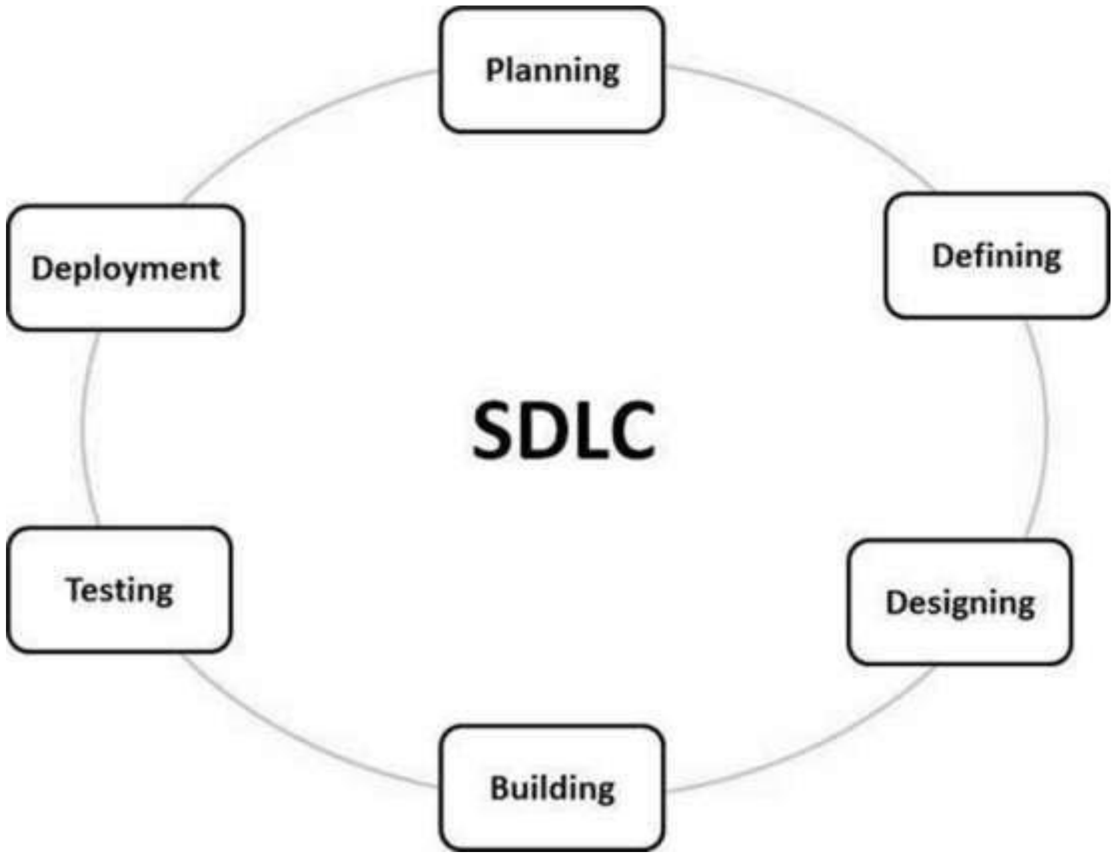
Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality softwares. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

- SDLC is the acronym of Software Development Life Cycle.
- It is also called as Software Development Process.
- SDLC is a framework defining tasks performed at each step in the software development process.
- ISO/IEC 12207 is an international standard for software life-cycle processes. It aims to be the standard that defines all the tasks required for developing and maintaining software.

What is SDLC?

SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.

The following figure is a graphical representation of the various stages of a typical SDLC.



A typical Software Development Life Cycle consists of the following stages − Stage 1:

Planning and Requirement Analysis

Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational and technical areas.

Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.

Stage 2: Defining Requirements

Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through an **SRS (Software Requirement Specification)** document which consists of all the product requirements to be designed and developed during the project life cycle.

Stage 3: Designing the Product Architecture

SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification.

This DDS is reviewed by all the important stakeholders and based on various parameters as risk assessment, product robustness, design modularity, budget and time constraints, the best design approach is selected for the product.

A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third party modules (if any). The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of the details in DDS.

Stage 4: Building or Developing the Product

In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.

Developers must follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers, etc. are used to generate the code. Different high level programming languages such as C, C++, Pascal, Java and PHP are used for coding. The programming language is chosen with respect to the type of software being developed.

Stage 5: Testing the Product

This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However, this stage refers to the testing only stage of the product where product defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS.

Stage 6: Deployment in the Market and Maintenance

Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that organization. The product may first be released in a limited segment and tested in the real business environment (UAT-User acceptance testing).

Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base.

SDLC Models

There are various software development life cycle models defined and designed which are followed during the software development process. These models are also referred as Software Development Process Models". Each process model follows a Series of steps unique to its type to ensure success in the process of software development.

Following are the most important and popular SDLC models followed in the industry −

- Waterfall Model

- Iterative Model
- Spiral Model
- V-Model
- Big Bang Model

CASE TOOLS FOR ANALYST

CASE Tools

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

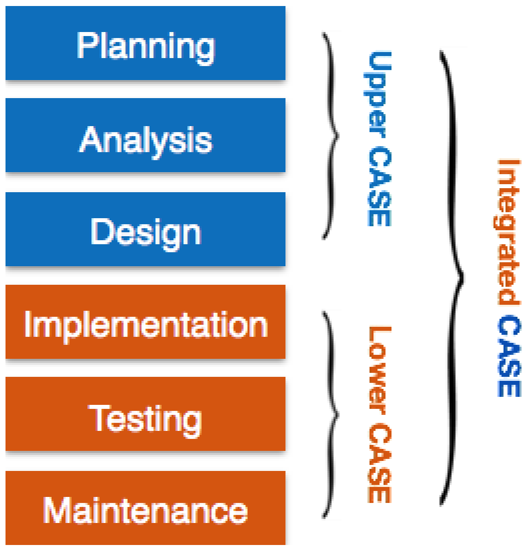
There are number of CASE tools available to simplify various stages of Software Development Life Cycle such as Analysis tools, Design tools, Project management tools, Database Management tools, Documentation tools are to name a few.

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

Components of CASE Tools

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

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



- **Upper Case Tools** - Upper CASE tools are used in planning, analysis and design stages of SDLC.
- **Lower Case Tools** - Lower CASE tools are used in implementation, testing and maintenance.

- **Integrated Case Tools** - Integrated CASE tools are helpful in all the stages of SDLC, from Requirement gathering to Testing and documentation.

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

Scope of Case Tools

The scope of CASE tools goes throughout the SDLC. Case

Tools Types

Now we briefly go through various CASE tools Diagram

tools

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

Process Modeling Tools

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

Project Management Tools

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

Documentation Tools

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

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

Analysis Tools

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

Design Tools

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

Configuration Management Tools

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

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

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

Change Control Tools

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

Programming Tools

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

Prototyping Tools

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

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

Web Development Tools

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

Quality Assurance Tools

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

Maintenance Tools

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

Role of Systems analyst

A **systems analyst** is an information technology (IT) professional who specializes in analyzing, designing and implementing information systems. Systems analysts assess the suitability of information systems in terms of their intended outcomes and liaise with end users, software vendors and programmers in order to achieve these outcomes.[1] A systems analyst is a person who uses analysis and design techniques to solve business problems using information technology. Systems analysts may serve as change agents who identify the organizational improvements needed, design systems to implement those changes, and train and motivate others to use the systems.[2]

Although they may be familiar with a variety of programming languages, operating systems, and computer hardware platforms, they do not normally involve themselves in the actual hardware or software development. They may be responsible for developing cost analysis, design considerations, staff impact amelioration, and implementation timelines.

A systems analyst is typically confined to an assigned or given system and will often work in conjunction with a business analyst. These roles, although having some overlap, are not the same. A business analyst will evaluate the business need and identify the appropriate solution and, to some degree, design a solution without diving too deep into its technical components, relying instead on a systems analyst to do so. A systems analyst will often evaluate and modify code as well as review scripting.

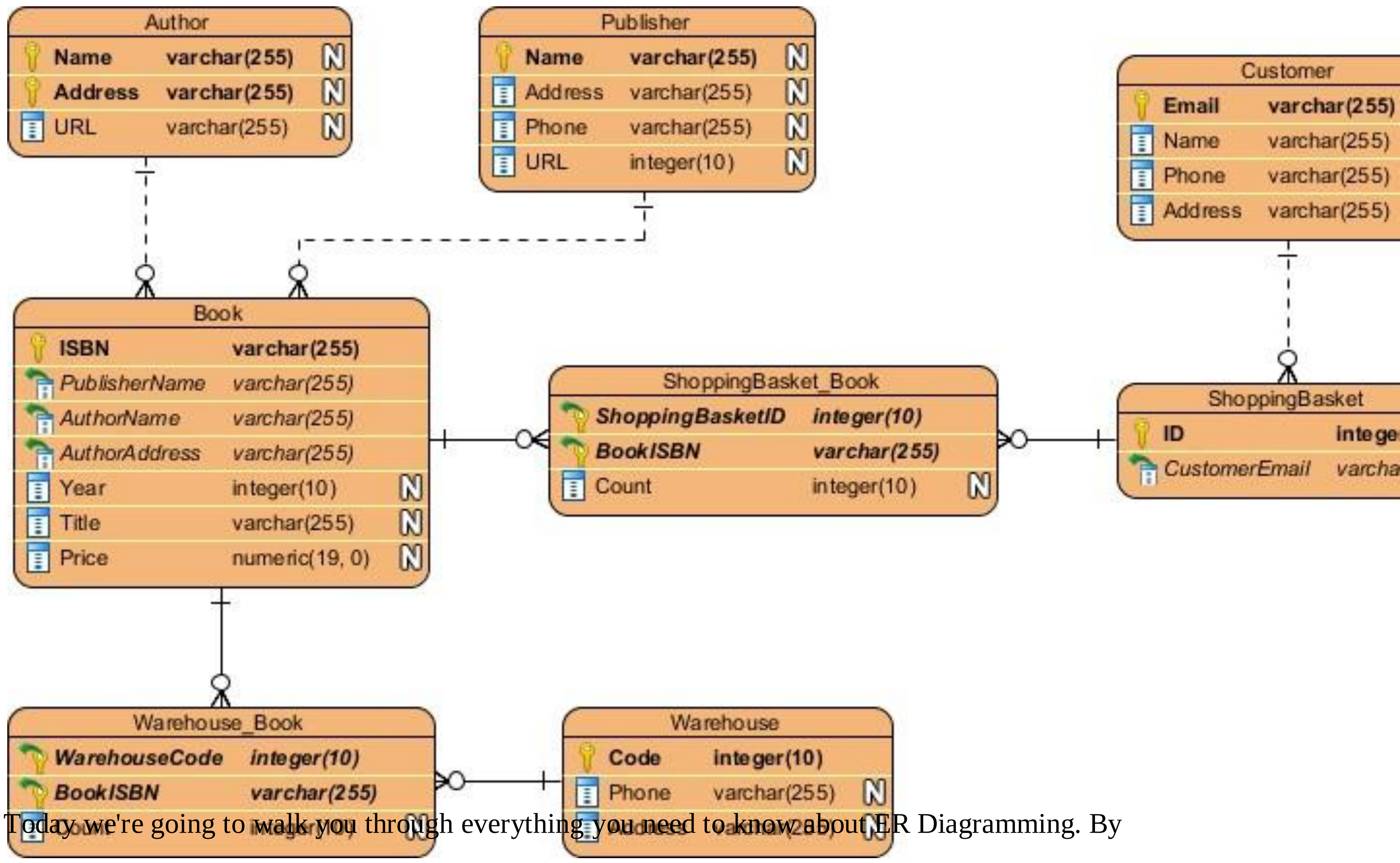
Some dedicated professionals possess practical knowledge in both areas (business and systems analysis) and manage to successfully combine both of these occupations, effectively blurring the line between business analyst and systems analyst.

Roles

- Identify, understand and plan for organizational and human impacts of planned systems, and ensure that new technical requirements are properly integrated with existing processes and skill sets.
- Plan a system flow from the ground up.
- Interact with internal users and customers to learn and document requirements that are then used to produce business required documents.
- Write technical requirements from a critical phase.
- Interact with software architect to understand software limitations.
- Help programmers during system development, e.g. provide use cases, flowcharts, UML and BPMN diagrams.
- Document requirements or contribute to user manuals.
- Whenever a development process is conducted, the system analyst is responsible for designing components and providing that information to the developer.

What is Entity Relationship Diagram (ERD)?

Database is absolutely an integral part of software systems. To fully utilize ER Diagram in database engineering guarantees you to produce high-quality database design to use in database creation, management, and maintenance. An ER model also provides a means for communication.



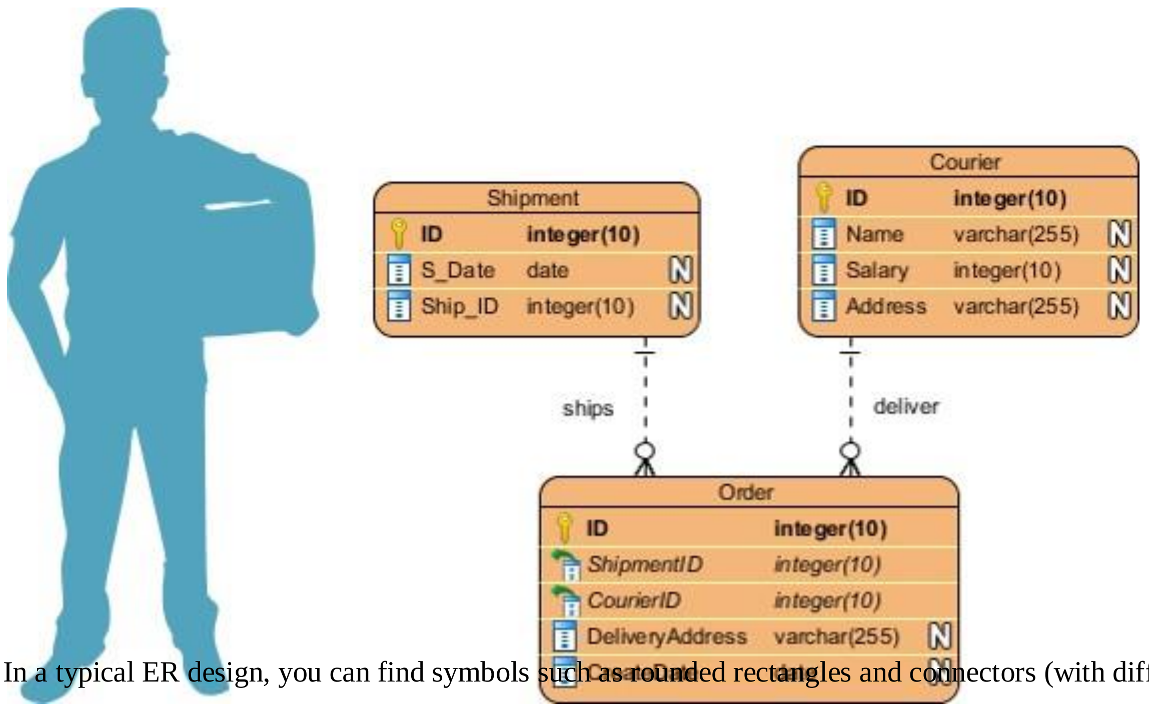
Today we're going to walk you through everything you need to know about ER Diagramming. By reading this ERD guide, you will get the essential knowledge and skills about ER Diagrams and database design. You will learn things like what is ERD, why ERD, ERD notations, how to draw ERD, etc. along with a bunch of ERD examples. What is an ER diagram (ERD)?

First of all, what is an Entity Relationship Diagram?

Entity Relationship Diagram, also known as ERD, ER Diagram or ER model, is a type of structural diagram for use in database design. An ERD contains different symbols and connectors that visualize two important information: **The major entities within the system scope**, and the **inter-relationships among these entities**.

And that's why it's called "Entity" "Relationship" diagram (ERD)!

When we talk about entities in ERD, very often we are referring to business objects such as people/roles (e.g. Student), tangible business objects (e.g. Product), intangible business objects (e.g. Log), etc. "Relationship" is about how these entities relate to each other within the system.



In a typical ER design, you can find symbols such as rounded rectangles and connectors (with different styles of their ends) that depict the entities, their attributes, and inter-relationships.

When to draw ER Diagrams?

So, when do we draw ERDs? While ER models are mostly developed for designing relational databases in terms of concept visualization and in terms of physical database design, there are still other situations when ER diagrams can help. Here are some typical use cases.

- **Database design** - Depending on the scale of change, it can be risky to alter a database structure directly in a DBMS. To avoid ruining the data in a production database, it is important to plan out the changes carefully. ERD is a tool that helps. By drawing ER diagrams to visualize database design ideas, you have a chance to identify the mistakes and design flaws, and to make corrections before executing the changes in the database.
- **Database debugging** - To debug database issues can be challenging, especially when the database contains many tables, which require writing complex SQL in getting the information you need. By visualizing a database schema with an ERD, you have a full picture of the entire database schema. You can easily locate entities, view their attributes and identify the relationships they have with others. All these allow you to analyze an existing database and to reveal database problems easier.
- **Database creation and patching** - Visual Paradigm, an ERD tool, supports a database generation tool that can automate the database creation and patching process by means of ER diagrams. So, with this ER Diagram tool, your ER design is no longer just a static diagram but a mirror that reflects truly the physical database structure.

- **Aid in requirements gathering** - Determine the requirements of an information system by drawing a conceptual ERD that depicts the high-level business objects of the system. Such an initial model can also be evolved into a physical database model that aids the creation of a relational database, or aids in the creation of process maps and data flow modes.

FEASIBILITY STUDY

Feasibility study

A **feasibility study** is an assessment of the practicality of a proposed project or system. A feasibility study aims to objectively and rationally uncover the strengths and weaknesses of an existing business or proposed venture, opportunities and threats present in the natural environment, the resources required to carry through, and ultimately the prospects for success. In its simplest terms, the two criteria to judge feasibility are cost required and value to be attained.

A well-designed feasibility study should provide a historical background of the business or project, a description of the product or service, accounting statements, details of the operations and management, marketing research and policies, financial data, legal requirements and tax obligations. Generally, feasibility studies precede technical development and project implementation. A feasibility study evaluates the project's potential for success; therefore, perceived objectivity is an important factor in the credibility of the study for potential investors and lending institutions. It must therefore be conducted with an objective, unbiased approach to provide information upon which decisions can be based.

Technical feasibility

This assessment is based on an outline design of system requirements, to determine whether the company has the technical expertise to handle completion of the project.When writing a feasibility report, the following should be taken to consideration:

- A brief description of the business to assess more possible factors which could affect the study
- The part of the business being examined
- The human and economic factor
- The possible solutions to the problem

At this level, the concern is whether the proposal is both *technically* and *legally* feasible (assuming moderate cost).

The technical feasibility assessment is focused on gaining an understanding of the present technical resources of the organization and their applicability to the expected needs of the proposed system. It is an evaluation of the hardware and software and how it meets the need of the proposed system

Method of production

The selection among a number of methods to produce the same commodity should be undertaken first. Factors that make one method being preferred to other method in agricultural projects are the following:

- Availability of inputs or raw materials and their quality and prices.

- Availability of markets for outputs of each method and the expected prices for these outputs.
- Various efficiency factors such as the expected increase in one additional unit of fertilizer or productivity of a specified crop per one thing

Production technique

After we determine the appropriate method of production of a commodity, it is necessary to look for the optimal technique to produce this commodity.

Project requirements

Once the method of production and its technique are determined, technical people have to determine the projects' requirements during the investment and operating periods. These include:

- Determination of tools and equipment needed for the project such as drinkers and feeders or pumps or pipes …etc.
- Determination of projects' requirements of constructions such as buildings, storage, and roads …etc. in addition to internal designs for these requirements.
- Determination of projects' requirements of skilled and unskilled labor and managerial and financial labor.
- Determination of construction period concerning the costs of designs and consultations and the costs of constructions and other tools.
- Determination of minimum storage of inputs, cash money to cope with operating and contingency costs.

Project location

The most important factors that determine the selection of project location are the following:

- Availability of land (proper acreage and reasonable costs).
- The impact of the project on the environment and the approval of the concerned institutions for license.
- The costs of transporting inputs and outputs to the project's location (i.e., the distance from the markets).
- Availability of various services related to the project such as availability of extension services or veterinary or water or electricity or good roads ...etc.

Legal feasibility

It determines whether the proposed system conflicts with legal requirements, e.g., a data processing system must comply with the local data protection regulations and if the proposed venture is acceptable in accordance to the laws of the land.

Operational feasibility study

Operational feasibility is the measure of how well a proposed system solves the problems, and takes advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development.

The operational feasibility assessment focuses on the degree to which the proposed development project fits in with the existing business environment and objectives with regard to development schedule, delivery date, corporate culture and existing business processes.

To ensure success, desired operational outcomes must be imparted during design and development. These include such design-dependent parameters as reliability, maintainability, supportability, usability, producibility, disposability, sustainability, affordability and others.

These parameters are required to be considered at the early stages of design if desired operational behaviours are to be realised. A system design and development requires appropriate and timely application of engineering and management efforts to meet the previously mentioned parameters. A system may serve its intended purpose most effectively when its technical and operating characteristics are engineered into the design. Therefore, operational feasibility is a critical aspect of systems engineering that needs to be an integral part of the early design phases.[11]

Time feasibility

A time feasibility study will take into account the period in which the project is going to take up to its completion. A project will fail if it takes too long to be completed before it is useful.
Typically this means estimating how long the system will take to develop, and if it can be completed in a given time period using some methods like payback period. Time feasibility is a measure of how reasonable the project timetable is. Given our technical expertise, are the project deadlines reasonable? Some projects are initiated with specific deadlines. It is necessary to determine whether the deadlines are mandatory or desirable

Resource feasibility

Describe how much time is available to build the new system, when it can be built, whether it interferes with normal business operations, type and amount of resources required, dependencies, and developmental procedures with company revenue prospectus.

Financial feasibility

In case of a new project, financial viability can be judged on the following parameters:

- Total estimated cost of the project
- Financing of the project in terms of its capital structure, debt to equity ratio and promoter's share of total cost
- Existing investment by the promoter in any other business
- Projected cash flow and profitability

The financial viability of a project should provide the following information:[12]

- Full details of the assets to be financed and how liquid those assets are.
- Rate of conversion to cash-liquidity (i.e., how easily the various assets can be converted to cash).
- Project's funding potential and repayment terms.
- Sensitivity in the repayments capability to the following factors:
  - Mild slowing of sales.
  - Acute reduction/slowing of sales.
  - Small increase in cost.
  - Large increase in cost.
  - Adverse economic conditions.

In 1983 the first generation of the Computer Model for Feasibility Analysis and Reporting (COMFAR), a computation tool for financial analysis of investments, was released. Since then, this United Nations Industrial Development Organization (UNIDO) software has been developed to also support the economic appraisal of projects. The COMFAR III Expert is intended as an aid in the analysis of investment projects. The main module of the program accepts financial and economic data, produces financial and economic statements and graphical displays and calculates measures of performance. Supplementary modules assist in the analytical process.

Cost-benefit and value-added methods of economic analysis developed by UNIDO are included in the program and the methods of major international development institutions are accommodated. The program is applicable for the analysis of investment in new projects and expansion or rehabilitation of existing enterprises as, e.g., in the case of re privatization projects. For joint ventures, the financial perspective of each partner or class of shareholder can be developed. Analysis can be performed under a variety of assumptions concerning inflation, currency revaluation and price escalations.

UNIT-2

What is Data Flow Diagram (DFD)? How to Draw DFD?

What is a data flow diagram (DFD)?

A picture is worth a thousand words. A Data Flow Diagram (DFD) is a traditional way to visualize the information flows within a system. A neat and clear DFD can depict a good amount of the system requirements graphically. It can be manual, automated, or a combination of both.

It shows how information enters and leaves the system, what changes the information and where information is stored. The purpose of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communications tool between a systems analyst and any person who plays a part in the system that acts as the starting point for redesigning a system.

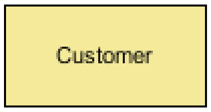
It is usually beginning with a context diagram as level 0 of the DFD diagram, a simple representation of the whole system. To elaborate further from that, we drill down to a level 1 diagram with lower-level functions decomposed from the major functions of the system. This could continue to evolve to become a level 2 diagram when further analysis is required.
Progression to levels 3, 4 and so on is possible but anything beyond level 3 is not very common. Please bear in mind that the level of detail for decomposing a particular function depending on the complexity that function.

DFD Diagram Notations

Now we'd like to briefly introduce to you a few diagram notations which you'll see in the tutorial below.

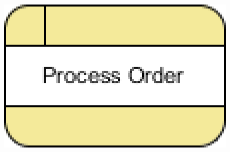
External Entity

An external entity can represent a human, system or subsystem. It is where certain data comes from or goes to. It is external to the system we study, in terms of the business process. For this reason, people used to draw external entities on the edge of a diagram.



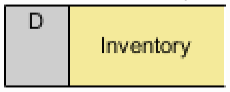
Process

A process is a business activity or function where the manipulation and transformation of data take place. A process can be decomposed to a finer level of details, for representing how data is being processed within the process.



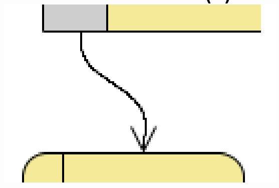
Data Store

A data store represents the storage of persistent data required and/or produced by the process. Here are some examples of data stores: membership forms, database tables, etc.



Data Flow

A data flow represents the flow of information, with its direction represented by an arrowhead that shows at the end(s) of flow connector.



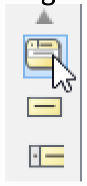
What will we do in this tutorial?

In this tutorial, we will show you how to draw a context diagram, along with a level 1 diagram.

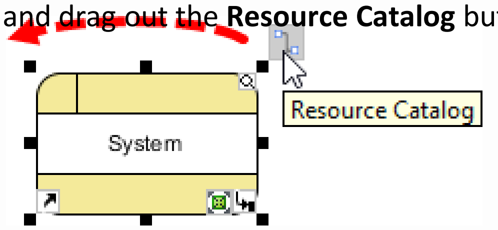
Note: The software we are using here is Visual Paradigm. You are welcome to download a **free 30-day** evaluation copy of Visual Paradigm to walk through the example below. No registration, email address or obligation is required.

How to Draw Context Level DFD?

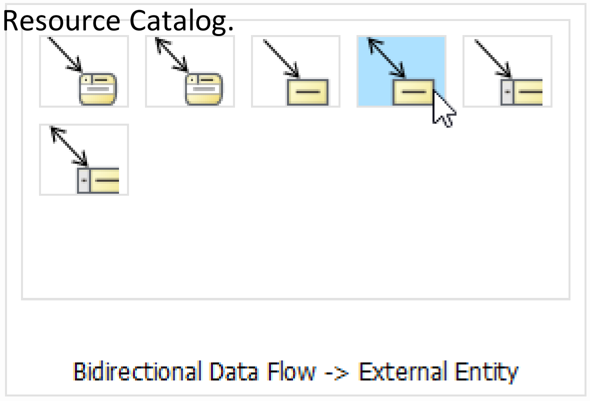
1. To create new DFD, select **Diagram > New** from the toolbar.
2. In the **New Diagram** window, select **Data Flow Diagram** and click **Next**.
3. Enter *Context* as diagram name and click **OK** to confirm.
4. We'll now draw the first process. From the Diagram Toolbar, drag **Process** onto the diagram. Name the new process *System*.



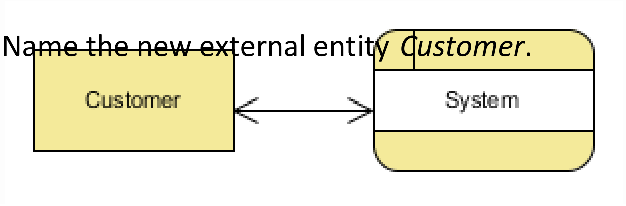
5. Next, let's create an external entity. Place your mouse pointer over *System*. Press and drag out the **Resource Catalog** button at the top right.



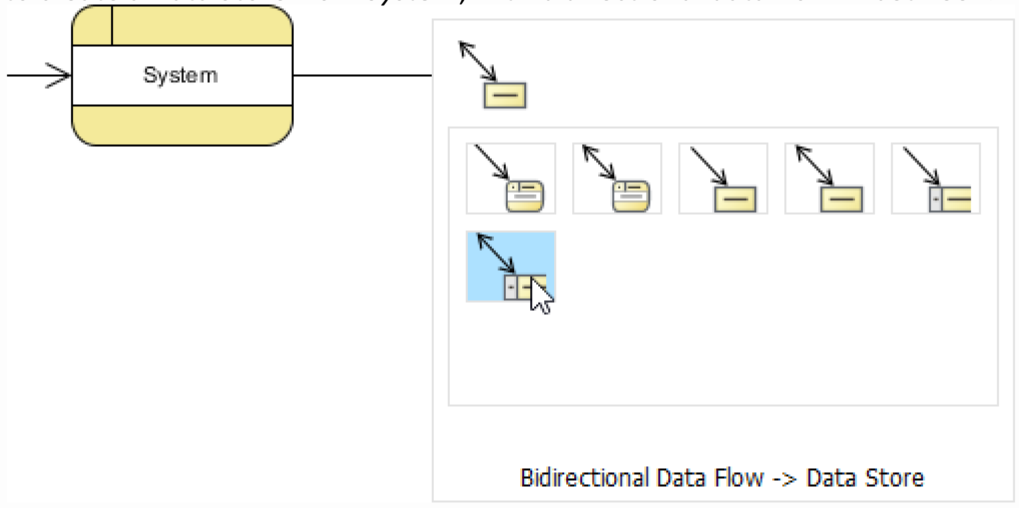
6. Release the mouse button and select **Bidirectional Data Flow -> External Entity** from Resource Catalog.



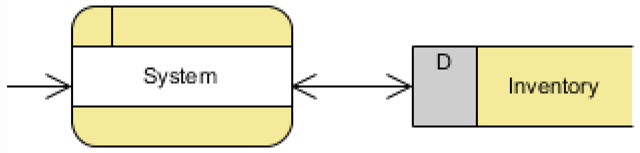
7. Name the new external entity *Customer*.



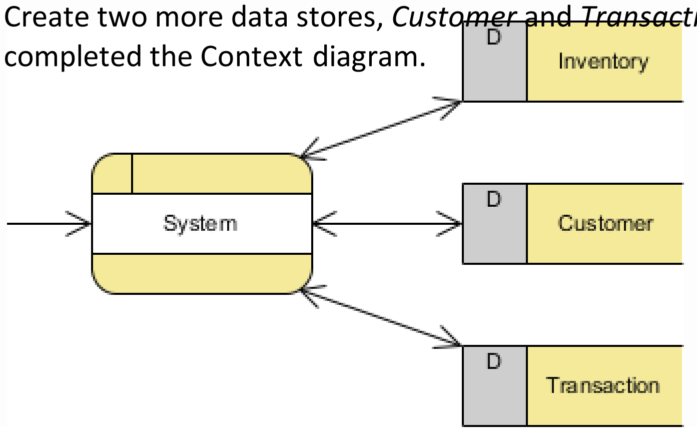
8. Now we'll model the database accessed by the system. Use the Resource Catalog to create a Data Store from *System*, with bidirectional data flow in between.



9. Name the new data store *Inventory*.



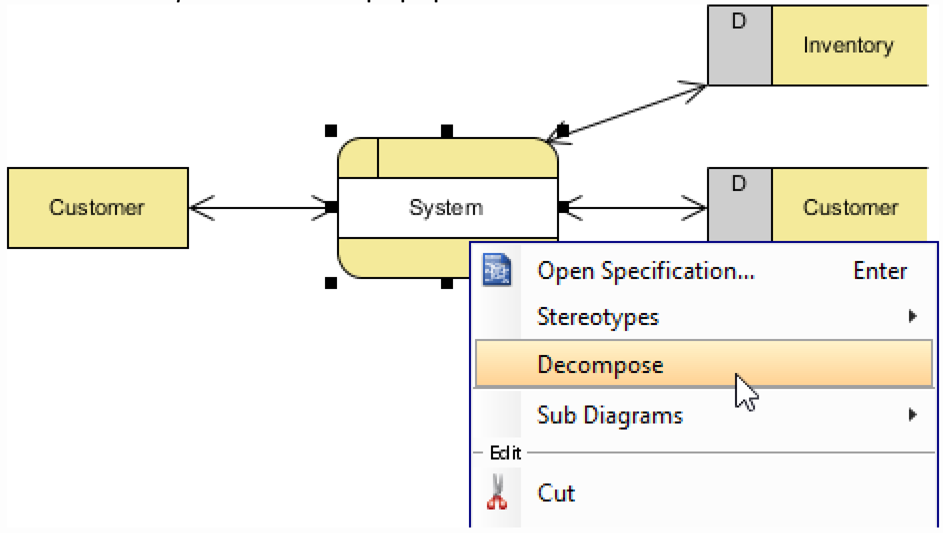
10. Create two more data stores, *Customer* and *Transaction*, as shown below. We have just completed the Context diagram.



How to Draw Level 1 DFD?

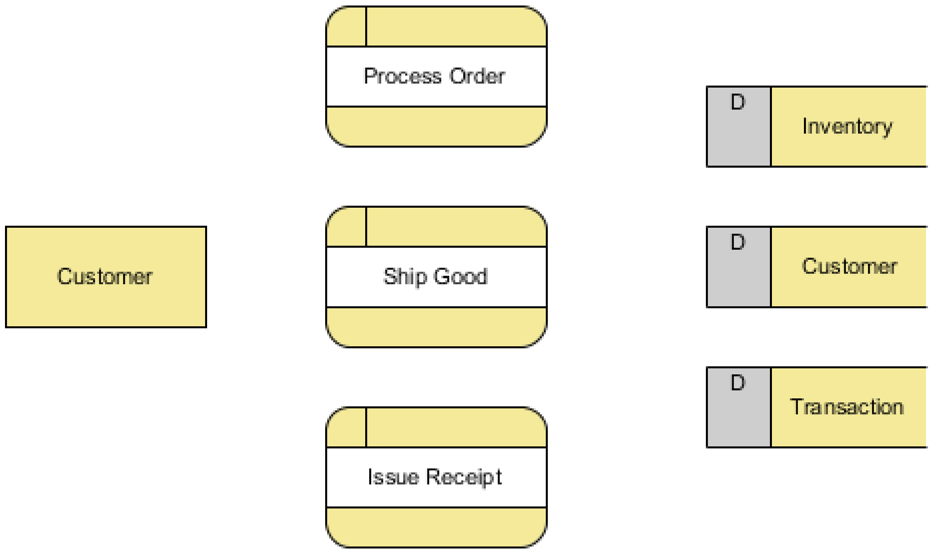
1. Instead of creating another diagram from scratch, we will decompose the *System* process to form a new DFD. Right-click on *System* and

select *Decompose* from the popup menu.



2. The data stores and/or external entities connected to the selected process (*System*) would be referred to in the level 1 DFD. So when you are prompted to add them to the new diagram, click **Yes** to confirm.
   Note: The new DFD should look very similar to the Context diagram initially. Every element should remain unchanged, except that the *System* process (from which this new DFD decomposes) is now gone and replaced by a blank space (to be elaborated).
3. Rename the new DFD. Right-click on its background and select **Rename......** In the diagram's name box, enter *Level 1 DFD* and press **ENTER**.
4. Create three processes (Process Order, Ship Good, Issue Receipt) in the center as shown below. That is the old spot for the *System* process and we place them there.

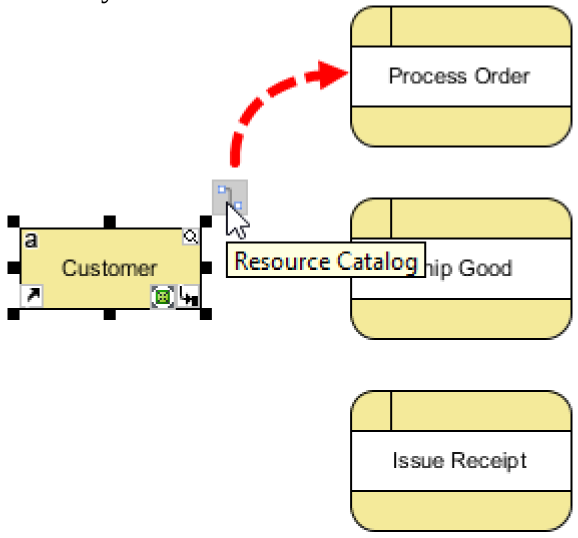
1.



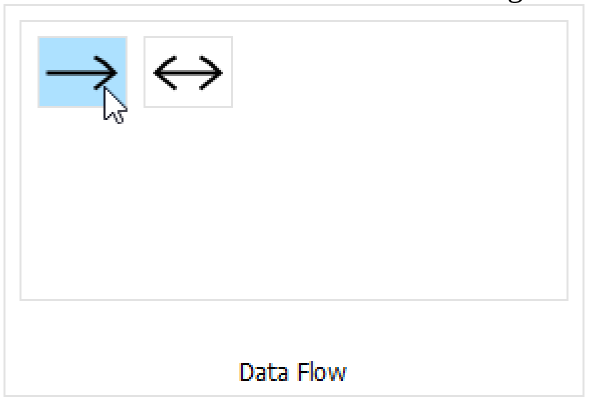
Wiring with connection lines for data flows

The remaining steps in this section are about connecting the model elements in the diagram. For example, *Customer* provides order information when placing an order for processing.

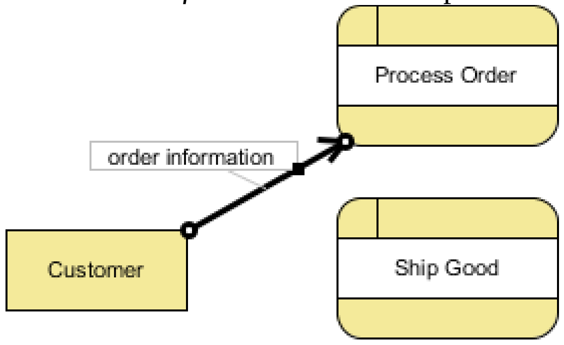
1. Place your mouse pointer over *Customer*. Drag out the **Resource Catalog** icon and release your mouse button on *Process Order*.



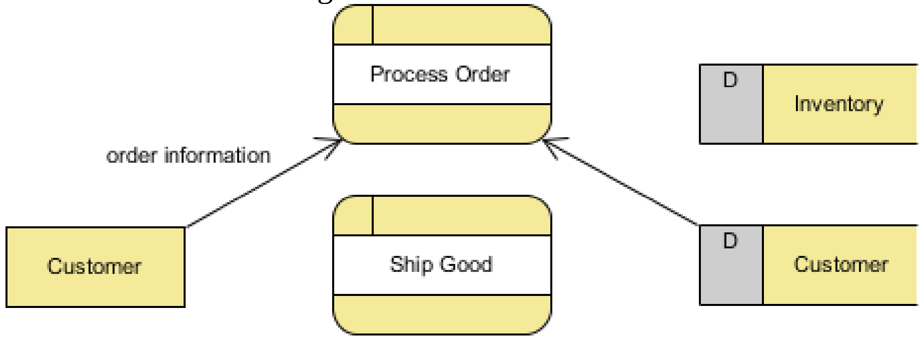
2. Select Data Flow from Resource Catalog.



3. Enter *order information* has the caption of flow.

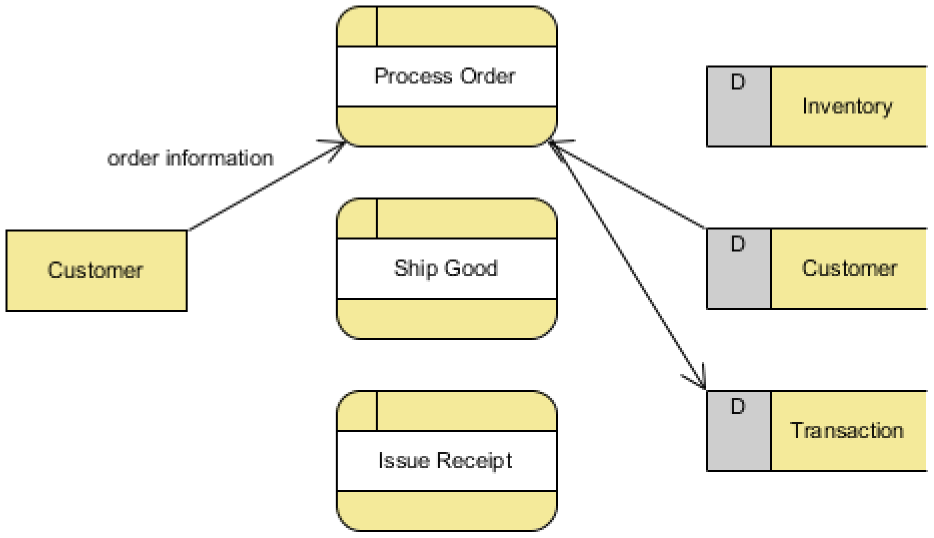


4. Meanwhile, the **Process Order** process also receives customer information from the database to process the order.
Use the Resource Catalog to create a data flow from *Customer* to *Process Order*.

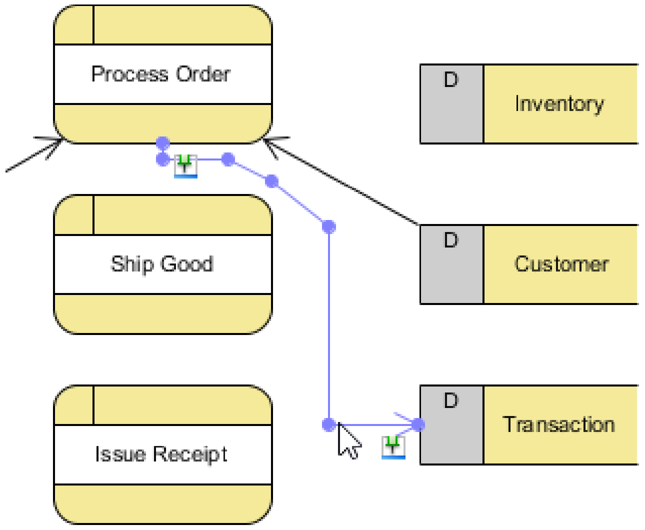


Optional: You can label the data flow "customer information" if you like. But since this data flow is quite self-explanatory visually, we are going to omit it here.

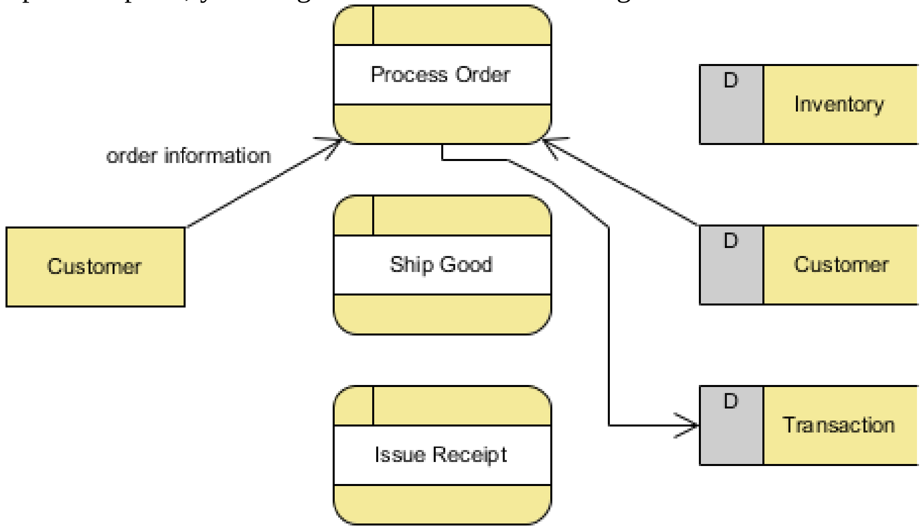
5. By combining the order information from *Customer* (external entity) and the customer information from *Customer* (data store), *Process Order* (process) then creates a transaction record in the database. Create a data flow from *Process Order* to *Transaction*.



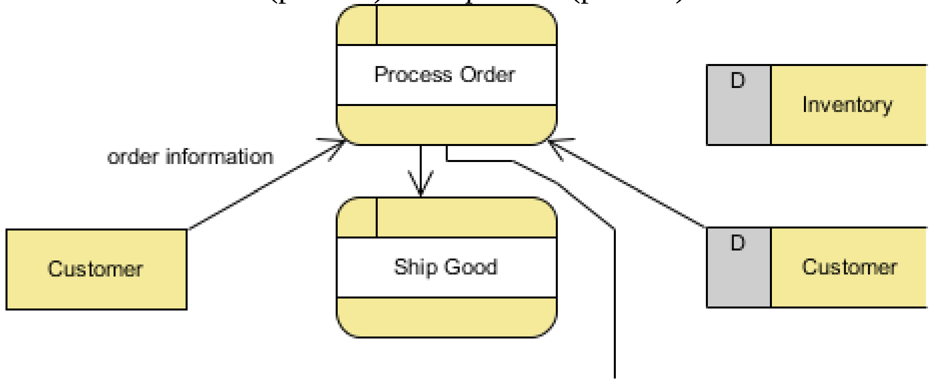
Drawing Tips:
To rearrange a connection line, place your mouse pointer over where you want to add a pivot point to it. You'll then see a bubble at your mouse pointer. Press and drag it to the position desired.



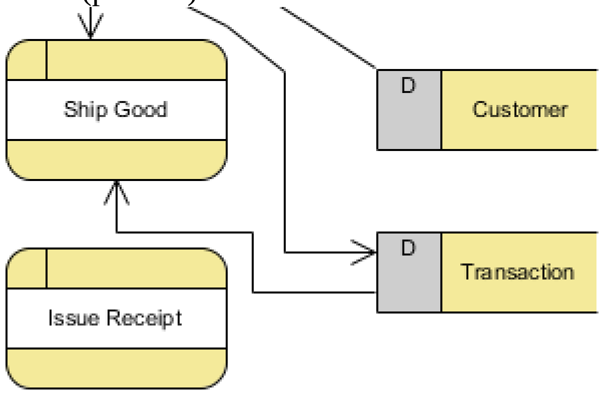
Up to this point, your diagram should look something like this.



6. Once a transaction is stored, the shipping process follows. Therefore, create a data flow from *Process Order* (process) to *Ship Good* (process).

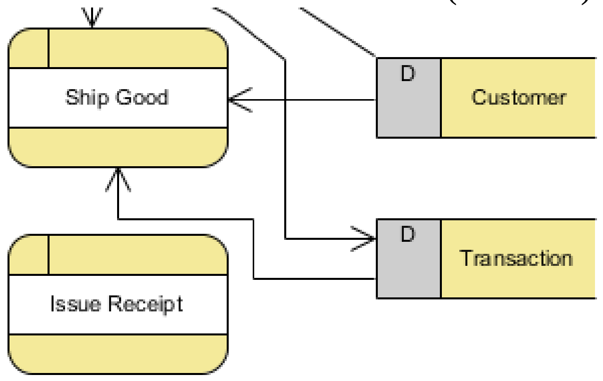


7. Ship Good needs to read the transaction information (i.e. The order number to pack the right product for delivery. Create a data flow from *Transaction* (data store) to *Ship Good* (process).

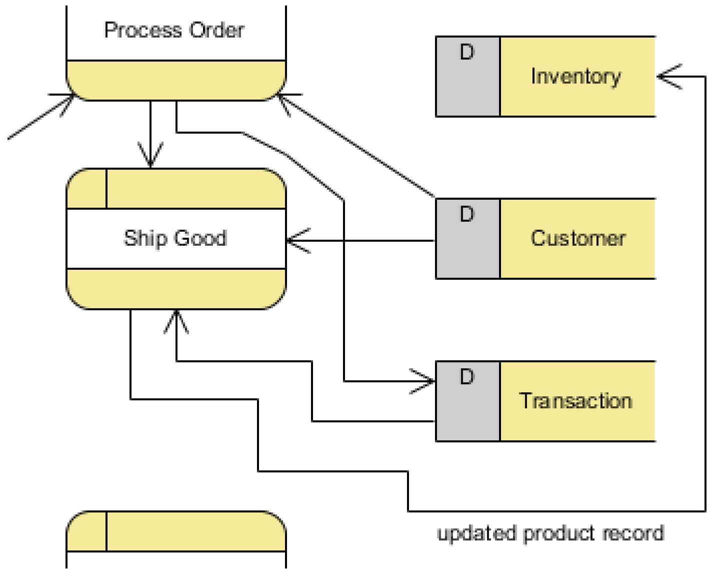


Note: If there is a lack of space, feel free to move the shapes around to make room.

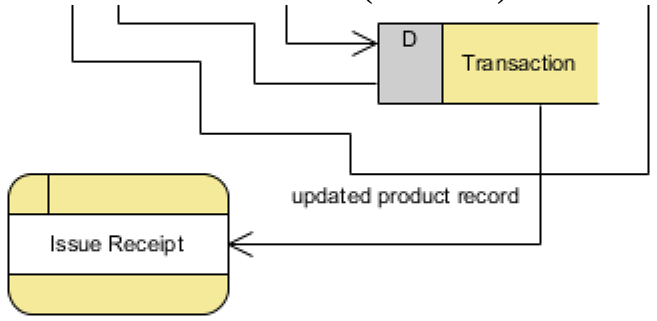
8. *Ship Good* also needs to read the customer information for his/her shipping address. Create a data flow from *Customer* (data store) to *Ship Good* (process).



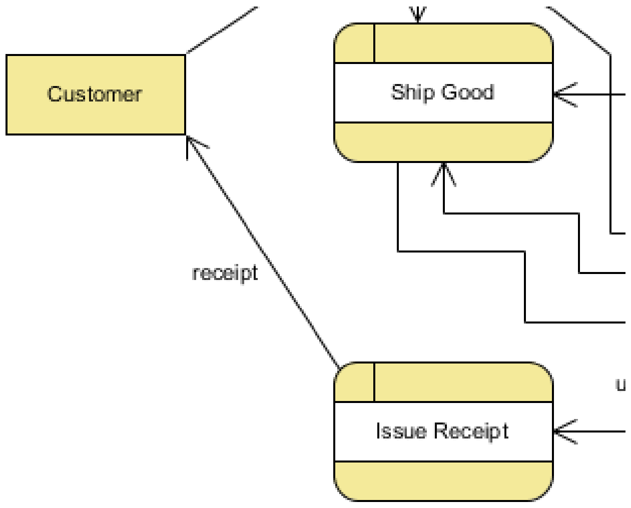
9. *Ship Good* then updates the *Inventory* database to reflect the goods shipped. Create a data flow from *Ship Good* (process) to *Inventory* (data store). Name it *updated product record*.



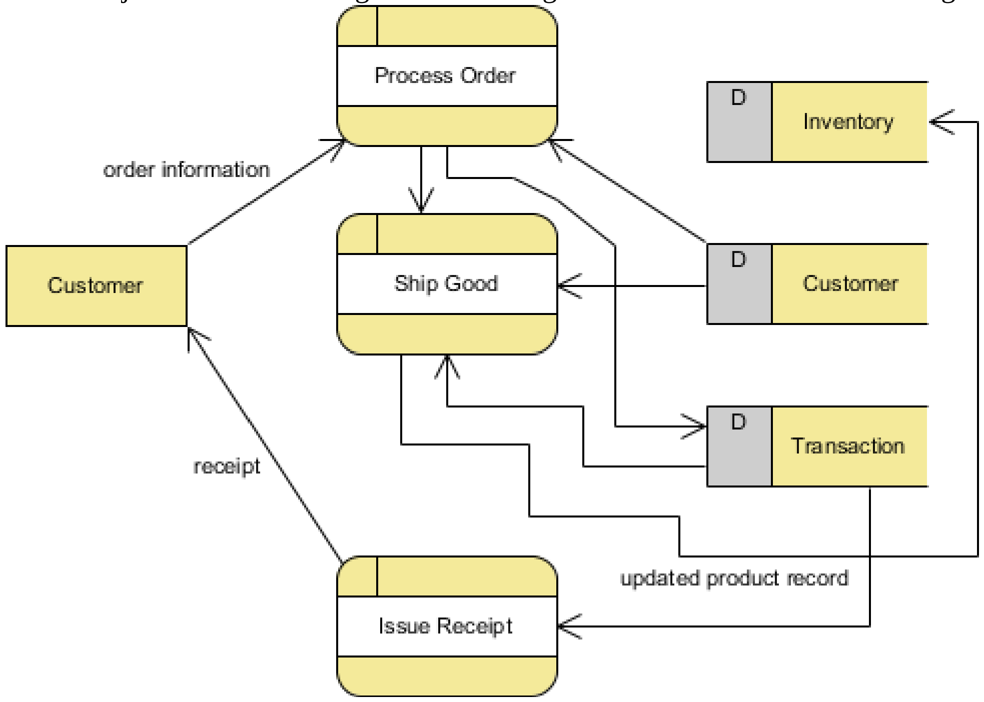
10. Once the order arrives in the customer's hands, the *Issue Receipt* process begins. In it, a receipt is prepared based on the transaction record stored in the database. So let's create a data flow from *Transaction* (data store) to *Issue Receipt* (process).



11. Then a receipt is issued to the customer. Let's create a data flow from *Issue Receipt* (process) to *Customer* (external entity). Name the data flow *receipt*.



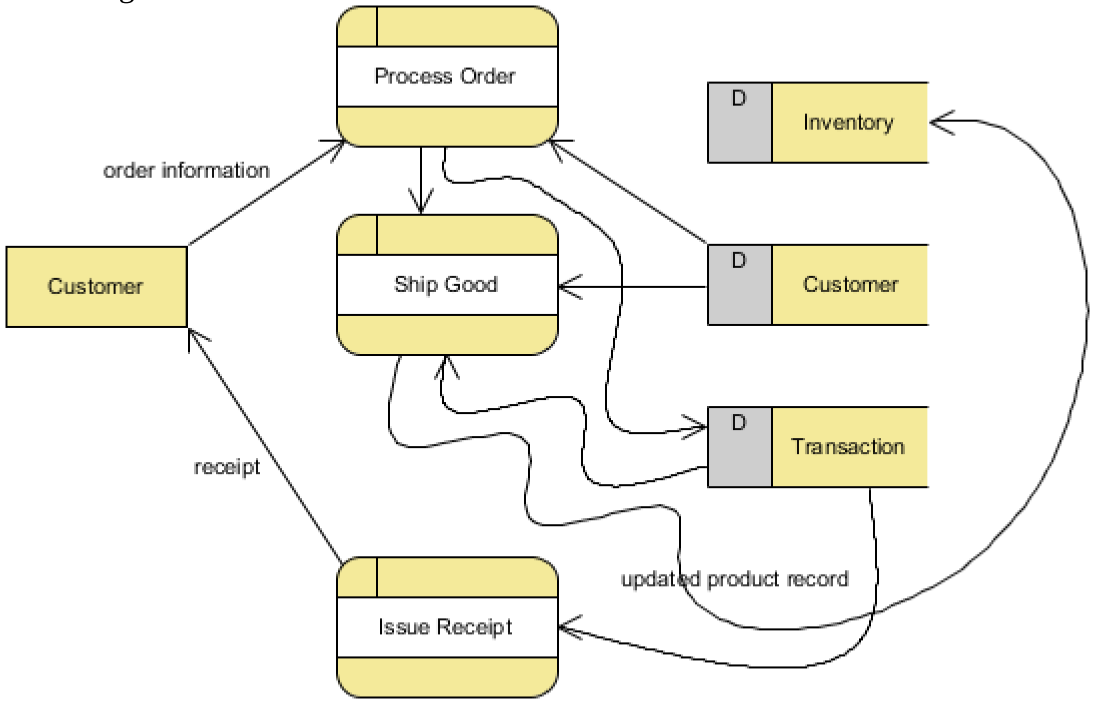
You have just finished drawing the level 1 diagram which should look something like this.



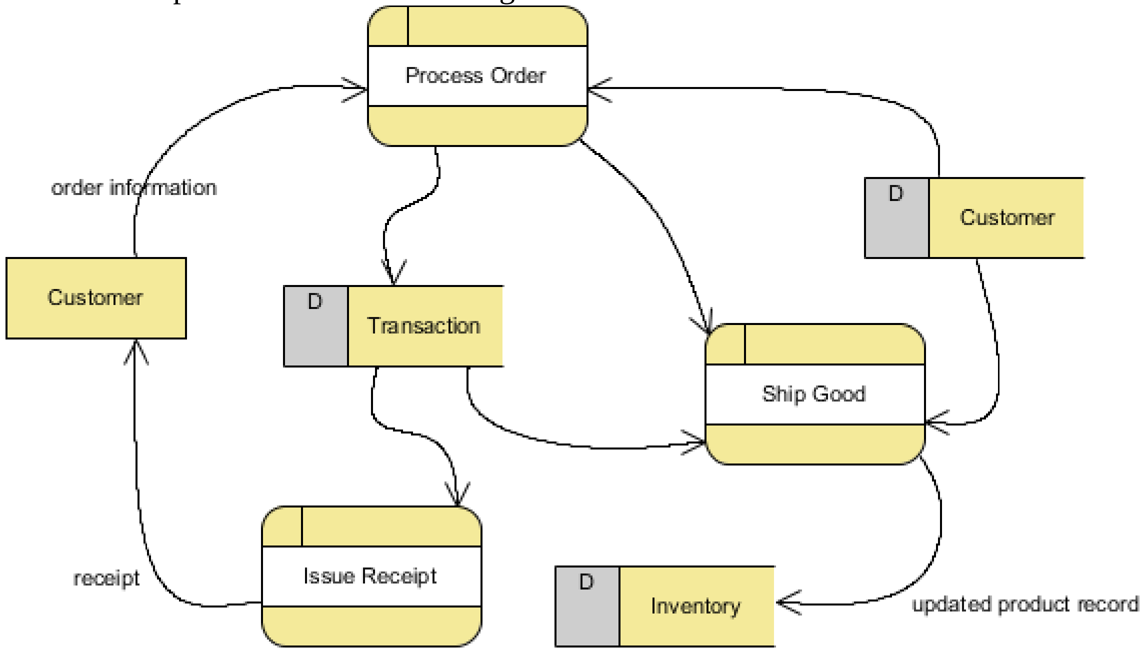
How to Improve a DFD's Readability?

The completed diagram above looks a bit rigid and busy. In this section, we are going to make some changes to the connectors to increase readability.

1. Right-click on the diagram (Level 1 DFD) and select **Connectors** > **Curve**. Connectors in the diagram are now in curve lines.



2. Move the shapes around so that the diagram looks less crowded.



Forms Design

Both forms and reports are the product of input and output design and are business document consisting of specified data. The main difference is that forms provide fields for data input but reports are purely used for reading. For example, order forms, employment and credit application, etc.

- During form designing, the designers should know −
  - who will use them

  - where would they be delivered
  - the purpose of the form or report
- During form design, automated design tools enhance the developer's ability to prototype forms and reports and present them to end users for evaluation.

Objectives of Good Form Design

A good form design is necessary to ensure the following −

- To keep the screen simple by giving proper sequence, information, and clear captions.
- To meet the intended purpose by using appropriate forms.
- To ensure the completion of form with accuracy.
- To keep the forms attractive by using icons, inverse video, or blinking cursors etc.
- To facilitate navigation.

Types of Forms

Flat Forms

- It is a single copy form prepared manually or by a machine and printed on a paper. For additional copies of the original, carbon papers are inserted between copies.
- It is a simplest and inexpensive form to design, print, and reproduce, which uses less volume.

Unit Set/Snap out Forms

- These are papers with one-time carbons interleaved into unit sets for either handwritten or machine use.
- Carbons may be either blue or black, standard grade medium intensity. Generally, blue carbons are best for handwritten forms while black carbons are best for machine use.

Continuous strip/Fanfold Forms

- These are multiple unit forms joined in a continuous strip with perforations between each pair of forms.
- It is a less expensive method for large volume use.

No Carbon Required (NCR) Paper

- They use carbonless papers which have two chemical coatings (capsules), one on the face and the other on the back of a sheet of paper.
- When pressure is applied, the two capsules interact and create an image.

Screen Design

User interface is the front-end application view to which user interacts in order to use the software. User can manipulate and control the software as well as hardware by means of user interface. Today, user interface is found at almost every place where digital technology exists, right from computers, mobile phones, cars, music players, airplanes, ships etc.

User interface is part of software and is designed such a way that it is expected to provide the user insight of the software. UI provides fundamental platform for human-computer interaction.

UI can be graphical, text-based, audio-video based, depending upon the underlying hardware and software combination. UI can be hardware or software or a combination of both.

The software becomes more popular if its user interface is:

- Attractive
- Simple to use
- Responsive in short time
- Clear to understand
- Consistent on all interfacing screens

UI is broadly divided into two categories:

- Command Line Interface
- Graphical User Interface

Command Line Interface (CLI)

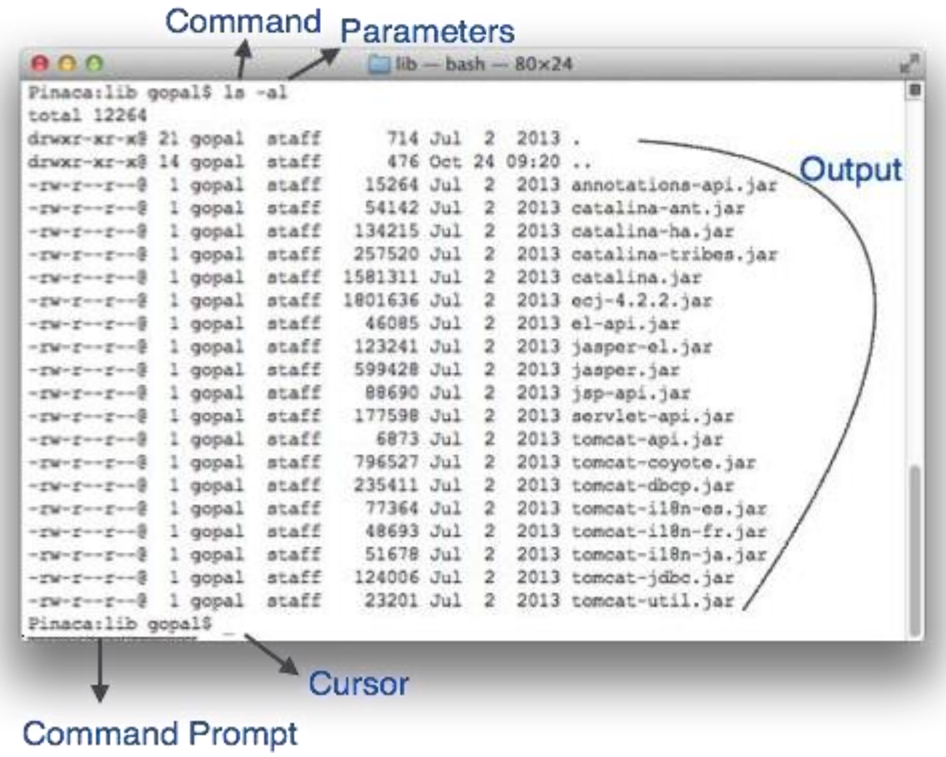
CLI has been a great tool of interaction with computers until the video display monitors came into existence. CLI is first choice of many technical users and programmers. CLI is minimum interface a software can provide to its users.

CLI provides a command prompt, the place where the user types the command and feeds to the system. The user needs to remember the syntax of command and its use. Earlier CLI were not programmed to handle the user errors effectively.

A command is a text-based reference to set of instructions, which are expected to be executed by the system. There are methods like macros, scripts that make it easy for the user to operate.

CLI uses less amount of computer resource as compared to GUI.

CLI Elements



```
Pinaca:lib gopal$ ls -al
total 12264
drwxr-xr-x@ 21 gopal  staff      714 Jul  2  2013 .
drwxr-xr-x@ 14 gopal  staff      476 Oct 24 09:20 ..
-rw-r--r--@  1 gopal  staff    15264 Jul  2  2013 annotations-api.jar
-rw-r--r--@  1 gopal  staff    54142 Jul  2  2013 catalina-ant.jar
-rw-r--r--@  1 gopal  staff   134215 Jul  2  2013 catalina-ha.jar
-rw-r--r--@  1 gopal  staff   257520 Jul  2  2013 catalina-tribes.jar
-rw-r--r--@  1 gopal  staff  1581311 Jul  2  2013 catalina.jar
-rw-r--r--@  1 gopal  staff  1801636 Jul  2  2013 ecj-4.2.2.jar
-rw-r--r--@  1 gopal  staff    46085 Jul  2  2013 el-api.jar
-rw-r--r--@  1 gopal  staff   123241 Jul  2  2013 jasper-el.jar
-rw-r--r--@  1 gopal  staff   599428 Jul  2  2013 jasper.jar
-rw-r--r--@  1 gopal  staff    88690 Jul  2  2013 jsp-api.jar
-rw-r--r--@  1 gopal  staff   177598 Jul  2  2013 servlet-api.jar
-rw-r--r--@  1 gopal  staff     6873 Jul  2  2013 tomcat-api.jar
-rw-r--r--@  1 gopal  staff   796527 Jul  2  2013 tomcat-coyote.jar
-rw-r--r--@  1 gopal  staff   235411 Jul  2  2013 tomcat-dbcp.jar
-rw-r--r--@  1 gopal  staff    77364 Jul  2  2013 tomcat-i18n-es.jar
-rw-r--r--@  1 gopal  staff    48693 Jul  2  2013 tomcat-i18n-fr.jar
-rw-r--r--@  1 gopal  staff    51678 Jul  2  2013 tomcat-i18n-ja.jar
-rw-r--r--@  1 gopal  staff   124006 Jul  2  2013 tomcat-jdbc.jar
-rw-r--r--@  1 gopal  staff    23201 Jul  2  2013 tomcat-util.jar
Pinaca:lib gopal$ _
```

A text-based command line interface can have the following elements:

- **Command Prompt** - It is text-based notifier that is mostly shows the context in which the user is working. It is generated by the software system.
- **Cursor** - It is a small horizontal line or a vertical bar of the height of line, to represent position of character while typing. Cursor is mostly found in blinking state. It moves as the user writes or deletes something.
- **Command** - A command is an executable instruction. It may have one or more parameters. Output on command execution is shown inline on the screen. When output is produced, command prompt is displayed on the next line.

Graphical User Interface

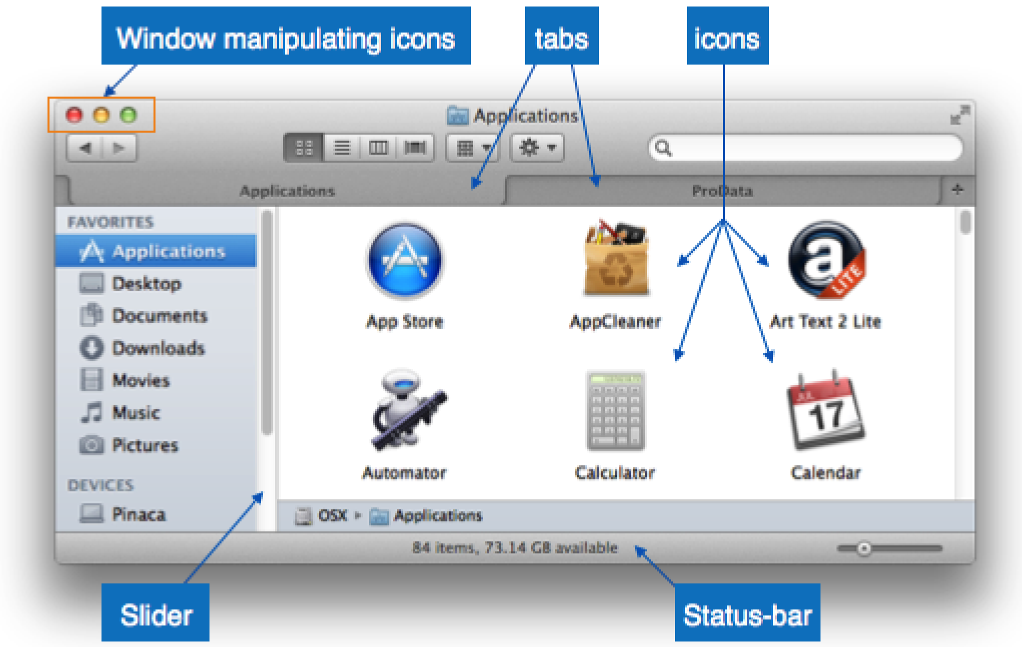
Graphical User Interface provides the user graphical means to interact with the system. GUI can be combination of both hardware and software. Using GUI, user interprets the software.

Typically, GUI is more resource consuming than that of CLI. With advancing technology, the programmers and designers create complex GUI designs that work with more efficiency, accuracy and speed.

GUI Elements

GUI provides a set of components to interact with software or hardware.

Every graphical component provides a way to work with the system. A GUI system has following elements such as:



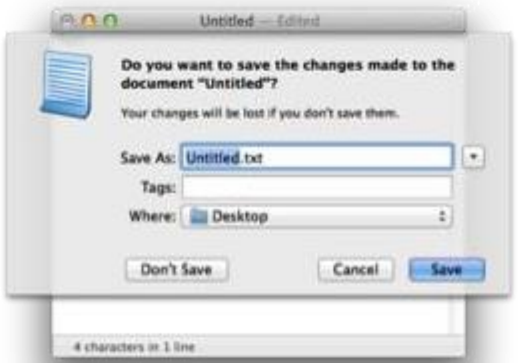
- **Window** - An area where contents of application are displayed. Contents in a window can be displayed in the form of icons or lists, if the window represents file structure. It is easier for a user to navigate in the file system in an exploring window. Windows can be minimized, resized or maximized to the size of screen. They can be moved anywhere on the screen. A window may contain another window of the same application, called child window.
- **Tabs** - If an application allows executing multiple instances of itself, they appear on the screen as separate windows. **Tabbed Document Interface** has come up to open multiple documents in the same window. This interface also helps in viewing preference panel in application. All modern web-browsers use this feature.
- **Menu** - Menu is an array of standard commands, grouped together and placed at a visible place (usually top) inside the application window. The menu can be programmed to appear or hide on mouse clicks.
- **Icon** - An icon is small picture representing an associated application. When these icons are clicked or double clicked, the application window is opened. Icon displays application and programs installed on a system in the form of small pictures.
- **Cursor** - Interacting devices such as mouse, touch pad, digital pen are represented in GUI as cursors. On screen cursor follows the instructions from hardware in almost real-time. Cursors are also named pointers in GUI systems. They are used to select menus, windows and other application features.

Application specific GUI components

A GUI of an application contains one or more of the listed GUI elements:

- **Application Window** - Most application windows uses the constructs supplied by operating systems but many use their own customer created windows to contain the contents of application.

- **Dialogue Box** - It is a child window that contains message for the user and request for some action to be taken. For Example: Application generate a dialogue to get confirmation from user to delete a file.



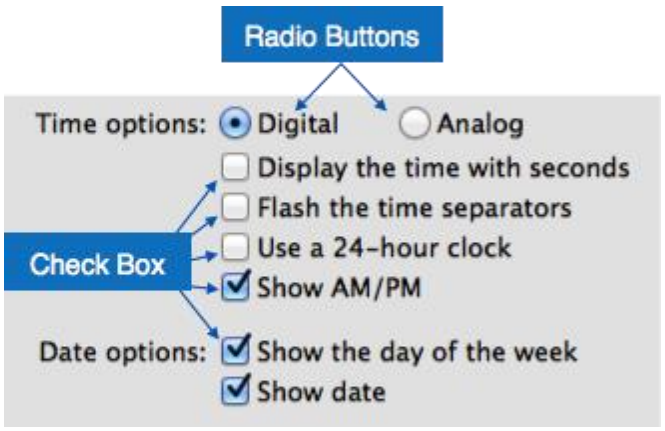
Untitled — Edited

Do you want to save the changes made to the document "Untitled"?

Your changes will be lost if you don't save them.

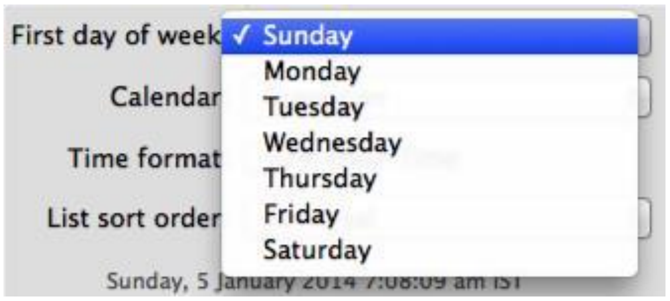
Save As: Untitled.txt
Tags:
Where: Desktop

- **Text-Box** - Provides an area for user to type and enter text-based data.
- **Buttons** - They imitate real life buttons and are used to submit inputs to the software.



Time options: [✓ Digital] Analog
Display the time with seconds
Flash the time separators
Use a 24-hour clock
[✓ Show AM/PM]
Date options: [✓ Show the day of the week]
[✓ Show date]

- **Radio-button** - Displays available options for selection. Only one can be selected among all offered.
- **Check-box** - Functions similar to list-box. When an option is selected, the box is marked as checked. Multiple options represented by check boxes can be selected.
- **List-box** - Provides list of available items for selection. More than one item can be selected.



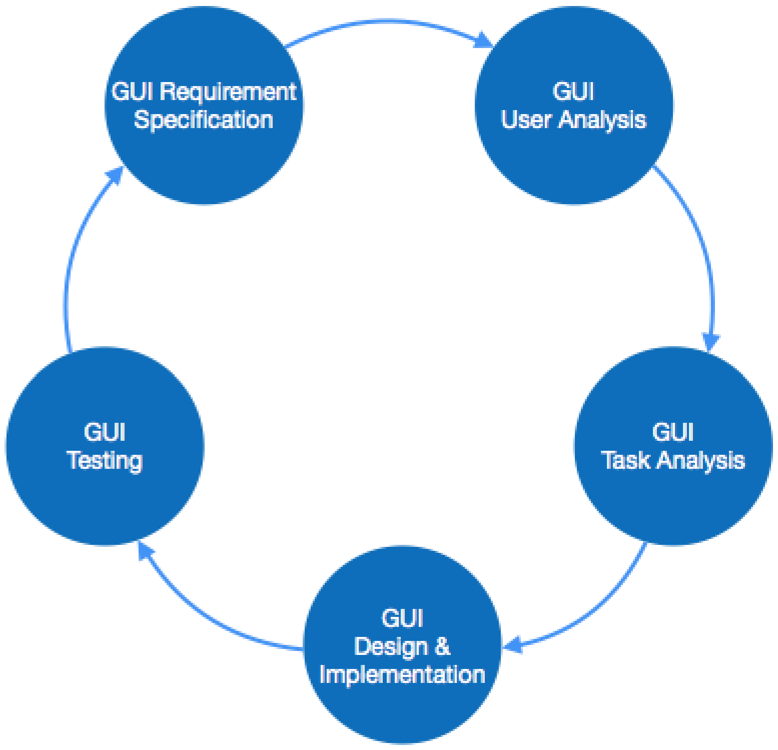
Other impressive GUI components are:

- Sliders
- Combo-box
- Data-grid
- Drop-down list

User Interface Design Activities

There are a number of activities performed for designing user interface. The process of GUI design and implementation is alike SDLC. Any model can be used for GUI implementation among Waterfall, Iterative or Spiral Model.

A model used for GUI design and development should fulfill these GUI specific steps.



- **GUI Requirement Gathering** - The designers may like to have list of all functional and non-functional requirements of GUI. This can be taken from user and their existing software solution.
- **User Analysis** - The designer studies who is going to use the software GUI. The target audience matters as the design details change according to the knowledge and competency level of the user. If user is technical savvy, advanced and complex GUI can be incorporated. For a novice user, more information is included on how-to of software.
- **Task Analysis** - Designers have to analyze what task is to be done by the software solution. Here in GUI, it does not matter how it will be done. Tasks can be represented in hierarchical manner taking one major task and dividing it further into smaller sub-tasks. Tasks provide goals for GUI presentation. Flow of information among sub-tasks determines the flow of GUI contents in the software.

- **GUI Design & implementation** - Designers after having information about requirements, tasks and user environment, design the GUI and implements into code and embed the GUI with working or dummy software in the background. It is then self-tested by the developers.
- **Testing** - GUI testing can be done in various ways. Organization can have in-house inspection, direct involvement of users and release of beta version are few of them. Testing may include usability, compatibility, user acceptance etc.

GUI Implementation Tools

There are several tools available using which the designers can create entire GUI on a mouse click. Some tools can be embedded into the software environment (IDE).

GUI implementation tools provide powerful array of GUI controls. For software customization, designers can change the code accordingly.

There are different segments of GUI tools according to their different use and platform. Example

Mobile GUI, Computer GUI, Touch-Screen GUI etc. Here is a list of few tools which come handy to build GUI:

- FLUID
- AppInventor (Android)
- LucidChart
- Wavemaker
- Visual Studio

User Interface Golden rules

The following rules are mentioned to be the golden rules for GUI design, described by Shneiderman and Plaisant in their book (Designing the User Interface).

- **Strive for consistency** - Consistent sequences of actions should be required in similar situations. Identical terminology should be used in prompts, menus, and help screens. Consistent commands should be employed throughout.
- **Enable frequent users to use short-cuts** - The user's desire to reduce the number of interactions increases with the frequency of use. Abbreviations, function keys, hidden commands, and macro facilities are very helpful to an expert user.
- **Offer informative feedback** - For every operator action, there should be some system feedback. For frequent and minor actions, the response must be modest, while for infrequent and major actions, the response must be more substantial.
- **Design dialog to yield closure** - Sequences of actions should be organized into groups with a beginning, middle, and end. The informative feedback at the completion of a group of actions gives the operators the satisfaction of accomplishment, a sense of relief, the signal to drop contingency plans and options from their minds, and this indicates that the way ahead is clear to prepare for the next group of actions.

- **Offer simple error handling** - As much as possible, design the system so the user will not make a serious error. If an error is made, the system should be able to detect it and offer simple, comprehensible mechanisms for handling the error.
- **Permit easy reversal of actions** - This feature relieves anxiety, since the user knows that errors can be undone. Easy reversal of actions encourages exploration of unfamiliar options. The units of reversibility may be a single action, a data entry, or a complete group of actions.
- **Support internal locus of control** - Experienced operators strongly desire the sense that they are in charge of the system and that the system responds to their actions. Design the system to make users the initiators of actions rather than the responders.
- **Reduce short-term memory load** - The limitation of human information processing in short-term memory requires the displays to be kept simple, multiple page displays be consolidated, window-motion frequency be reduced, and sufficient training time be allotted for codes, mnemonics, and sequences of actions.

Report Design

Definition of Report: A business report is a document that conveys specific information about your business to other individuals those can be investors, employees, managers or other superior. **Report Writing Software:**It consist of programs that produce both periodic reports and special reports.

- **Periodic Reports:**They are routine reports on a scheduled time(periodically). These reports are issued weekly, quarterly, or annually. For example, feedback.
- **Specific Reports:** They are type of unscheduled report that may be requested by managers.

Steps of writing a Report:

1. Define the problem.
2. Gather the necessary information.
3. Analyse the information.
4. Organize the information.
5. Write report.

Principles of writing a Report:

1. **One should highlight the important information-** managers should not have to waste their time searching the information they need.
2. **Report should be as simple as possible-** should be in simple format, consistency of language should be maintained, free from jargon etc.
3. Backup detail should be available.

Presentation Modes:

1. **Graphic Form:** Used when the information is not quantitative or an overview is needed, pictures can be used.
2. **Tabular Form:** Frequently used, which tabulates the production figures of one department.
3. **Narrative Form:** Used when the information is subjective and quantitative.

Management by Exception:

1. **Prepare the report only when exceptions occur:** These report printed only when employees work overtime, each entry on the report is an exception.
2. **By showing variance from normal:** Here the reports contain the details of the comparisons of the actual activity with the planned activity.

3. **Grouping the exception together:** The exceptions of special interest for the manager can be grouped and highlighted for quick attention.
4. **Highlighting exception by maintaining a certain sequence:** It is possible to sort report records into either in an ascending or descending sequence based on one or more key fields.

Importance of writing a Report:

1. Providing a deeper insight.
2. Highlights the specific problem of a business.
3. Help adopt right marketing strategy.
4. Helps in decision making and problem solving.

Structure chart

A **structure chart** (SC) in software engineering and organizational theory is a chart which shows the breakdown of a system to its lowest manageable levels.[2] They are used in structured programming to arrange program modules into a tree. Each module is represented by a box, which contains the module's name. The tree structure visualizes the relationships between modules.

A structure chart is a top-down modular design tool, constructed of squares representing the different modules in the system, and lines that connect them. The lines represent the connection and or ownership between activities and subactivities as they are used in organization charts

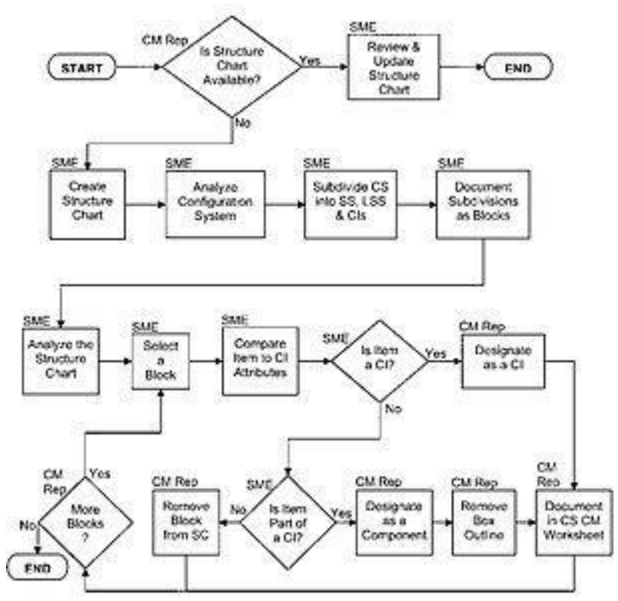
In structured analysis structure charts, according to Wolber (2009), "are used to specify the high- level design, or architecture, of a computer program. As a design tool, they aid the programmer in dividing and conquering a large software problem, that is, recursively breaking a problem down into parts that are small enough to be understood by a human brain. The process is
called top-down design, or functional decomposition. Programmers use a structure chart to build a program in a manner similar to how an architect uses a blueprint to build a house. In the design stage, the chart is drawn and used as a way for the client and the various software designers to communicate. During the actual building of the program (implementation), the chart is continually referred to as "the master-plan".[5]

A structure chart depicts[2]

- the size and complexity of the system, and
- number of readily identifiable functions and modules within each function and
- whether each identifiable function is a manageable entity or should be broken down into smaller components.

A structure chart is also used to diagram associated elements that comprise a run stream or thread. It is often developed as a hierarchical diagram, but other representations are allowable. The representation must describe the breakdown of the configuration system into subsystems and the lowest manageable level. An accurate and complete structure chart is the key to the determination of the configuration items (CI), and a visual representation of the configuration system and the internal interfaces among its CIs(define CI clearly). During the configuration control process, the structure chart is used to identify CIs and their associated artifacts that a proposed change may impact.[2]

Structure chart construction



A process flow diagram describing the construction of a structure chart by a so-called Subject Matter Experts (SME)

According to Wolber (2009), "a structure chart can be developed starting with the creating of a structure, which places the root of an upside-down tree which forms the structure chart. The next step is to conceptualize the main sub-tasks that must be performed by the program to solve the problem. Next, the programmer focuses on each sub-task individually, and conceptualizes how each can be broken down into even smaller tasks. Eventually, the program is broken down to a point where the leaves of the tree represent simple methods that can be coded with just a few program statements".[5]

In practice, see figure, first it is checked if a structure chart has been developed already. If so an expert needs to review it to ensure it represents the current structure and if not, updates the chart where needed.

Equipment Specification and Selection

The software requirements are description of features and functionalities of the target system. Requirements convey the expectations of users from the software product. The requirements can be obvious or hidden, known or unknown, expected or unexpected from client's point of view.

Requirement Engineering

The process to gather the software requirements from client, analyze and document them is known as requirement engineering.

The goal of requirement engineering is to develop and maintain sophisticated and descriptive 'System Requirements Specification' document.

Requirement Engineering Process

It is a four step process, which includes:

- Feasibility Study
- Requirement Gathering
- Software Requirement Specification
- Software Requirement Validation

Let us see the process briefly –

Feasibility study

When the client approaches the organization for getting the desired product developed, it comes
up with rough idea about what all functions the software must perform and which all features are expected from the software.

Referencing to this information, the analysts does a detailed study about whether the desired system and its functionality are feasible to develop.

This feasibility study is focused towards goal of the organization. This study analyzes whether the software product can be practically materialized in terms of implementation, contribution of project to organization, cost constraints and as per values and objectives of the organization. It explores technical aspects of the project and product such as usability, maintainability, productivity and integration ability.

The output of this phase should be a feasibility study report that should contain adequate comments and recommendations for management about whether or not the project should be undertaken.

Requirement Gathering

If the feasibility report is positive towards undertaking the project, next phase starts with gathering requirements from the user. Analysts and engineers communicate with the client and end-users to know their ideas on what the software should provide and which features they want the software to include.

Software Requirement Specification

SRS is a document created by system analyst after the requirements are collected from various stakeholders.

SRS defines how the intended software will interact with hardware, external interfaces, speed of operation, response time of system, portability of software across various platforms, maintainability, speed of recovery after crashing, Security, Quality, Limitations etc.

The requirements received from client are written in natural language. It is the responsibility of system analyst to document the requirements in technical language so that they can be comprehended and useful by the software development team.

SRS should come up with following features:

- User Requirements are expressed in natural language.
- Technical requirements are expressed in structured language, which is used inside the organization.
- Design description should be written in Pseudo code.
- Format of Forms and GUI screen prints.
- Conditional and mathematical notations for DFDs etc.

Software Requirement Validation

After requirement specifications are developed, the requirements mentioned in this document are validated. User might ask for illegal, impractical solution or experts may interpret the requirements incorrectly. This results in huge increase in cost if not nipped in the bud. Requirements can be checked against following conditions –

- If they can be practically implemented
- If they are valid and as per functionality and domain of software
- If there are any ambiguities
- If they are complete
- If they can be demonstrated

Requirement Elicitation Process

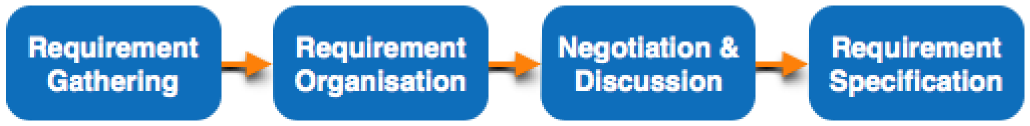
Requirement elicitation process can be depicted using the folloiwng diagram:

- **Requirements gathering** – The developers discuss with the client and end users and know their expectations from the software.
- **Organizing Requirements** – The developers prioritize and arrange the requirements in order of importance, urgency and convenience.

- **Negotiation & discussion** – If requirements are ambiguous or there are some conflicts in requirements of various stakeholders, if they are, it is then negotiated and discussed with stakeholders. Requirements may then be prioritized and reasonably compromised.

  The requirements come from various stakeholders. To remove the ambiguity and conflicts, they are discussed for clarity and correctness. Unrealistic requirements are compromised reasonably.

- **Documentation** – All formal & informal, functional and non-functional requirements are documented and made available for next phase processing.



Requirement Elicitation Techniques

Requirements Elicitation is the process to find out the requirements for an intended software system by communicating with client, end users, system users and others who have a stake in the software system development.

There are various ways to discover requirements

Interviews

Interviews are strong medium to collect requirements. Organization may conduct several types of interviews such as:

- Structured (closed) interviews, where every single information to gather is decided in advance, they follow pattern and matter of discussion firmly.
- Non-structured (open) interviews, where information to gather is not decided in advance, more flexible and less biased.
- Oral interviews
- Written interviews
- One-to-one interviews which are held between two persons across the table.
- Group interviews which are held between groups of participants. They help to uncover any missing requirement as numerous people are involved.

Surveys

Organization may conduct surveys among various stakeholders by querying about their expectation and requirements from the upcoming system.

Questionnaires

A document with pre-defined set of objective questions and respective options is handed over to all stakeholders to answer, which are collected and compiled.

A shortcoming of this technique is, if an option for some issue is not mentioned in the questionnaire, the issue might be left unattended.

Task analysis

Team of engineers and developers may analyze the operation for which the new system is required. If the client already has some software to perform certain operation, it is studied and requirements of proposed system are collected.

Domain Analysis

Every software falls into some domain category. The expert people in the domain can be a great help to analyze general and specific requirements.

Brainstorming

An informal debate is held among various stakeholders and all their inputs are recorded for further requirements analysis.

Prototyping

Prototyping is building user interface without adding detail functionality for user to interpret the features of intended software product. It helps giving better idea of requirements. If there is no software installed at client's end for developer's reference and the client is not aware of its own requirements, the developer creates a prototype based on initially mentioned requirements. The prototype is shown to the client and the feedback is noted. The client feedback serves as an input for requirement gathering.

Observation

Team of experts visit the client's organization or workplace. They observe the actual working of the existing installed systems. They observe the workflow at client's end and how execution problems are dealt. The team itself draws some conclusions which aid to form requirements expected from the software.

Software Requirements Characteristics

Gathering software requirements is the foundation of the entire software development project. Hence they must be clear, correct and well-defined.

A complete Software Requirement Specifications must be:

- Clear
- Correct
- Consistent
- Coherent
- Comprehensible
- Modifiable
- Verifiable
- Prioritized
- Unambiguous
- Traceable
- Credible source

Software Requirements

We should try to understand what sort of requirements may arise in the requirement elicitation phase and what kinds of requirements are expected from the software system.

Broadly software requirements should be categorized in two categories: Functional

Requirements

Requirements, which are related to functional aspect of software fall into this category.

They define functions and functionality within and from the software system.

EXAMPLES –

- Search option given to user to search from various invoices.
- User should be able to mail any report to management.
- Users can be divided into groups and groups can be given separate rights.
- Should comply business rules and administrative functions.
- Software is developed keeping downward compatibility intact.

Non-Functional Requirements

Requirements, which are not related to functional aspect of software, fall into this category. They are implicit or expected characteristics of software, which users make assumption of.

Non-functional requirements include –

- Security
- Logging
- Storage
- Configuration
- Performance
- Cost
- Interoperability
- Flexibility
- Disaster recovery
- Accessibility

Requirements are categorized logically as

- **Must Have** : Software cannot be said operational without them.
- **Should have** : Enhancing the functionality of software.
- **Could have** : Software can still properly function with these requirements.
- **Wish list** : These requirements do not map to any objectives of software.

While developing software, 'Must have' must be implemented, 'Should have' is a matter of debate with stakeholders and negation, whereas 'could have' and 'wish list' can be kept for software updates.

User Interface requirements

UI is an important part of any software or hardware or hybrid system. A software is widely accepted if it is –

- easy to operate
- quick in response
- effectively handling operational errors
- providing simple yet consistent user interface

User acceptance majorly depends upon how user can use the software. UI is the only way for users to perceive the system. A well performing software system must also be equipped with attractive, clear, consistent and responsive user interface. Otherwise the functionalities of software system can not be used in convenient way. A system is said be good if it provides means to use it efficiently. User interface requirements are briefly mentioned below –

- Content presentation
- Easy Navigation
- Simple interface
- Responsive
- Consistent UI elements
- Feedback mechanism
- Default settings
- Purposeful layout
- Strategical use of color and texture.
- Provide help information
- User centric approach
- Group based view settings.

Software System Analyst

System analyst in an IT organization is a person, who analyzes the requirement of proposed system and ensures that requirements are conceived and documented properly & correctly. Role of an analyst starts during Software Analysis Phase of SDLC. It is the responsibility of analyst to make sure that the developed software meets the requirements of the client.

System Analysts have the following responsibilities:

- Analyzing and understanding requirements of intended software
- Understanding how the project will contribute in the organization objectives
- Identify sources of requirement
- Validation of requirement
- Develop and implement requirement management plan
- Documentation of business, technical, process and product requirements
- Coordination with clients to prioritize requirements and remove and ambiguity
- Finalizing acceptance criteria with client and other stakeholders

Software Metrics and Measures

Software Measures can be understood as a process of quantifying and symbolizing various attributes and aspects of software.

Software Metrics provide measures for various aspects of software process and software product.

Software measures are fundamental requirement of software engineering. They not only help to control the software development process but also aid to keep quality of ultimate product excellent.

According to Tom DeMarco, a (Software Engineer), “You cannot control what you cannot measure.” By his saying, it is very clear how important software measures are.

Let us see some software metrics:

- **Size Metrics** – LOC (Lines of Code), mostly calculated in thousands of delivered source code lines, denoted as KLOC.

  Function Point Count is measure of the functionality provided by the software. Function Point count defines the size of functional aspect of software.

- **Complexity Metrics** – McCabe's Cyclomatic complexity quantifies the upper bound of the number of independent paths in a program, which is perceived as complexity of the program or its modules. It is represented in terms of graph theory concepts by using control flow graph.

- **Quality Metrics** – Defects, their types and causes, consequence, intensity of severity and their implications define the quality of product.

  The number of defects found in development process and number of defects reported by the client after the product is installed or delivered at client-end, define quality of product.

- **Process Metrics** – In various phases of SDLC, the methods and tools used, the company standards and the performance of development are software process metrics.
- **Resource Metrics** – Effort, time and various resources used, represents metrics for resource measurement.

Personnel Estimates

Staffing level estimation

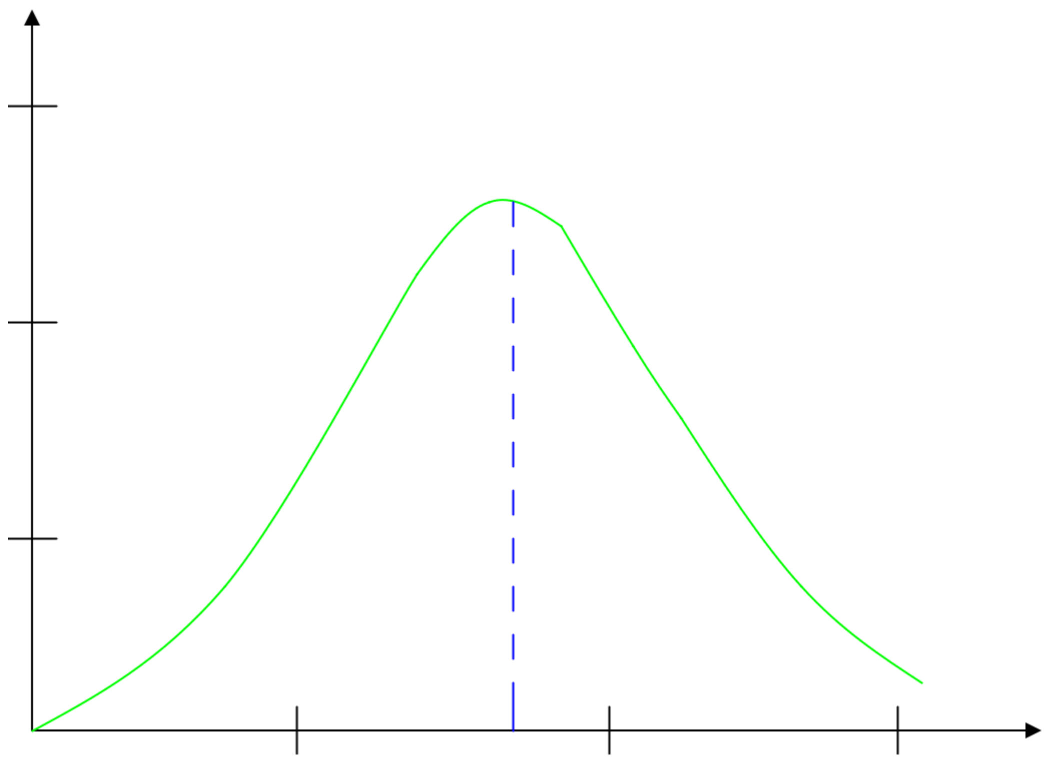
Once the effort required to develop a software has been determined, it is necessary to determine the staffing requirement for the project. Putnam first studied the problem of what should be a proper staffing pattern for software projects. He extended the work of Norden who had earlier investigated the staffing pattern of research and development (R&D) type of projects. In order to appreciate the staffing pattern of software projects, Norden's and Putnam's results must be understood.

Norden's Work

Norden studied the staffing patterns of several R & D projects. He found that the staffing pattern can be approximated by the Rayleigh distribution curve (as shown in fig. 11.6). Norden represented the Rayleigh curve by the following equation:

$$E = K/t^2_d * t * e^{-t^2 / 2 t^2_d}$$

Where E is the effort required at time t. E is an indication of the number of engineers (or the staffing level) at any particular time during the duration of the project, K is the area under the curve, and t_d is the time at which the curve attains its maximum value. It must be remembered that the results of Norden are applicable to general R & D projects and were not meant to model the staffing pattern of software development projects.



Effort per Unit Time

t_d

Time

Fig. 11.6: Rayleigh curve

Putnam's Work

Putnam studied the problem of staffing of software projects and found that the software development has characteristics very similar to other R & D projects studied by Norden and that the Rayleigh-Norden curve can be used to relate the number of delivered lines of code to the effort and the time required to develop the project. By analyzing a large number of army projects, Putnam derived the following expression:

$$\mathbf{L = C_k K^{1/3} t_d^{4/3}}$$

The various terms of this expression are as follows:

- K is the total effort expended (in PM) in the product development and L is the product size in KLOC.
- t_d corresponds to the time of system and integration testing. Therefore, t_d can be approximately considered as the time required to develop the software.

- C_k is the state of technology constant and reflects constraints that impede the progress of the programmer. Typical values of $C_k = 2$ for poor development environment (no methodology, poor documentation, and review, etc.), $C_k = 8$ for good software development environment (software engineering principles are adhered to), $C_k = 11$ for an excellent environment (in addition to following software engineering principles, automated tools and techniques are used). The exact value of C_k for a specific project can be computed from the historical data of the organization developing it.

Putnam suggested that optimal staff build-up on a project should follow the Rayleigh curve. Only a small number of engineers are needed at the beginning of a project to carry out planning and specification tasks. As the project progresses and more detailed work is required, the number of engineers reaches a peak. After implementation and unit testing, the number of project staff falls.

However, the staff build-up should not be carried out in large installments. The team size should either be increased or decreased slowly whenever required to match the Rayleigh-Norden curve. Experience shows that a very rapid build up of project staff any time during the project development correlates with schedule slippage.

It should be clear that a constant level of manpower through out the project duration would lead to wastage of effort and increase the time and effort required to develop the product. If a constant number of engineers are used over all the phases of a project, some phases would be overstaffed and the other phases would be understaffed causing inefficient use of manpower, leading to schedule slippage and increase in cost.

Effect of schedule change on cost

By analyzing a large number of army projects, Putnam derived the following expression:

$$L = CkK1/3td4/3$$

Where, K is the total effort expended (in PM) in the product development and L is the product size in KLOC, t_d corresponds to the time of system and integration testing and C_k is the state of technology constant and reflects constraints that impede the progress of the programmer

Now by using the above expression it is obtained that,

$$\mathbf{K = L^3/C_k^3 t_d^4}$$

Or,

$\mathbf{K = C/t_d^4}$

For the same product size, $\mathbf{C = L^3 / C_k^3}$ is a constant.

or, $\mathbf{K_1/K_2 = t_{d2}^4/t_{d1}^4}$

or, $\mathbf{K \propto 1/t_d^4}$

or, $\mathbf{cost \propto 1/t_d}$

(as project development effort is equally proportional to project development cost)

From the above expression, it can be easily observed that when the schedule of a project is compressed, the required development effort as well as project development cost increases in proportion to the fourth power of the degree of compression. It means that a relatively small compression in delivery schedule can result in substantial penalty of human effort as well as development cost. For example, if the estimated development time is 1 year, then in order to develop the product in 6 months, the total effort required to develop the product (and hence the project cost) increases 16 times.

I-O Design

Input Design

In an information system, input is the raw data that is processed to produce output. During the input design, the developers must consider the input devices such as PC, MICR, OMR, etc.

Therefore, the quality of system input determines the quality of system output. Welldesigned input forms and screens have following properties −

- It should serve specific purpose effectively such as storing, recording, and retrieving the information.
- It ensures proper completion with accuracy.
- It should be easy to fill and straightforward.
- It should focus on user’s attention, consistency, and simplicity.
- All these objectives are obtained using the knowledge of basic design principles regarding −
    - What are the inputs needed for the system?
    - How end users respond to different elements of forms and screens.

Objectives for Input Design

The objectives of input design are −

- To design data entry and input procedures
- To reduce input volume
- To design source documents for data capture or devise other data capture methods
- To design input data records, data entry screens, user interface screens, etc.
- To use validation checks and develop effective input controls.

Data Input Methods

It is important to design appropriate data input methods to prevent errors while entering data. These methods depend on whether the data is entered by customers in forms manually and later entered by data entry operators, or data is directly entered by users on the PCs.

A system should prevent user from making mistakes by −

- Clear form design by leaving enough space for writing legibly.
- Clear instructions to fill form.
- Clear form design.
- Reducing key strokes.
- Immediate error feedback.

Some of the popular data input methods are −

- Batch input method (Offline data input method)
- Online data input method
- Computer readable forms
- Interactive data input

Input Integrity Controls

Input integrity controls include a number of methods to eliminate common input errors by end- users. They also include checks on the value of individual fields; both for format and the completeness of all inputs.

Audit trails for data entry and other system operations are created using transaction logs which gives a record of all changes introduced in the database to provide security and means of recovery in case of any failure.

Output Design

The design of output is the most important task of any system. During output design, developers identify the type of outputs needed, and consider the necessary output controls and prototype report layouts.

Objectives of Output Design

The objectives of input design are −

- To develop output design that serves the intended purpose and eliminates the production of unwanted output.
- To develop the output design that meets the end users requirements.
- To deliver the appropriate quantity of output.
- To form the output in appropriate format and direct it to the right person.
- To make the output available on time for making good decisions.

Let us now go through various types of outputs −

External Outputs

Manufacturers create and design external outputs for printers. External outputs enable the system to leave the trigger actions on the part of their recipients or confirm actions to their recipients.

Some of the external outputs are designed as turnaround outputs, which are implemented as a form and re-enter the system as an input.

Internal outputs

Internal outputs are present inside the system, and used by end-users and managers. They support the management in decision making and reporting.

There are three types of reports produced by management information −

- **Detailed Reports** − They contain present information which has almost no filtering or restriction generated to assist management planning and control.
- **Summary Reports** − They contain trends and potential problems which are categorized and summarized that are generated for managers who do not want details.
- **Exception Reports** − They contain exceptions, filtered data to some condition or standard before presenting it to the manager, as information.

Output Integrity Controls

Output integrity controls include routing codes to identify the receiving system, and verification messages to confirm successful receipt of messages that are handled by network protocol.

Printed or screen-format reports should include a date/time for report printing and the data. Multipage reports contain report title or description, and pagination. Pre-printed forms usually include a version number and effective date.

UNIT-3

Data Dictionaries

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

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

- Name of the data item
- Aliases
- Description/purpose
- Related data items
- Range of values
- Data structure definition/Forms

The **name of the data item** is self-explanatory.

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

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

Related data items capture relationships between data items e.g., total_marks must always equal to internal_marks plus external_marks.

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

structure Forms: Data flows capture the name of processes that generate or receive the data items. If the data

item is primitive, then data structure form captures the physical structures of the data item.

If the data is itself a data aggregate, then data structure form capture the composition of the data items in term other data items.

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

Notations	Meaning
x=a+b	x includes of data elements a and b.
x=[a/b]	x includes of either data elements a or b.
x=a x	includes of optimal data elements a.
x=y[a]	x includes of y or more occurrences of data element a
x=[a]z	x includes of z or fewer occurrences of data element a
x=y[a]z	x includes of some occurrences of data element a which are between y and z.

Decision table

A decision table is a good way to deal with different combination inputs with their associated outputs.
It is a black box test design technique to determine the test scenarios for complex business logic.

What is Decision Table in Software Testing?

The decision table is a software testing technique which is used for testing the system behavior for different input combinations. This is a systematic approach where the different input combinations and their corresponding system behavior are captured in a tabular form.

This table helps you deal with different combination inputs with their associated outputs. Also, it is known as the cause-effect table because of an associated logical diagramming technique called cause-effect graphing that is basically used to derive the decision table.

Why is Decision Table Important?

A decision table is an outstanding technique used for testing and requirements management. Some of the reasons why the decision table is important include:

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

- It helps testers to search the effects of combinations of different inputs and other software states that implement business rules.
- It provides a regular way of stating complex business rules which benefits the developers as well as the testers.
- It assists in the development process with the developer to do a better job. Testing with all combination might be impractical.
- It the most preferable choice for testing and requirements management.
- It is a structured exercise to prepare requirements when dealing with complex business rules.
- It is also used in model complicated logic.

Advantages of Decision Table in Software Testing

There are different advantages of using the decision table in software testing such as:

- Any complex business flow can be easily converted into the test scenarios & test cases using this technique.
- Decision tables work iteratively. Therefore, the table created at the first iteration is used as the input table for the next tables. The iteration is done only if the initial table is not satisfactory.
- Simple to understand and everyone can use this method to design the test scenarios & test cases.
- It provides complete coverage of test cases which help to reduce the rework on writing test scenarios & test cases.
- These tables guarantee that we consider every possible combination of condition values. This is known as its completeness property.

Way to use Decision Table: Example

A **Decision Table** is a **tabular representation** of inputs versus rules, cases or test conditions. Let’s take an example and see how to create a decision table for a login screen:

The condition states that if the user provides the correct username and password the user will be redirected to the homepage. If any of the input is wrong, an error message will be displayed.

conditions	Rule 1	Rule 2	Rule 3	Rule 4
Username	F	T	F	T
Password	F	F	T	T
Output	E	E	E	H

In the above example,

- **T** – Correct username/password
- **F** – Wrong username/password
- **E** – Error message is displayed
- **H** – Home screen is displayed

Now let’s understand the interpretation of the above cases:

- **Case 1** – Username and password both were wrong. The user is shown an error message.
- **Case 2** – Username was correct, but the password was wrong. The user is shown an error message.
- **Case 3** – Username was wrong, but the password was correct. The user is shown an error message.
- **Case 4** – Username and password both were correct, and the user is navigated to the homepage.

Decision Trees

A **decision tree** is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements.

Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal, but are also a popular tool in machine learning.

A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.

In decision analysis, a decision tree and the closely related influence diagram are used as a visual and analytical decision support tool, where the expected values (or expected utility) of competing alternatives are calculated.

A decision tree consists of three types of nodes:[1]

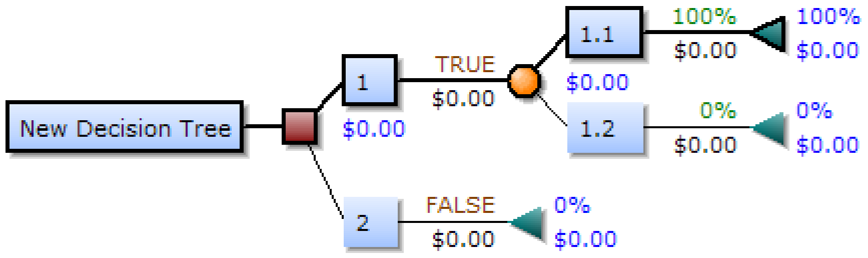
1. Decision nodes – typically represented by squares
2. Chance nodes – typically represented by circles
3. End nodes – typically represented by triangles

Decision trees are commonly used in operations research and operations management. If, in practice, decisions have to be taken online with no recall under incomplete knowledge, a decision tree should be paralleled by a probability model as a best choice model or online selection model algorithm. Another use of decision trees is as a descriptive means for calculating conditional probabilities.

Decision trees, influence diagrams, utility functions, and other decision analysis tools and methods are taught to undergraduate students in schools of business, health economics, and public health, and are examples of operations research or management science methods.

Decision tree building blocks

Decision tree elements



Drawn from left to right, a decision tree has only burst nodes (splitting paths) but no sink nodes (converging paths). Therefore, used manually, they can grow very big and are then often hard to draw fully by hand. Traditionally, decision trees have been created manually — as the aside example shows — although increasingly, specialized software is employed.

Decision rules

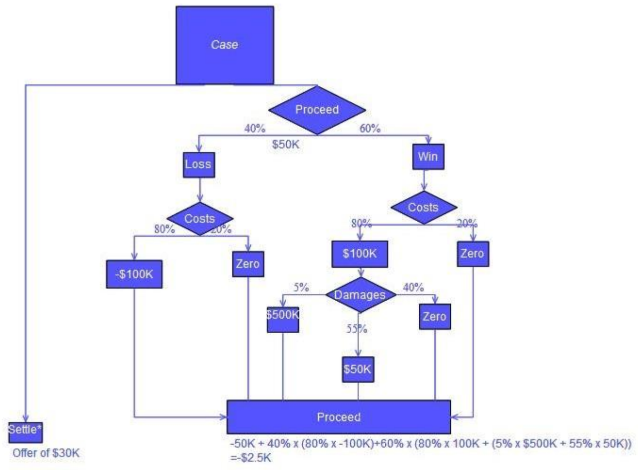
The decision tree can be linearized into **decision rules**,[2] where the outcome is the contents of the leaf node, and the conditions along the path form a conjunction in the if clause. In general, the rules have the form:

if condition1 *and* condition2 *and* condition3 *then* outcome.

Decision rules can be generated by constructing association rules with the target variable on the right. They can also denote temporal or causal relations.[3]

Decision tree using flowchart symbols

Commonly a decision tree is drawn using flowchart symbols as it is easier for many to read and understand.



Education and training System testing Change over

After acquiring a computer program (system) you are require to change from the old system to the newly acquired system in a process called system changeover. It involves replacing the old system and business processes with new once.

General activities carried out during

***changeover* Replace old hardware**

systems with new.

The new system may not be supported by the old hardware system; hence you will be required to replace them with new, i.e. the new system may require a high speed to process
information. This means that you will require a server that has a bigger RAM and processing speed. Again you may require more storage space.

Replacing business procedures

New system may mean that you change the procedures that you use to carry out a certain process. If for example in old system you use to scan customer image before using the system and then now the new system require to use a webcam to capture image the it means procedure has to change.

Training users

System and other relevant administrators should be trained on how system should be managed because they will be tasked with maintaining the system. Another group to be trained is the end user who will be using system for data capturing and entry. Training should be done to give the user confident required when serving customer.

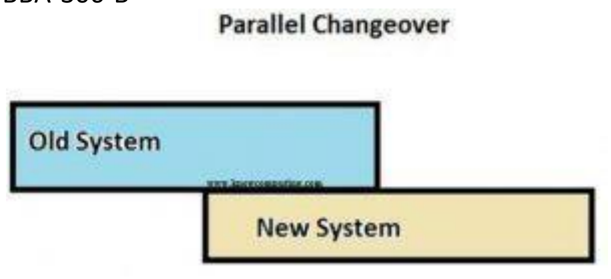
Conversion of data to suit the new system

If the old and new systems are using different data types then you will be required to convert the data from old format to new once. If the old system was a manual system then you digitalized data to be used in computerized system that you want to implement.

System Changeover strategies

Parallel changeover

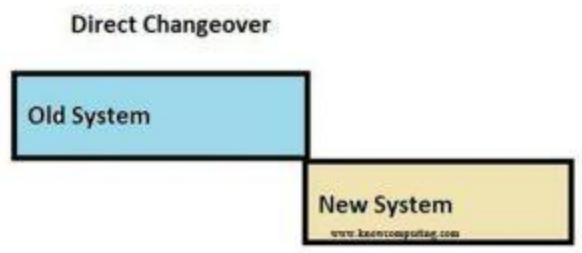
It involves running both the new and old system concurrently until you are confident that the new system is working effectively with low risk. The strategy assures a rollback to old system in-case something goes wrong with the new system. The strategy also allows user time to familiarize with new system and gain confidence to use it.



The main shortcoming of this strategy is that it over strains resources because they are required in both systems. It requires large storage capacity to accommodate both systems. User will be required to do double entry of data, both in the old and new systems. It is also difficult to compare the two systems and how they output data because they may be completely different. Another problem comes in determining for how long you should run the old system before replacing it completely.

Direct changeover

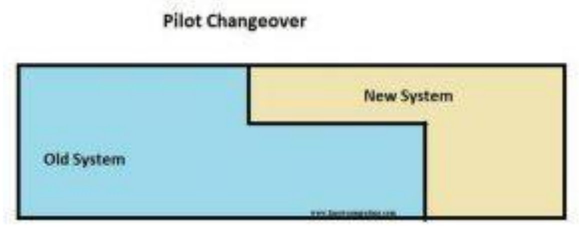
In direct changeover the old system is replace on time with a new system. It is mostly used when the risk of losing data from the old system is significantly low or if the system has most the functions that are new. It uses less resource because only one system is running.



With this system if something happens with the new system then it means no going back to the old system because it is no more. Users are not confident enough to use system on initial day of system implementation hence they operate relatively slow on the operation involved.

Pilot changeover

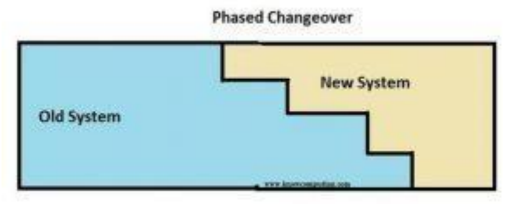
It means choosing a specific location or branch of the organization and implementing the system in that branch first. The branch (location) where the system is first test before it is implemented in the whole organization is called pilot site. It allow to test the system on a small scale on all it functionality and make any changes necessary to avoid any problems when later it is rolled on all organization branches.



The strategy is used as alternative to parallel change over because it cost less and it almost achieves the same results.

Phase changeover

Phased involves implementing a module of the system at a time until the whole system is implemented. It combines the parallel and direct change over strategies. The module can be a functional part of the system or a specific subsystem. Each sub system is implemented until it succeeds that when the next one is implemented. It means in case the new system fail then it is only that part that is affected and not the whole system.



The main challenge is that before the whole system is implemented it may take long time.

Different strategies are used at different time depending with the organization circumstances at that particular time.

Software maintenance is widely accepted part of SDLC now a days. It stands for all the modifications and updations done after the delivery of software product. There are number of reasons, why modifications are required, some of them are briefly mentioned below:

- **Market Conditions** - Policies, which changes over the time, such as taxation and newly introduced constraints like, how to maintain bookkeeping, may trigger need for modification.
- **Client Requirements** - Over the time, customer may ask for new features or functions in the software.
- **Host Modifications** - If any of the hardware and/or platform (such as operating system) of the target host changes, software changes are needed to keep adaptability.
- **Organization Changes** - If there is any business level change at client end, such as reduction of organization strength, acquiring another company, organization venturing into new business, need to modify in the original software may arise.

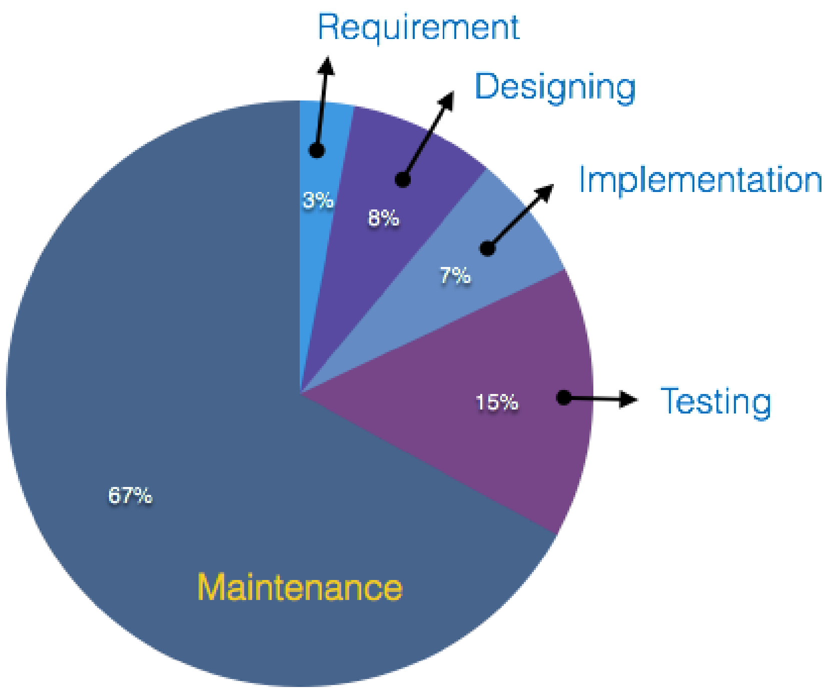
Types of maintenance

In a software lifetime, type of maintenance may vary based on its nature. It may be just a routine maintenance tasks as some bug discovered by some user or it may be a large event in itself based on maintenance size or nature. Following are some types of maintenance based on their characteristics:

- **Corrective Maintenance** - This includes modifications and updations done in order to correct or fix problems, which are either discovered by user or concluded by user error reports.
- **Adaptive Maintenance** - This includes modifications and updations applied to keep the software product up-to date and tuned to the ever changing world of technology and business environment.
- **Perfective Maintenance** - This includes modifications and updates done in order to keep the software usable over long period of time. It includes new features, new user requirements for refining the software and improve its reliability and performance.
- **Preventive Maintenance** - This includes modifications and updations to prevent future problems of the software. It aims to attend problems, which are not significant at this moment but may cause serious issues in future.

Cost of Maintenance

Reports suggest that the cost of maintenance is high. A study on estimating software maintenance found that the cost of maintenance is as high as 67% of the cost of entire software process cycle.



On an average, the cost of software maintenance is more than 50% of all SDLC phases. There are various factors, which trigger maintenance cost go high, such as:

Real-world factors affecting Maintenance Cost

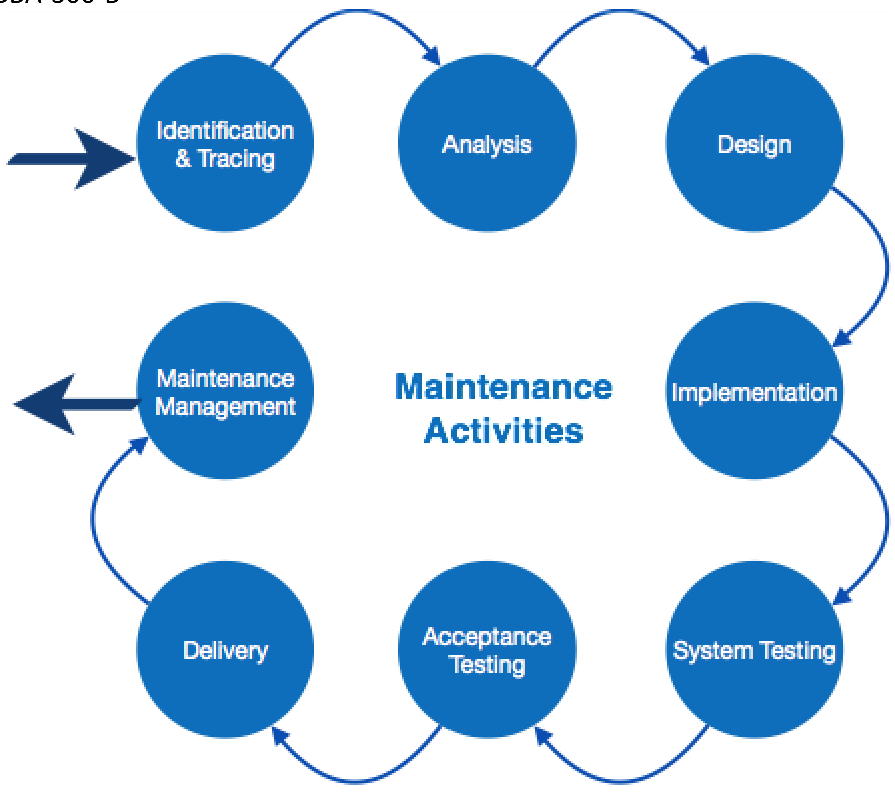
- The standard age of any software is considered up to 10 to 15 years.
- Older softwares, which were meant to work on slow machines with less memory and storage capacity cannot keep themselves challenging against newly coming enhanced softwares on modern hardware.
- As technology advances, it becomes costly to maintain old software.
- Most maintenance engineers are newbie and use trial and error method to rectify problem.
- Often, changes made can easily hurt the original structure of the software, making it hard for any subsequent changes.
- Changes are often left undocumented which may cause more conflicts in future.

Software-end factors affecting Maintenance Cost

- Structure of Software Program
- Programming Language
- Dependence on external environment
- Staff reliability and availability

Maintenance Activities

IEEE provides a framework for sequential maintenance process activities. It can be used in iterative manner and can be extended so that customized items and processes can be included.



These activities go hand-in-hand with each of the following phase:

- **Identification & Tracing** - It involves activities pertaining to identification of requirement of modification or maintenance. It is generated by user or system may itself report via logs or error messages.Here, the maintenance type is classified also.
- **Analysis** - The modification is analyzed for its impact on the system including safety and security implications. If probable impact is severe, alternative solution is looked for. A set of required modifications is then materialized into requirement specifications. The cost of modification/maintenance is analyzed and estimation is concluded.
- **Design** - New modules, which need to be replaced or modified, are designed against requirement specifications set in the previous stage. Test cases are created for validation and verification.
- **Implementation** - The new modules are coded with the help of structured design created in the design step.Every programmer is expected to do unit testing in parallel.
- **System Testing** - Integration testing is done among newly created modules. Integration testing is also carried out between new modules and the system. Finally the system is tested as a whole, following regressive testing procedures.
- **Acceptance Testing** - After testing the system internally, it is tested for acceptance with the help of users. If at this state, user complaints some issues they are addressed or noted to address in next iteration.

- **Delivery** - After acceptance test, the system is deployed all over the organization either by small update package or fresh installation of the system. The final testing takes place at client end after the software is delivered.

  Training facility is provided if required, in addition to the hard copy of user manual.

- **Maintenance management** - Configuration management is an essential part of system maintenance. It is aided with version control tools to control versions, semi-version or patch management.

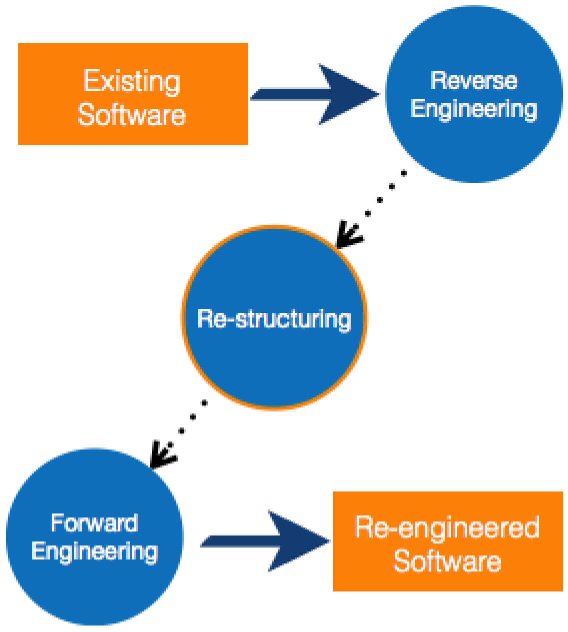
Software Re-engineering

When we need to update the software to keep it to the current market, without impacting its functionality, it is called software re-engineering. It is a thorough process where the design of software is changed and programs are re-written.

Legacy software cannot keep tuning with the latest technology available in the market. As the hardware become obsolete, updating of software becomes a headache. Even if software grows old with time, its functionality does not.

For example, initially Unix was developed in assembly language. When language C came into existence, Unix was re-engineered in C, because working in assembly language was difficult.

Other than this, sometimes programmers notice that few parts of software need more maintenance than others and they also need re-engineering.



Re-Engineering Process

- **Decide** what to re-engineer. Is it whole software or a part of it?
- **Perform** Reverse Engineering, in order to obtain specifications of existing software.

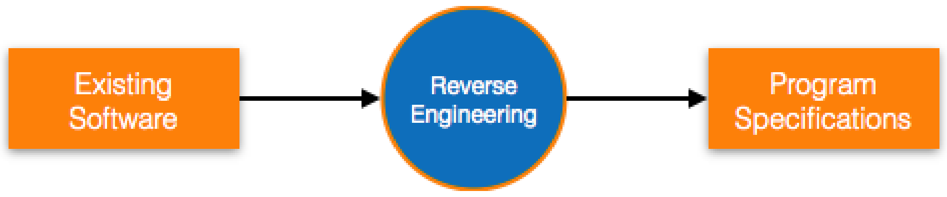
- **Restructure Program** if required. For example, changing function-oriented programs into object-oriented programs.
- **Re-structure data** as required.
- **Apply Forward engineering** concepts in order to get re-engineered software.

There are few important terms used in Software re-engineering

Reverse Engineering

It is a process to achieve system specification by thoroughly analyzing, understanding the existing system. This process can be seen as reverse SDLC model, i.e. we try to get higher abstraction level by analyzing lower abstraction levels.

An existing system is previously implemented design, about which we know nothing. Designers then do reverse engineering by looking at the code and try to get the design. With design in hand, they try to conclude the specifications. Thus, going in reverse from code to system specification.



Program Restructuring

It is a process to re-structure and re-construct the existing software. It is all about re-arranging the source code, either in same programming language or from one programming language to a different one. Restructuring can have either source code-restructuring and data-restructuring or both.

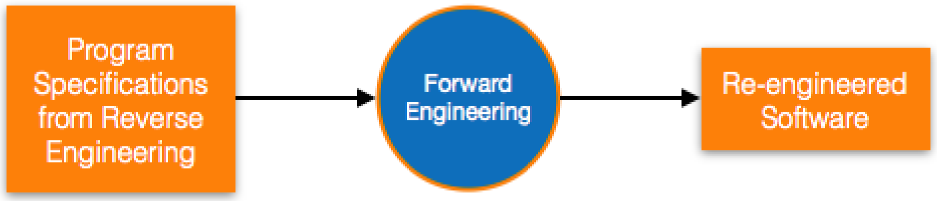
Re-structuring does not impact the functionality of the software but enhance reliability and maintainability. Program components, which cause errors very frequently can be changed, or updated with re-structuring.

The dependability of software on obsolete hardware platform can be removed via re-structuring.

Forward Engineering

Forward engineering is a process of obtaining desired software from the specifications in hand which were brought down by means of reverse engineering. It assumes that there was some software engineering already done in the past.

Forward engineering is same as software engineering process with only one difference – it is carried out always after reverse engineering.



Component reusability

A component is a part of software program code, which executes an independent task in the system. It can be a small module or sub-system itself.

Example

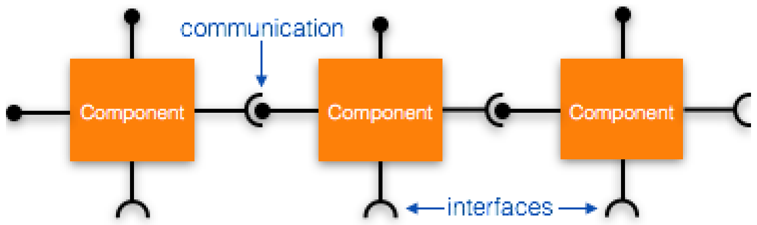
The login procedures used on the web can be considered as components, printing system in software can be seen as a component of the software.

Components have high cohesion of functionality and lower rate of coupling, i.e. they work independently and can perform tasks without depending on other modules.

In OOP, the objects are designed are very specific to their concern and have fewer chances to be used in some other software.

In modular programming, the modules are coded to perform specific tasks which can be used across number of other software programs.

There is a whole new vertical, which is based on re-use of software component, and is known as Component Based Software Engineering (CBSE).



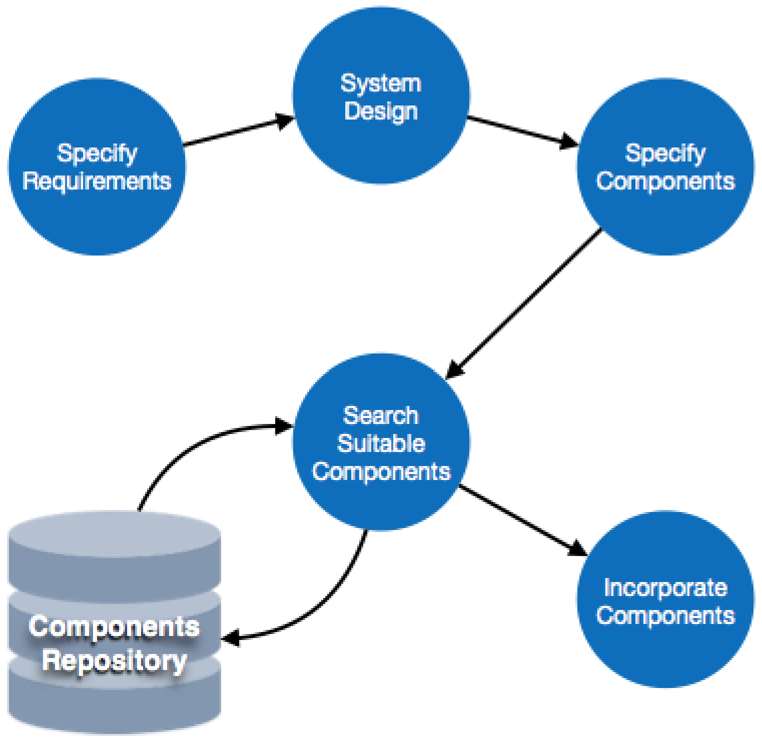
Re-use can be done at various levels

- **Application level** - Where an entire application is used as sub-system of new software.
- **Component level** - Where sub-system of an application is used.
- **Modules level** - Where functional modules are re-used.

Software components provide interfaces, which can be used to establish communication among different components.

Reuse Process

Two kinds of method can be adopted: either by keeping requirements same and adjusting components or by keeping components same and modifying requirements.



- **Requirement Specification** - The functional and non-functional requirements are specified, which a software product must comply to, with the help of existing system, user input or both.
- **Design** - This is also a standard SDLC process step, where requirements are defined in terms of software parlance. Basic architecture of system as a whole and its sub-systems are created.
- **Specify Components** - By studying the software design, the designers segregate the entire system into smaller components or sub-systems. One complete software design turns into a collection of a huge set of components working together.
- **Search Suitable Components** - The software component repository is referred by designers to search for the matching component, on the basis of functionality and intended software requirements..
- **Incorporate Components** - All matched components are packed together to shape them as complete software.

Unit -4

Introduction to Distributed Data Processing and Real Time System

Distributed systems use multiple central processors to serve multiple real-time applications and multiple users. Data processing jobs are distributed among the processors accordingly.

The processors communicate with one another through various communication lines (such as high-speed buses or telephone lines). These are referred as **loosely coupled systems** or distributed systems. Processors in a distributed system may vary in size and function. These processors are referred as sites, nodes, computers, and so on.

The advantages of distributed systems are as follows:

- With resource sharing facility, a user at one site may be able to use the resources available at another.
- Speedup the exchange of data with one another via electronic mail.
- If one site fails in a distributed system, the remaining sites can potentially continue operating.
- Better service to the customers.
- Reduction of the load on the host computer.
- Reduction of delays in data processing.

Real Time operating System

A real-time system is defined as a data processing system in which the time interval required to process and respond to inputs is so small that it controls the environment. The time taken by the system to respond to an input and display of required updated information is termed as the **response time**. So in this method, the response time is very less as compared to online processing.

Real-time systems are used when there are rigid time requirements on the operation of a processor or the flow of data and real-time systems can be used as a control device in a dedicated application. A real-time operating system must have well-defined, fixed time constraints, otherwise the system will fail. For example, Scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.

There are two types of real-time operating systems.

Hard real-time systems

Hard real-time systems guarantee that critical tasks complete on time. In hard real-time systems, secondary storage is limited or missing and the data is stored in ROM. In these systems, virtual memory is almost never found.

Soft real-time systems

Soft real-time systems are less restrictive. A critical real-time task gets priority over other tasks and retains the priority until it completes. Soft real-time systems have limited utility than hard real-time systems. For example, multimedia, virtual reality, Advanced Scientific Projects like undersea exploration and planetary rovers, etc.

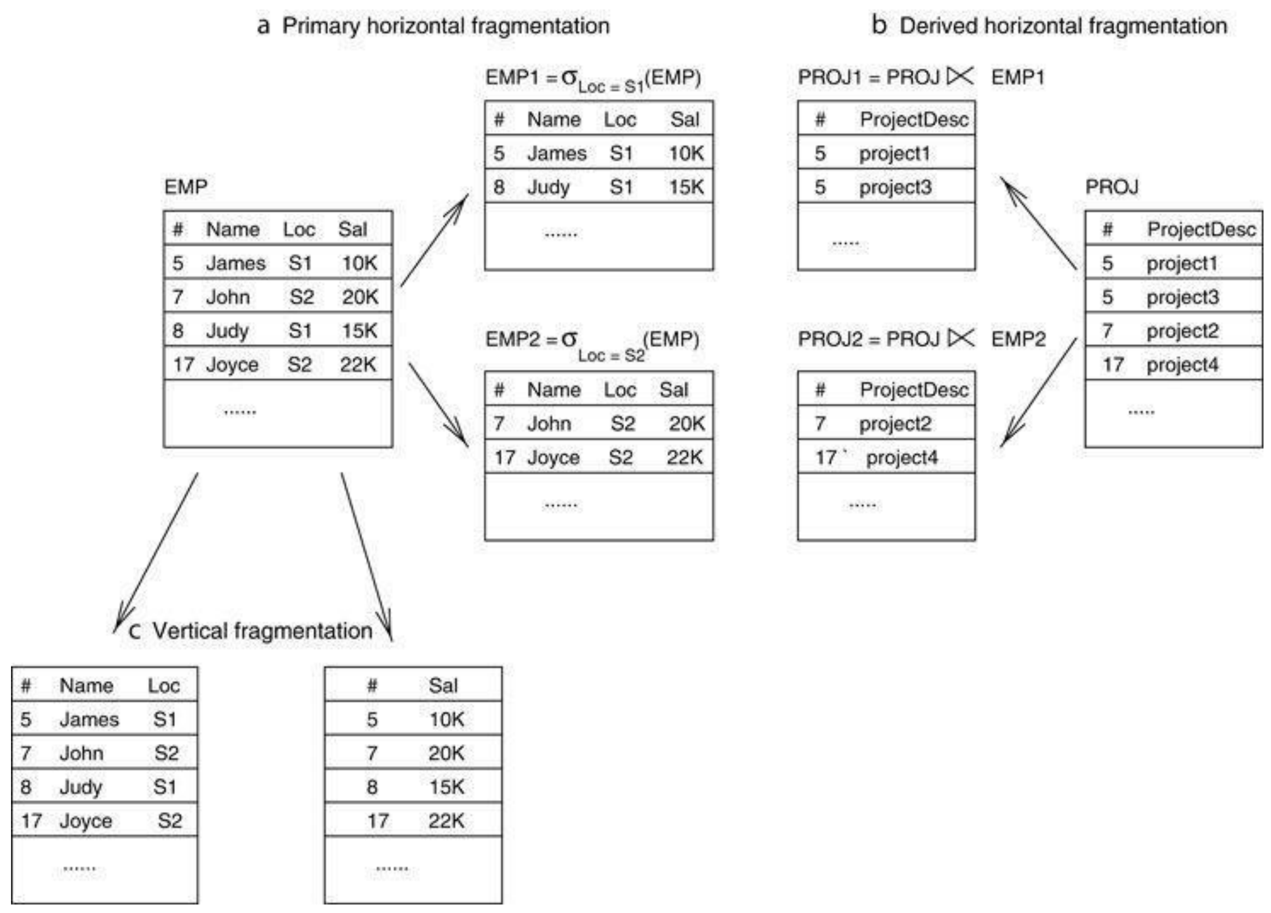
Evaluating distributed System

The performance evaluation of hardware and software system components is based on statistics that are long views on the behavior of these components. Since system resources may have unexpected behavior, relevant current information becomes useful in the management process of these systems, especially for data gathering, reconfiguration, and fault detection activities.
Actually, there are few criteria to property evaluate the current availability of component services within distributed systems. Hence, the management system can not realistically select the most suitable decision for configuration. In this paper, we present a proposal for a continuous evaluation of component behaviour related to state changes. This model is further extended by considering different categories of events concerning the degradation of the operational state or usage state. Our proposals are based on the possibility of computing at the component level, the current availability of this component by continuous evaluation. we introduce a several current availability features and propose formula to compute them. Other events concerning a managed object are classified as warning, critical or outstanding, which leads to a more accurate operational view on a component. Several counter-based events are thresholded to improve predictable reconfiguration decisions concerning the usability of a component. The main goal is to offer to the management system current relevant information which can be used within management policies the flexible polling frequency tuned with respect to the current evaluation, or particular aspects related to dynamic tests within distributed systems. Implementation issues with respect to the standard recommendations within distributed systems are presented. Finally we describe how the reconfiguration management systems can use these features in order to monitor, predict, improve the existing configuration, or accommodate the polling frequency according to several simple criteria.

Designing distributed database

Distributed database design refers to the following problem: given a database and its workload, how should the database be split and allocated to sites so as to optimize certain objective function (e.g., to minimize the resource consumption in processing the query workload). There are two issues: (i) Data fragmentation which determines how the data should be fragmented; and (ii) Data allocation which determines how the fragments should be allocated. While these two problems are inter-related, the two issues have traditionally been studied independently, giving rise to a two-phase approach to the design problem.

The design problem is applicable when a distributed database system has to be built from scratch. In the case when multiple existing databases are to be integrated (e.g., in multi-database context), there is no design issue.


a Primary horizontal fragmentation
b Derived horizontal fragmentation
c Vertical fragmentation

EMP

#	Name	Loc	Sal
5	James	S1	10K
7	John	S2	20K
8	Judy	S1	15K
17	Joyce	S2	22K
......			

EMP1 = $\sigma_{Loc = S1}$(EMP)

#	Name	Loc	Sal
5	James	S1	10K
8	Judy	S1	15K
......			

EMP2 = $\sigma_{Loc = S2}$(EMP)

#	Name	Loc	Sal
7	John	S2	20K
17	Joyce	S2	22K
......			

PROJ

#	ProjectDesc
5	project1
5	project3
7	project2
17	project4
.....	

PROJ1 = PROJ $\ltimes$ EMP1

#	ProjectDesc
5	project1
5	project3
.....	

PROJ2 = PROJ $\ltimes$ EMP2

#	ProjectDesc
7	project2
17`	project4
.....	

#	Name	Loc
5	James	S1
7	John	S2
8	Judy	S1
17	Joyce	S2
......		

#	Sal
5	10K
7	20K
8	15K
17	22K
......	

Whether the database is centralized or distributed, the design principles and concepts are same. However, the design of a distributed database introduces three new issues:

- How to partition the database into fragments.
- Which fragments to replicate.
- Where to locate those fragments and replicas.

Data fragmentation and data replication deal with the first two issues and data allocation deals with the third issue.

Data Fragmentation:

Data fragmentation allows you to break a single object into two or more segments, or fragments. The object might be a user's database, a system database, or a table. Each fragment can be stored at any site over a computer network. Information about data fragmentation is stored in the distributed data catalog (DDC), from which it is accessed by the TP to process user requests.

Data fragmentation strategies, as discussed here, are based at the table level and consist of dividing a table into logical fragments. You will explore three types of data fragmentation strategies: horizontal, vertical, and mixed.

Horizontal fragmentation refers to the division of a relation into subsets (fragments) of tuples (rows). Each fragment is stored at a different node, and each fragment has unique rows.
However, the unique rows all have the same attributes (columns). In short, each fragment represents the equivalent of a SELECT statement, with the WHERE clause on a single attribute.

Vertical fragmentation refers to the division of a relation into attribute (column) subsets. Each subset (fragment) is stored at a different node, and each fragment has unique columns—with the exception of the key column, which is common to all fragments. This is the equivalent of the PROJECT statement in SQL.

Mixed fragmentation refers to a combination of horizontal and vertical strategies. In other words, a table may be divided into several horizontal subsets (rows), each one having a subset of the attributes (columns).

To illustrate the fragmentation strategies, let's use the CUSTOMER table for the XYZ Company. The table contains the attributes CUS_NUM, CUS_NAME, CUS_ADDRESS, CUS_STATE, CUS_LIMIT, CUS_BAL, CUS_RATING, and CUS_DUE.

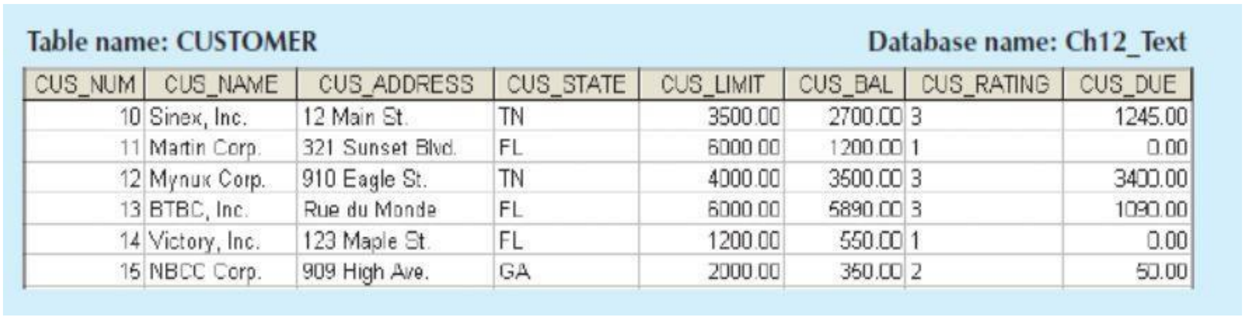
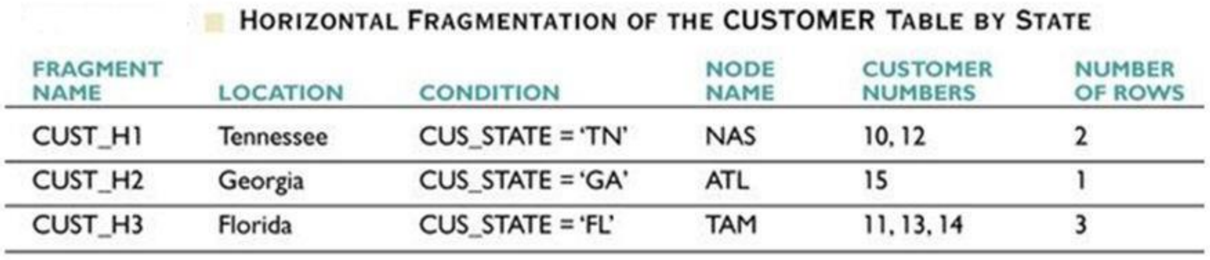


Table name: CUSTOMER **Database name: Ch12_Text**

CUS_NUM	CUS_NAME	CUS_ADDRESS	CUS_STATE	CUS_LIMIT	CUS_BAL	CUS_RATING	CUS_DUE
10	Sinex, Inc.	12 Main St.	TN	3500.00	2700.00	3	1245.00
11	Martin Corp.	321 Sunset Blvd.	FL	6000.00	1200.00	1	0.00
12	Mynux Corp.	910 Eagle St.	TN	4000.00	3500.00	3	3400.00
13	BTBC, Inc.	Rue du Monde	FL	6000.00	5890.00	3	1090.00
14	Victory, Inc.	123 Maple St.	FL	1200.00	550.00	1	0.00
15	NBCC Corp.	909 High Ave.	GA	2000.00	350.00	2	50.00

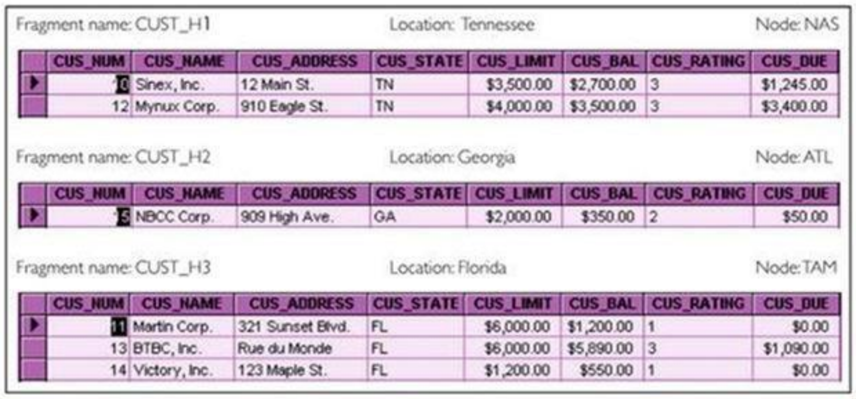
Horizontal Fragmentation:

Suppose that XYZ Company's corporate management requires information about its customers in all three states, but company locations in each state (TN, FL, and GA) require data regarding local customers only. Based on such requirements, you decide to distribute the data by state.
Therefore, you define the horizontal fragments to conform to the structure shown in the following table.



HORIZONTAL FRAGMENTATION OF THE CUSTOMER TABLE BY STATE

FRAGMENT NAME	LOCATION	CONDITION	NODE NAME	CUSTOMER NUMBERS	NUMBER OF ROWS
CUST_H1	Tennessee	CUS_STATE = 'TN'	NAS	10, 12	2
CUST_H2	Georgia	CUS_STATE = 'GA'	ATL	15	1
CUST_H3	Florida	CUS_STATE = 'FL'	TAM	11, 13, 14	3



Fragment name: CUST_H1 Location: Tennessee Node: NAS

CUS_NUM	CUS_NAME	CUS_ADDRESS	CUS_STATE	CUS_LIMIT	CUS_BAL	CUS_RATING	CUS_DUE
10	Sinex, Inc.	12 Main St.	TN	$3,500.00	$2,700.00	3	$1,245.00
12	Mynux Corp.	910 Eagle St.	TN	$4,000.00	$3,500.00	3	$3,400.00

Fragment name: CUST_H2 Location: Georgia Node: ATL

CUS_NUM	CUS_NAME	CUS_ADDRESS	CUS_STATE	CUS_LIMIT	CUS_BAL	CUS_RATING	CUS_DUE
15	NBCC Corp.	909 High Ave.	GA	$2,000.00	$350.00	2	$50.00

Fragment name: CUST_H3 Location: Florida Node: TAM

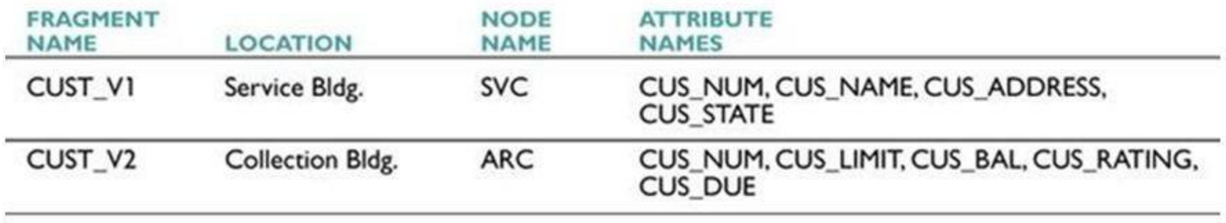
CUS_NUM	CUS_NAME	CUS_ADDRESS	CUS_STATE	CUS_LIMIT	CUS_BAL	CUS_RATING	CUS_DUE
11	Martin Corp.	321 Sunset Blvd.	FL	$6,000.00	$1,200.00	1	$0.00
13	BTBC, Inc.	Rue du Monde	FL	$6,000.00	$5,890.00	3	$1,090.00
14	Victory, Inc.	123 Maple St.	FL	$1,200.00	$550.00	1	$0.00

TABLE FRAGMENTS IN THREE LOCATIONS

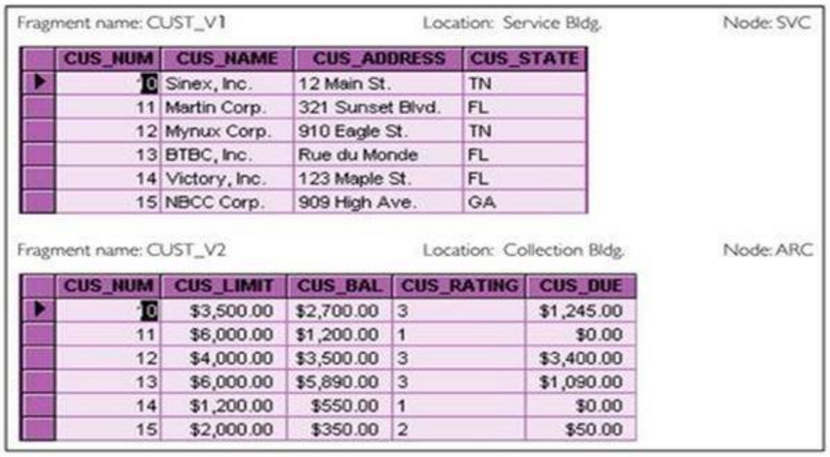
Vertical Fragmentation

You may also divide the CUSTOMER relation into vertical fragments that are composed of a collection of attributes. For example, suppose that the company is divided into two departments: the service department and the collections department. Each department is located in a separate building, and each has an interest in only a few of the CUSTOMER table's attributes. In this case, the fragments are defined as shown in the following table.

VERTICAL FRAGMENTATION OF THE CUSTOMER TABLE



FRAGMENT NAME	LOCATION	NODE NAME	ATTRIBUTE NAMES
CUST_V1	Service Bldg.	SVC	CUS_NUM, CUS_NAME, CUS_ADDRESS, CUS_STATE
CUST_V2	Collection Bldg.	ARC	CUS_NUM, CUS_LIMIT, CUS_BAL, CUS_RATING, CUS_DUE



Fragment name: CUST_V1 Location: Service Bldg. Node: SVC

CUS_NUM	CUS_NAME	CUS_ADDRESS	CUS_STATE
10	Sinex, Inc.	12 Main St.	TN
11	Martin Corp.	321 Sunset Blvd.	FL
12	Mynux Corp.	910 Eagle St.	TN
13	BTBC, Inc.	Rue du Monde	FL
14	Victory, Inc.	123 Maple St.	FL
15	NBCC Corp.	909 High Ave.	GA

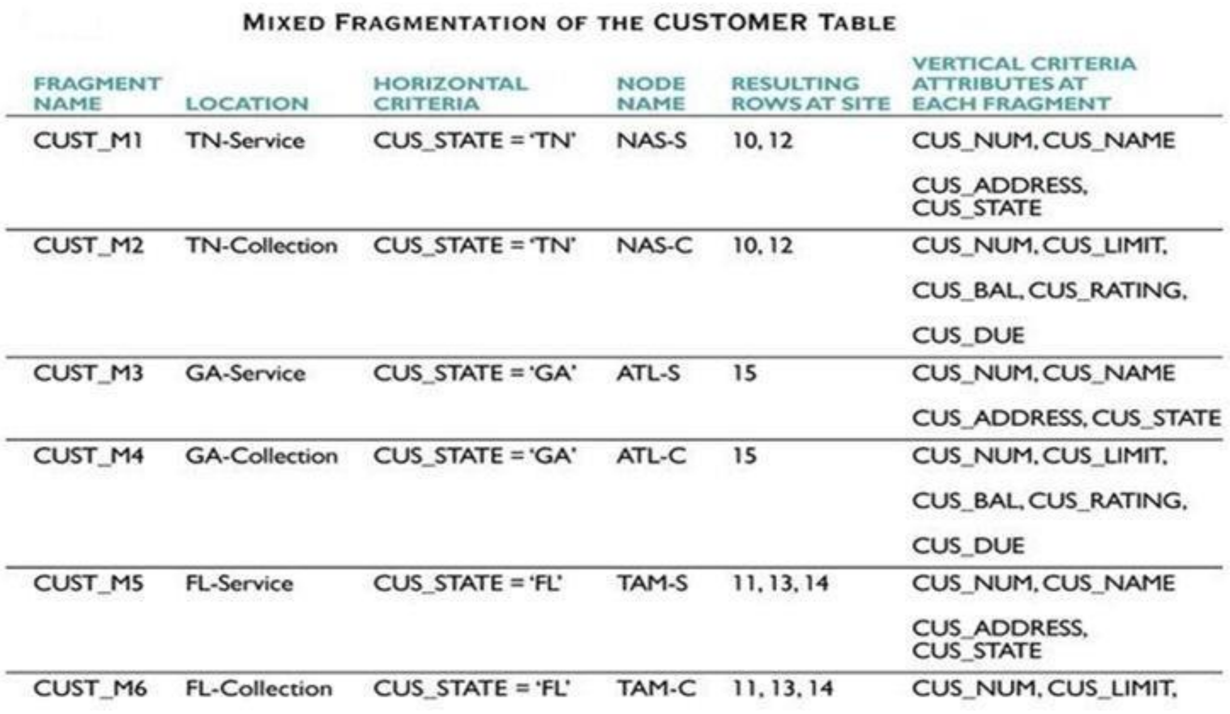
Fragment name: CUST_V2 Location: Collection Bldg. Node: ARC

CUS_NUM	CUS_LIMIT	CUS_BAL	CUS_RATING	CUS_DUE
10	$3,500.00	$2,700.00	3	$1,245.00
11	$6,000.00	$1,200.00	1	$0.00
12	$4,000.00	$3,500.00	3	$3,400.00
13	$6,000.00	$5,890.00	3	$1,090.00
14	$1,200.00	$550.00	1	$0.00
15	$2,000.00	$350.00	2	$50.00

VERTICALLY FRAGMENTED TABLE CONTENTS

Mixed Fragmentation

The XYZ Company's structure requires that the CUSTOMER data be fragmented horizontally to accommodate the various company locations; within the locations, the data must be fragmented vertically to accommodate the two departments (service and collection). In short, the CUSTOMER table requires mixed fragmentation. Mixed fragmentation requires a two-step procedure. First, horizontal fragmentation is introduced for each site based on the location within a state (CUS_STATE). The horizontal fragmentation yields the subsets of customer tuples (horizontal fragments) that are located at each site. Because the departments are located in different buildings, vertical fragmentation is used within each horizontal fragment to divide the attributes, thus meeting each department's information needs at each sub site. Mixed fragmentation yields the results displayed in the following Table.



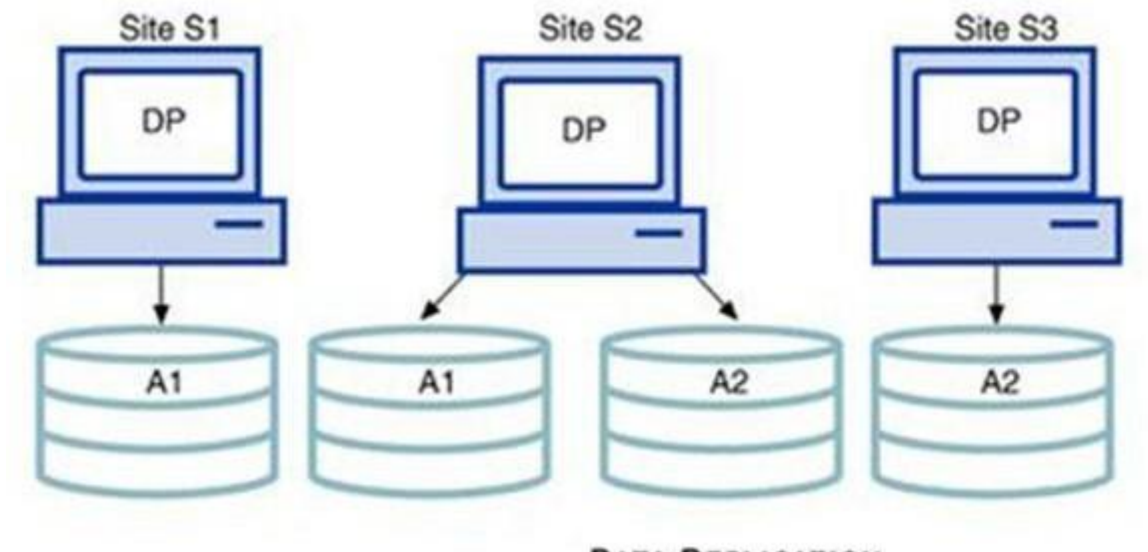
MIXED FRAGMENTATION OF THE CUSTOMER TABLE

FRAGMENT NAME	LOCATION	HORIZONTAL CRITERIA	NODE NAME	RESULTING ROWS AT SITE	VERTICAL CRITERIA ATTRIBUTES AT EACH FRAGMENT
CUST_M1	TN-Service	CUS_STATE = 'TN'	NAS-S	10, 12	CUS_NUM, CUS_NAME CUS_ADDRESS, CUS_STATE
CUST_M2	TN-Collection	CUS_STATE = 'TN'	NAS-C	10, 12	CUS_NUM, CUS_LIMIT, CUS_BAL, CUS_RATING, CUS_DUE
CUST_M3	GA-Service	CUS_STATE = 'GA'	ATL-S	15	CUS_NUM, CUS_NAME CUS_ADDRESS, CUS_STATE
CUST_M4	GA-Collection	CUS_STATE = 'GA'	ATL-C	15	CUS_NUM, CUS_LIMIT, CUS_BAL, CUS_RATING, CUS_DUE
CUST_M5	FL-Service	CUS_STATE = 'FL'	TAM-S	11, 13, 14	CUS_NUM, CUS_NAME CUS_ADDRESS, CUS_STATE
CUST_M6	FL-Collection	CUS_STATE = 'FL'	TAM-C	11, 13, 14	CUS_NUM, CUS_LIMIT,

Data Replication:

Data replication refers to the storage of data copies at multiple sites served by a computer network. Fragment copies can be stored at several sites to serve specific information requirements. Because the existence of fragment copies can enhance data availability and response time, data copies can help to reduce communication and total query costs.

Suppose database A is divided into two fragments, A1 and A2. Within a replicated distributed database, the scenario depicted in the following Figure is possible: fragment A1 is stored at sites S1 and S2, while fragment A2 is stored at sites S2 and S3.



DATA REPLICATION

Event Based Real Time Analysis Tools

Each real-time design concern for software must be applied in the context of system ***performance***. In most cases, the performance of a real-time system is measured as one or more time-related characteristics, but other measures such as fault-tolerance may also be used.

Some real-time systems are designed for applications in which only the response time or the data transfer rate is critical. Other real-time applications require optimization of both parameters under peak loading conditions. What's more, real-time systems must handle their peak loads while performing a number of simultaneous tasks.

Since the performance of a real-time system is determined primarily by the system response time and its data transfer rate, it is important to understand these two parameters. System *response time* is the time within which a system must detect an internal or external event and respond with an action. Often, event detection and response generation are simple. It is the processing of the information about the event to determine the appropriate response that may involve complex, time-consuming algorithms.

Among the key parameters that affect the response time are *context switching* and *interrupt latency*.Context switching involves the time and overhead to switch among tasks, and interrupt latency is the time lag before the switch is actually possible. Other parameters that affect response time are the speed of computation and of access to mass storage.

The *data transfer rate* indicates how fast serial or parallel, as well as analog or digital data must be moved into or out of the system. Hardware vendors often quote timing and capacity values for performance characteristics. However, hardware specifications for performance are usually measured in isolation and are often of little value in determining overall real-time system performance. Therefore, I/O device performance, bus latency, buffer size, disk performance, and a host of other factors, although important, are only part of the story of real-time system design.

Real-time systems are often required to process a continuous stream of incoming data. Design

must assure that data are not missed. In addition, a real-time system must respond to events that are asynchronous. Therefore, the arrival sequence and data volume cannot be easily predicted in advance.

Although all software applications must be reliable, real-time systems make special demands on reliability, restart, and fault recovery. Because the real-world is being monitored and controlled, loss of monitoring or control (or both) is intolerable in many circumstances (e.g., an air traffic control system). Consequently, real-time systems contain restart and fault-recovery mechanisms and frequently have built-in redundancy to insure backup.

The need for reliability, however, has spurred an on-going debate about whether *on-line* systems, such as airline reservation systems and automatic bank tellers, also qualify as real-time. On one hand, such on-line systems must respond to external interrupts within prescribed response times on the order of one second. On the other hand, nothing catastrophic occurs if an on-line system fails to meet response requirements; instead, only system degradation results.