Index:-

Sr.No.	***Aim of the Practical***:-	***Date***	***Signature***
01.	Write a Program to print different "*" Patterns.		
02.	Write a Program to print the Type of Triangle By their sides.		
03.	To Study About Software Development Life Cycle.		
04.	To Study About Functional Point Analysis.		
05.	To Study Software Testing, Blackbox and Whitebox Testing and Different Types of Testing.		
06.	To Study Critical Path Method (Cpm), Project Evaluation and Review Technique (Pert), Gantt Chart And Work-Breakdown Structure In Software Projects.		
07.	To Study Data Flow Diagrams (Dfds) and Levels In Dfds.		
08.	To Study the Risk Management During The Software Development.		

PRACTICAL NO:-

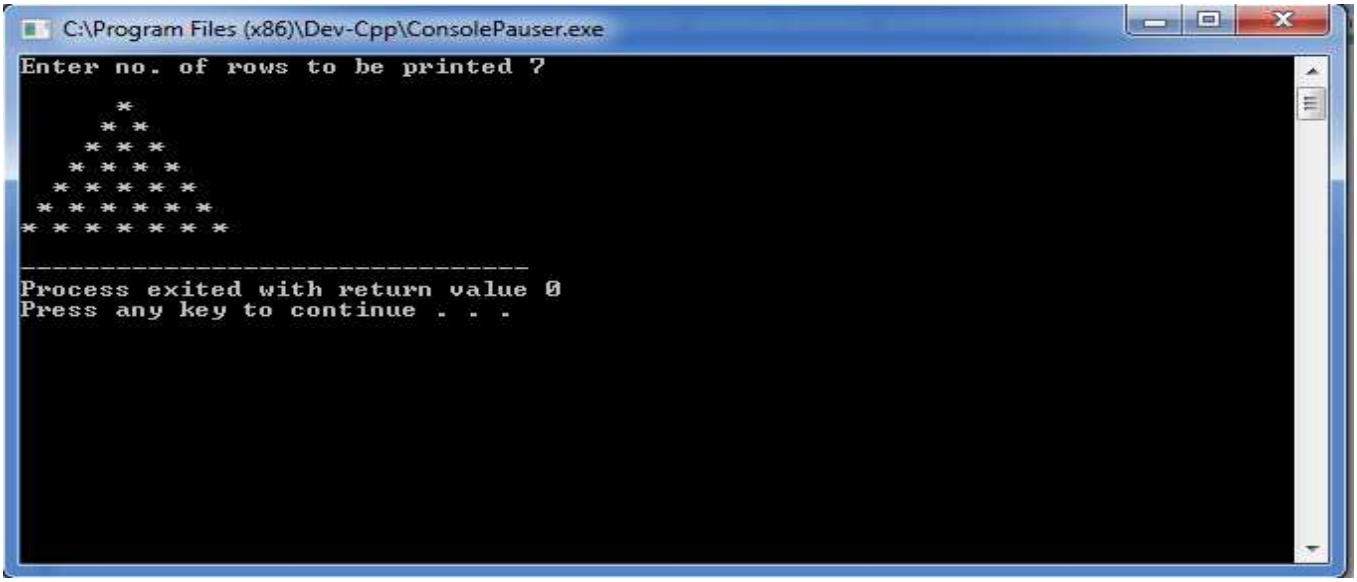
AIM:- Write a Program to print different "*" Patterns .

1.

SOURCE CODE:-

```
//Program to Print Triangle pattern of Stars…
#include<stdio.h>
#include<conio.h>                                   //for clrscr( )
int main( )
{
      clrscr( );
      int i, j, k, n;
      printf("Enter no. of rows to be printed\t");
      scanf("%d",&n);
      for (i=0; i<=n; i++)                          //Loop to print no. of  Rows
       {   for (j=n-i;j>=1;j--)                     //Loop to print spaces..
       {
          printf(" ");
       }
     for (k=1; k<=i; k++)                           //Loop to print Star patterns.
     {
       printf("* ");
     }
   printf("\n");
  }
getch( );
}
```

OUTPUT:-

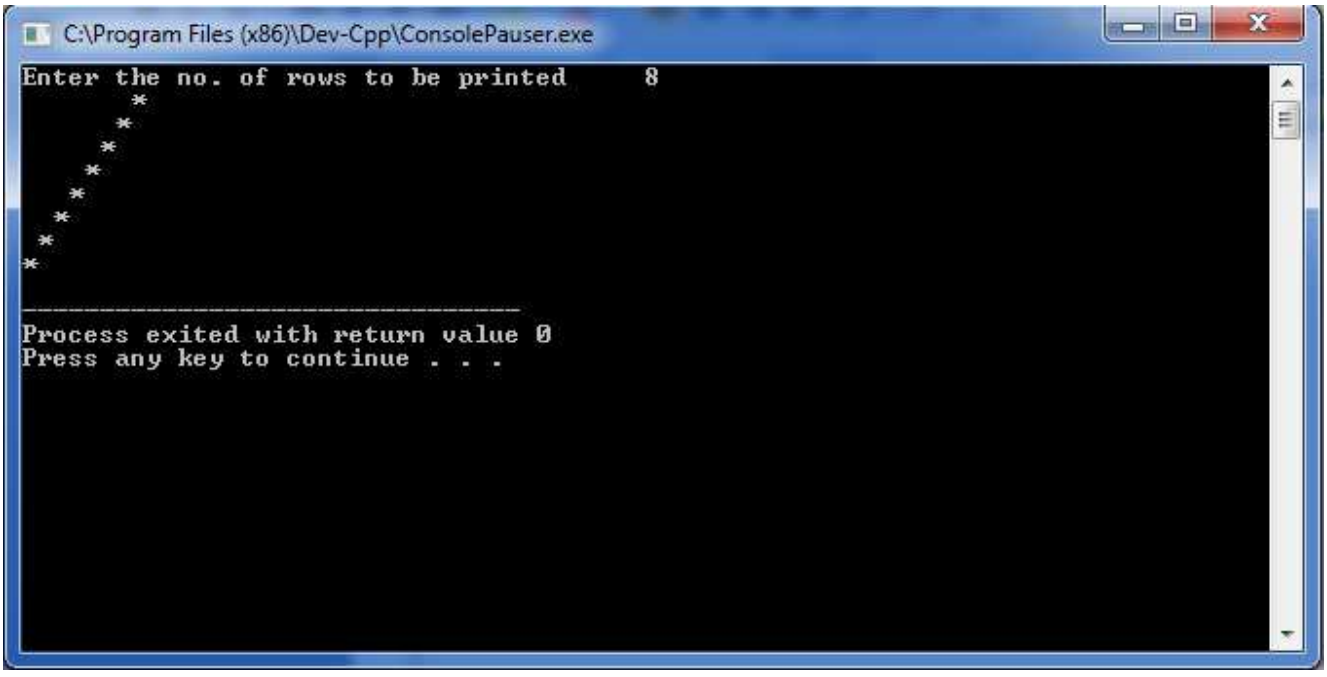


2.

SOURCE CODE:-

```
//Program to Print Diagonal pattern of Stars…
#include<stdio.h>
#include<conio.h>                                   //for clrscr( )
int main( )
{
    clrscr( );
    int n, temp, i, j;
    printf("Enter the no. of rows to be printed\t");
    scanf("%d",&n);
    temp=n;
   for (i=0; i<n; i++)                              //For Rows to be printed.
   {
     for (j=1; j<temp; j++)                         //For Spaces Before Star.
       printf(" ");
       temp--;
       printf("*");
       printf("\n");
   }
   return 0;
   getch( );
}
```

OUTPUT:-



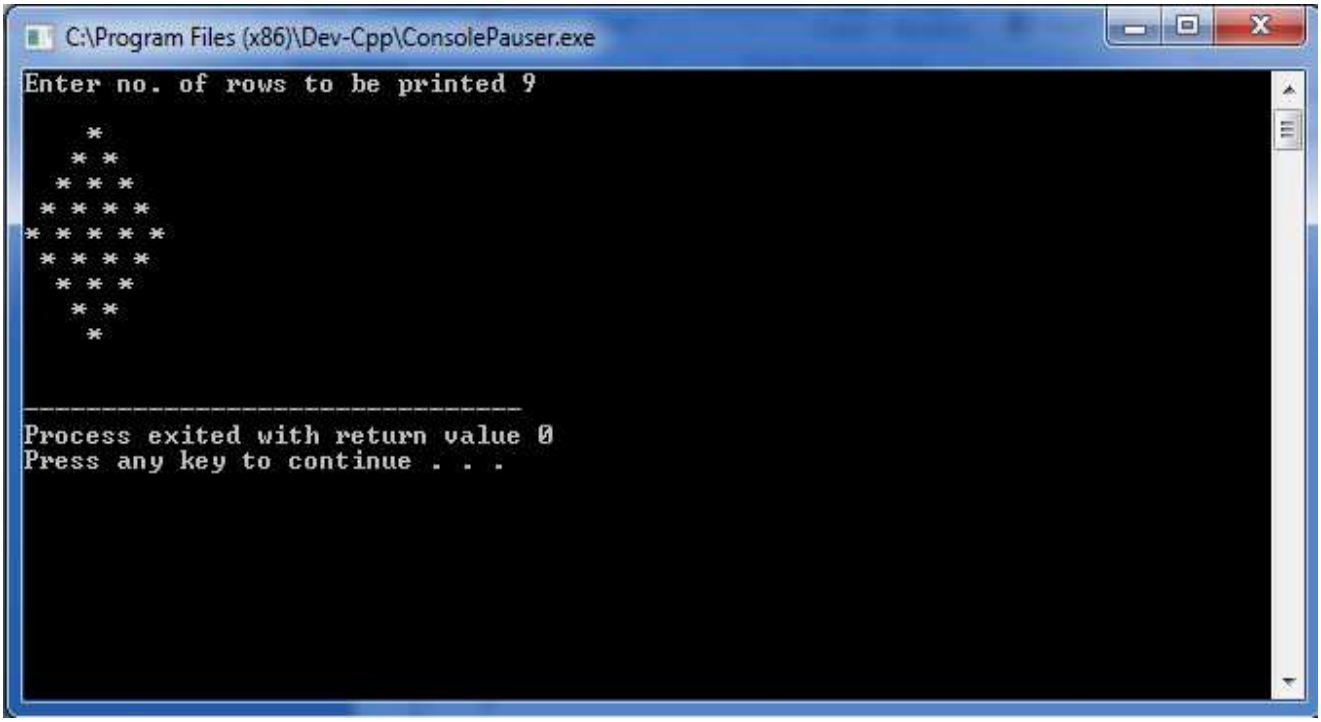
3.

SOURCE CODE:-

//Program to Print Diamond pattern of Stars…

```
#include<stdio.h>
#include<conio.h>                                   //for clrscr( )
int main( )
{
   int i, j, k, n, n1, n2;
  printf("Enter no. of rows to be printed\t");
  scanf("%d",&n);
  n1=(n+1)/2;
//Loop to Print Upper half of the Diamond..

  for (i=0; i<=n1; i++)                              //Loop to print no. of  rows.
  {
    for (j=n1-i; j>=1; j--)                          //Loop to print no. of Spaces.
    {
       printf(" ");
    }
    for (k=1; k<=i; k++)                             //Loop to print Star Pattern.
    {
      printf("* ");
    }
    printf("\n");
  }
  n2=n1-1;
//Loop to print Lower Half of Diamond..

  for (i=n2; i>=0; i--)
  {
    for (j=(n1-i); j>=1; j--)
    {
       printf(" ");
    }
    for (k=i; k>=1; k--)
    {
      printf("* ");
    }
    printf("\n");
  }
  getch( );
}
```

OUTPUT:-

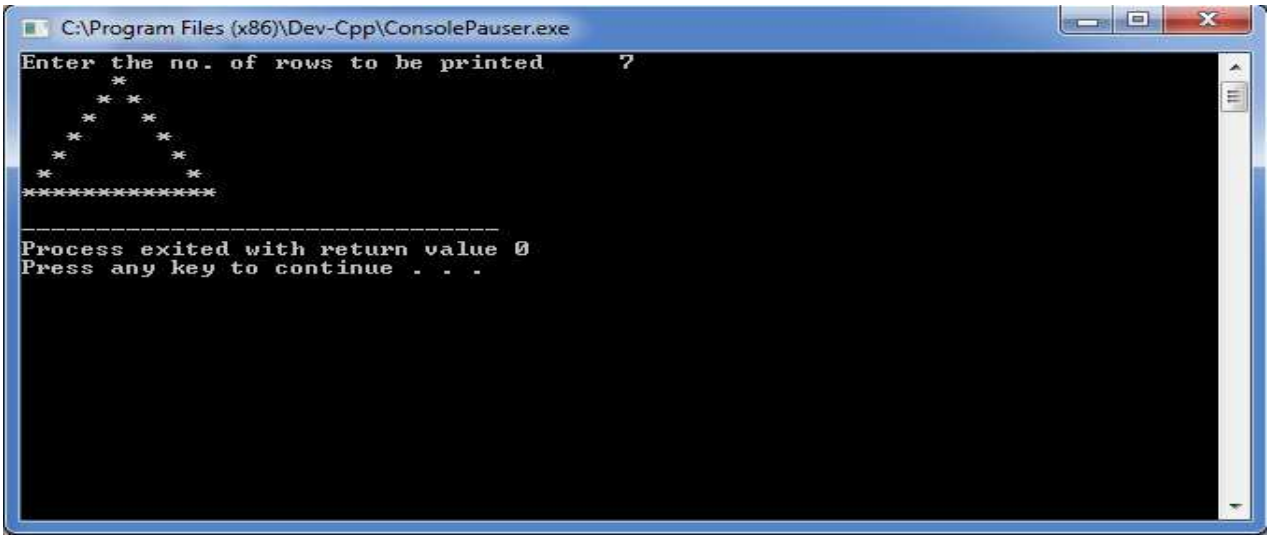

C:\Program Files (x86)\Dev-Cpp\ConsolePauser.exe
Enter no. of rows to be printed 9
*
* *
* * *
* * * *
* * * * *
* * * *
* * *
* *
*
Process exited with return value 0
Press any key to continue . . .

4.

SOURCE CODE:-

```
//Program to Print Triangle pattern of Stars space in between…
#include<stdio.h>                                   //Header file.
#include<conio.h>                                   //For clrscr( )..
int main( )
{
    clrscr( );
    int n, temp, i, j;
    printf("Enter the no. of rows to be printed\t");
    scanf("%d",&n);
    temp=n;
    for (i=0; i<n; i++)                             //For no of Rows
   {
     for (j=1; j<temp; j++)                         //For Spaces before Star
       printf(" ");
       temp--;
       printf("*");
     for (j=1; j<=(2*i-1); j++)                     //For Making Triangle By Star.
      {
            if(i==(n-1))
            printf("*");
            else
            printf(" ");
      }
     if(i>0)
     printf("*");
     printf("\n");
   }
   return 0;
}
```

OUTPUT:-



PRACTICAL NO:-

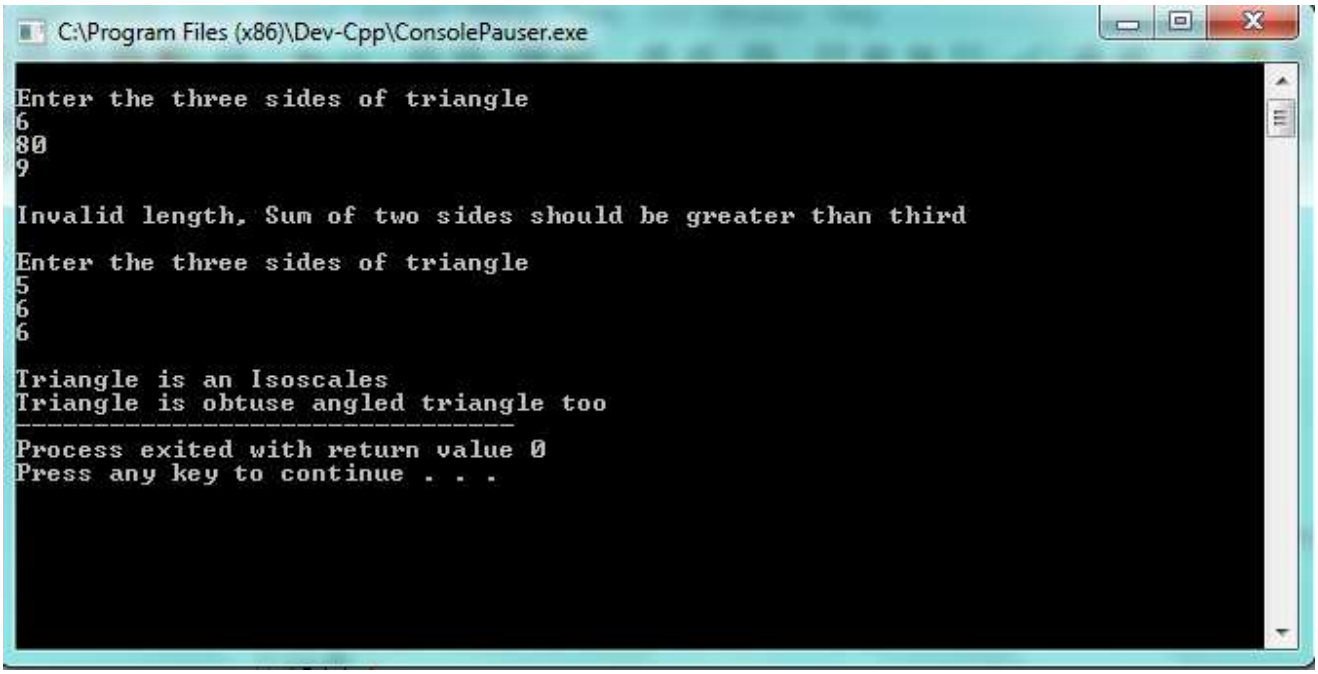
AIM:- Write a Program to print the Type of Triangle By their sides .

1.

SOURCE CODE:-

```
//Program to Print The type of Triangle…
#include<stdio.h>
#include<conio.h>                                          //for clrscr( )
int main( )
{
     clrscr( );
      int a, b, c;
     printf("Enter the three sides of triangle\n");
     scanf("%u%u%u",&a,&b,&c);
     if(a<=0||b<=0||c<=0)                                  //condition to Check negative sides.
       {
     printf("\nInvalid length:Please Enter the positive values");
     main( );                                              //To Recall The main function.
       }
     else if((a+b)<c||(b+c)<a||(c+a)<b)
       {
     printf("\nInvalid length, Sum of two sides should be greater than third");
     main( );
       }
     if((a==b)&&(b==c))
     printf("\nTriangle is an Equilateral");

     else if((a==b)||(b==c)||(c==a))
      printf("\nTriangle is an Isoscales");

     else
     printf("\nTriangle is scaler");

     if((a*a==b*b+c*c)||(b*b==a*a+c*c)||(c*c==a*a+b*b))     //For right angled Triangle.
      printf("\nTriangle is right angled triangle too");

     else if((a*a>=b*b+c*c)||(b*b>=a*a+c*c)||(c*c>=a*a+b*b))     //For obtuse angled Triangle
      printf("\nTriangle is obtuse angled triangle too");

     else if((a*a<b*b+c*c)||(b*b<a*a+c*c)||(c*c<a*a+b*b))     //For Acute angled Triangle
     printf("\nTriangle is acute angled triangle too");

     getch( );
     return 0;
     }
```

OUTPUT:-


C:\Program Files (x86)\Dev-Cpp\ConsolePauser.exe
Enter the three sides of triangle
6
80
9
Invalid length, Sum of two sides should be greater than third
Enter the three sides of triangle
5
6
6
Triangle is an Isoscales
Triangle is obtuse angled triangle too
Process exited with return value 0
Press any key to continue . . .


C:\Program Files (x86)\Dev-Cpp\ConsolePauser.exe
Enter the three sides of triangle
3
4
5
Traiangle is scaler
Triangle is right angled triangle too
Process exited with return value 0
Press any key to continue . . .

PRACTICAL NO:-

AIM:- To Study About Software Development Life Cycle.

SOFTWARE DEVELOPMENT LIFE CYCLE:

The development models are the various processes or methodologies that are being selected for the development of the project depending on the project's aims and goals. There are many development life cycle models that have been developed in order to achieve different required objectives. The models specify the various stages of the process and the order in which they are carried out.

The selection of model has very high impact on the testing that is carried out. It will define the what, where and when of our planned testing, influence regression testing and largely determines which test techniques to use.

There are various Software development models or methodologies. They are as follows:

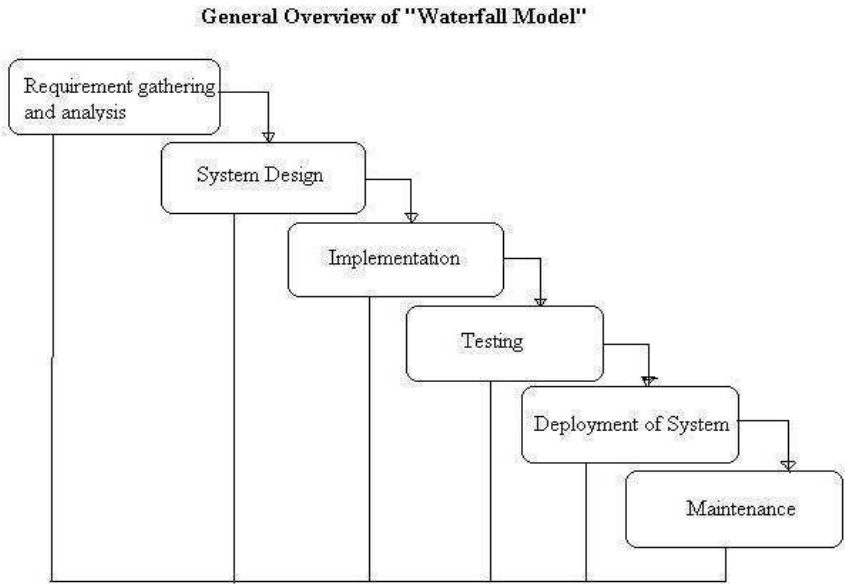
1. Waterfall Model
2. Prototype Model
3. Incremental Model
4. Iterative Model
5. Spiral Model

WATERFALL MODEL:

The Waterfall Model was first Process Model to be introduced. It is very simple to understand and use. In a waterfall model, each phase must be completed fully before the next phase can begin. At the end of each phase, a review takes place to determine if the project is on the right path and whether or not to continue or discard the project. In waterfall model phases do not overlap. Waterfall Model contains the following stages:

- Requirement gathering and analysis
- System Design
- Implementation
- Testing
- Deployment of system
- maintenance

DIAGRAM OF WATERFALL MODEL:



ADVANTAGES OF WATERFALL MODEL:

- Simple and easy to understand and use.
- Easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.

DISADVANTAGES OF WATERFALL MODEL:

- Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.

WHEN TO USE THE WATERFALL MODEL:

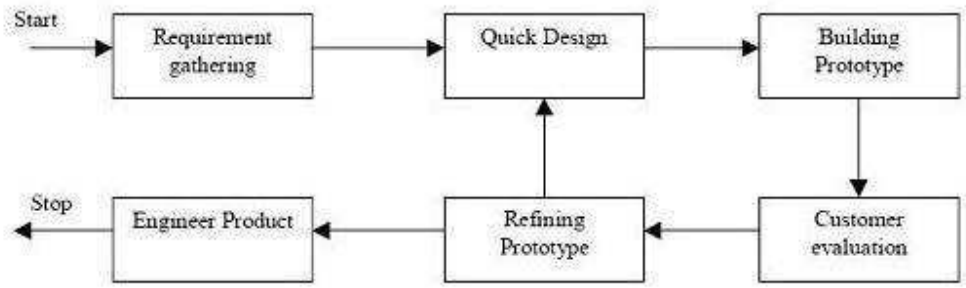
- Requirements are very well known, clear and fixed.
- Product definition is stable.

- Technology is understood.
- There are no ambiguous requirements
- Ample resources with required expertise are available freely
- The project is short.

PROTOTYPE MODEL:

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

DIAGRAM OF PROTOTYPE MODEL:



Prototyping Model

ADVANTAGES OF PROTOTYPE MODEL:

- Users are actively involved in the development.
- Since in this methodology a working model of the system is provided, the users get a better understanding of the system being developed.
- Errors can be detected much earlier.
- Quicker user feedback is available leading to better solutions.
- Missing functionality can be identified easily.

DISADVANTAGES OF PROTOTYPE MODEL:

- Leads to implementing and then repairing way of building systems.
- Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.

- Incomplete application may cause application not to be used as the full system was designed.
- Incomplete or inadequate problem analysis.

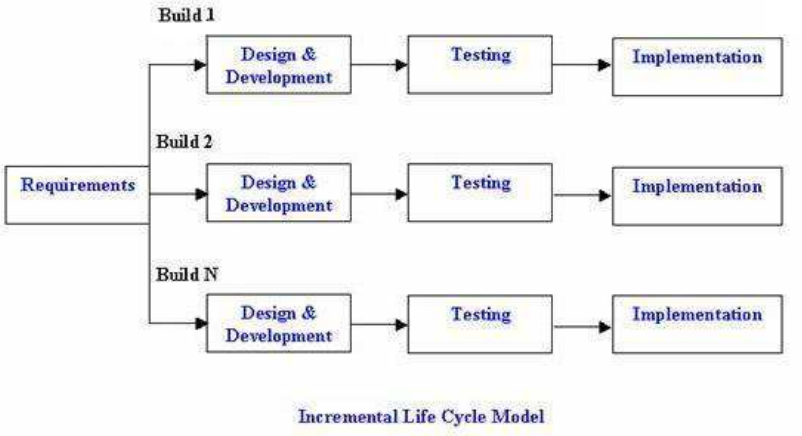
WHEN TO USE PROTOTYPE MODEL:

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

INCREMENTAL MODEL:

In incremental model the whole requirement is divided into various builds. Multiple development cycles take place. Cycles are divided up into smaller, more easily managed modules. Each module passes through the requirements, design, implementation and testing phases. A working version of software is produced during the first module, so you have working software early on during the software life cycle. Each subsequent release of the module adds function to the previous release. The process continues till the complete system is achieved.

DIAGRAM OF INCREMENTAL MODEL:



Incremental Life Cycle Model

ADVANTAGES OF INCREMENTAL MODEL:

- Generates working software quickly and early during the software life cycle.
- More flexible – less costly to change scope and requirements.

- Easier to test and debug during a smaller iteration.
- Customer can respond to each built.
- Lowers initial delivery cost.

DISADVANTAGES OF INCREMENTAL MODEL:

- Needs good planning and design.
- Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
- Total cost is higher than waterfall.

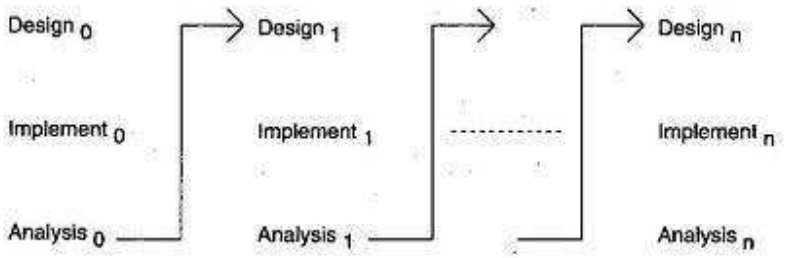
WHEN TO USE THE INCREMENTAL MODEL:

- Requirements of the complete system are clearly defined and understood.
- Major requirements must be defined; however, some details can evolve with time.
- There is a need to get a product to the market early.
- A new technology is being used.

ITERATIVE MODEL:

An iterative life cycle model does not attempt to start with a full specification of requirements. Instead, development begins by specifying and implementing just part of the software, which can then be reviewed in order to identify further requirements. This process is then repeated, producing a new version of the software for each cycle of the model.

DIAGRAM OF ITERATIVE MODEL:



ADVANTAGES OF ITERATIVE MODEL:

- In iterative model we can only create a high-level design of the application before we actually begin to build the product and define the design solution for the entire product. Later on we can design and built a skeleton version of that, and then evolved the design based on what had been built.
- In iterative model we build and improve the product step by step. Hence we can track the defects at early stages. This avoids the downward flow of the defects.

- Reliable user feedback. When presenting sketches and blueprints of the product to users for their feedback, we are effectively asking them to imagine how the product will work.
- Less time is spent on documenting and more time is given for designing.

DISADVANTAGES OF ITERATIVE MODEL:

- Each phase of an iteration is rigid with no overlaps.
- Costly system architecture or design issues may arise because not all requirements are gathered up front for the entire lifecycle.

WHEN TO USE ITERATIVE MODEL:

- Requirements of the complete system are clearly defined and understood.
- When the project is big.
- Major requirements must be defined; however, some details can evolve with time.

SPIRAL MODEL:

The spiral model is similar to the incremental model, with more emphasis placed on risk analysis. The spiral model has four phases: Planning, Risk Analysis, Engineering and Evaluation. A software project repeatedly passes through these phases in iterations (called Spirals in this model). The baseline spiral, starting in the planning phase, requirements are gathered and risk is assessed. Each subsequent spirals builds on the baseline spiral. Requirements are gathered during the planning phase. In the risk analysis phase, a process is undertaken to identify risk and alternate solutions. A prototype is produced at the end of the risk analysis phase. Software is produced in the engineering phase, along with testing at the end of the phase. The evaluation phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.

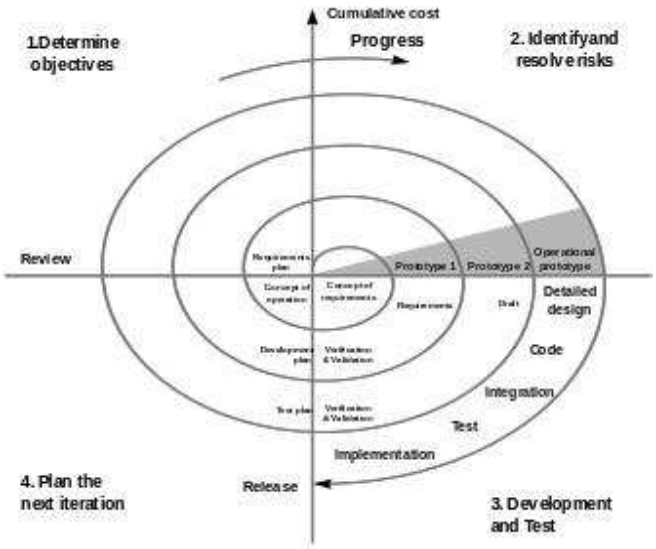
ADVANTAGES OF SPIRAL MODEL:

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

DISADVANTAGES OF SPIRAL MODEL:

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

DIAGRAM OF SPIRAL MODEL:



WHEN TO USE SPIRAL MODEL:

- When costs and risk evaluation is important.
- For medium to high-risk projects.
- Users are unsure of their needs.
- Requirements are complex.
- New product line.
- Significant changes are expected (research and exploration).

PRACTICAL NO:-

AIM: To Study About Functional Point Analysis.

FUNCTIONAL POINT ANALYSIS:

Function Point Analysis is a structured technique of problem solving. It is a method to break systems into smaller components, so they can be better understood and analyzed.

Function points are a unit measure for software much like an hour is to measuring time, miles are to measuring distance or Celsius is to measuring temperature. Function Points are an ordinal measure much like other measures such as kilometers, Fahrenheit, hours, so on and so forth.

The functional user requirements of the software are identified and each one is categorized into one of five types: outputs, inquiries, inputs, internal files, and external interfaces. Once the function is identified and categorized into a type, it is then assessed for complexity and assigned a number of function points.

OBJECTIVES OF FUNCTION POINT ANALYSIS

Since Function Points measures systems from a functional perspective they are independent of technology. Regardless of language, development method, or hardware platform used, the number of function points for a system will remain constant. The only variable is the amount of effort needed to deliver a given set of function points; therefore, Function Point Analysis can be used to determine whether a tool, an environment, a language is more productive compared with others within an organization or among organizations. This is a critical point and one of the greatest values of Function Point Analysis.

Function Point Analysis can provide a mechanism to track and monitor scope creep. Function Point Counts at the end of requirements, analysis, design, code, testing and implementation can be compared. The function point count at the end of requirements and/or designs can be compared to function points actually delivered. If the project has grown, there has been scope creep. The amount of growth is an indication of how well requirements were gathered by and/or communicated to the project team. If the amount of growth of projects declines over time it is a natural assumption that communication with the user has improved.

FUNCTION POINT CALCULATION

A function point is a rough estimate of a unit of delivered functionality of a software project. Function points (FP) measure size in terms of the amount of functionality in a system. Function points are computed by first calculating an unadjusted function point count (UFC). Counts are made for the following categories

- **Number of user inputs:** Each user input that provides distinct application oriented data to the software is counted.

- **Number of user outputs:** Each user output that provides application oriented information to the user is counted. In this context "output" refers to reports, screens, error messages, etc. Individual data items within a report are not counted separately.
- **Number of user inquiries:** An inquiry is defined as an on-line input that results in the generation of some immediate software response in the form of an on-line output. Each distinct inquiry is counted.
- **Number of files:** Each logical master file is counted.
- **Number of external interfaces:** All machine-readable interfaces that are used to transmit information to another system are counted.

Once this data has been collected, a complexity rating is associated with each count according to the following table:

Function point complexity weights.

Measurement parameter	Weighting factor		
	Simple	Average	Complex
Number of user inputs	3	4	6
Number of user outputs	4	5	7
Number of user inquiries	3	4	6
Number of files	7	10	15
Number of external interfaces	5	7	10

Each count is multiplied by its corresponding complexity weight and the results are summed to provide the UFC. The adjusted function point count (FP) is calculated by multiplying the UFC by a technical complexity factor (TCF) also referred to as Value Adjustment Factor (VAF). Components of the TCF are listed in below given table.

Components of the technical complexity factor.

F1	Reliable back-up and recovery	F2	Data communications
F3	Distributed functions	F4	Performance
F5	Heavily used configuration	F6	Online data entry
F7	Operational ease	F8	Online update
F9	Complex interface	F10	Complex processing
F11	Reusability	F12	Installation ease
F13	Multiple sites	F14	Facilitate change

Alternatively the following questionnaire could be utilized

1. Does the system require reliable backup and recovery?
2. Are data communications required?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing, heavily utilized operational environment?

6. Does the system require on-line data entry?
7. Does the on-line data entry require the input transaction to be built over multiple screens or operations?
8. Are the master files updated online?
9. Are the input, outputs, files or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Are conversions and installation included in the design?
13. Is the system designed for multiple installations in different organizations?
14. Is the applications designed to facilitate change and ease of use?

ADVANTAGES OF FUNCTIONAL POINT ANALYSIS:

- Function Points can be used to size software applications accurately. Sizing is an important component in determining productivity (outputs/inputs).
- They can be counted by different people, at different times, to obtain the same measure within a reasonable margin of error.
- Function Points are easily understood by the non-technical user. This helps communicate sizing information to a user or customer.
- Function Points can be used to determine whether a tool, a language, an environment, is more productive when compared with others.

PRACTICAL NO:-

AIM: To Study Software Testing, Blackbox and Whitebox Testing and Different Types of Testing.

DEFINITION:

Software testing is performed to verify that the completed software package functions according to the expectations defined by the requirements/specifications. The overall objective to not to find every software bug that exists, but to uncover situations that could negatively impact the customer, usability and/or maintainability. Software testing is the process of evaluating a software item to detect differences between given input and expected output. Testing assesses the quality of the product. Software testing is a process that should be done during the development process. In other words software testing is a verification and validation process.

There are two basics of software testing: blackbox testing and whitebox testing.

BLACKBOX TESTING:

Black box testing is a testing technique that ignores the internal mechanism of the system and focuses on the output generated against any input and execution of the system. It is also called functional testing.

The technique of testing without having any knowledge of the interior workings of the application is Black Box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, when performing a black box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

Advantages:

- Well suited and efficient for large code segments.
- Code Access not required.
- Clearly separates user's perspective from the developer's perspective through visibly defined roles.
- Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language or operating systems.

Disadvantages:

- Limited Coverage since only a selected number of test scenarios are actually performed.
- Inefficient testing, due to the fact that the tester only has limited knowledge about an application.
- Blind Coverage, since the tester cannot target specific code segments or error prone areas.

- The test cases are difficult to design.

WHITEBOX TESTING:

White box testing is a testing technique that takes into account the internal mechanism of a system.
It is also called structural testing and glass box testing.

White box testing is the detailed investigation of internal logic and structure of the code. White box testing is also called glass testing or open box testing. In order to perform white box testing on an application, the tester needs to possess knowledge of the internal working of the code.

The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

Advantages:

- As the tester has knowledge of the source code, it becomes very easy to find out which type of data can help in testing the application effectively.
- It helps in optimizing the code.
- Extra lines of code can be removed which can bring in hidden defects.
- Due to the tester's knowledge about the code, maximum coverage is attained during test scenario writing.

Disadvantages:

- Due to the fact that a skilled tester is needed to perform white box testing, the costs are increased.
- Sometimes it is impossible to look into every nook and corner to find out hidden errors that may create problems as many paths will go untested.
- It is difficult to maintain white box testing as the use of specialized tools like code analyzers and debugging tools are required.

Black box testing is often used for validation and white box testing is often used for verification.

DIFFERENT TYPES OF SOFTWARE TESTING:

There are many types of software testing like:

1. **Acceptance Testing:** Formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system. It is usually performed by the customer.

2. **Accessibility Testing:** Type of testing which determines the usability of a product to the people having disabilities (deaf, blind, mentally disabled etc). The evaluation process is conducted by persons having disabilities.

3. **Age Testing:** Type of testing which evaluates a system's ability to perform in the future. The evaluation process is conducted by testing teams.

4. **Alpha Testing:** Type of testing a software product or system conducted at the developer's site. Usually it is performed by the end user.

5. **Backward Compatibility Testing:** Testing method which verifies the behavior of the developed software with older versions of the test environment. It is performed by testing teams.

6. **Beta Testing:** Final testing before releasing application for commercial purpose. It is typically done by end-users or others.

7. **Bottom Up Integration Testing:** In bottom up integration testing, module at the lowest level are developed first and other modules which go towards the 'main' program are integrated and tested one at a time. It is usually performed by the testing teams.

8. **Branch Testing:** Testing technique in which all branches in the program source code are tested at least once. This is done by the developer.

9. **Compatibility Testing:** Testing technique that validates how well a software performs in a particular hardware/software/operating system/network environment. It is performed by the testing teams.

10. **Component Testing:** Testing technique similar to unit testing but with a higher level of integration - testing is done in the context of the application instead of just directly testing a specific method. Can be performed by testing or development teams.

11. **Compliance Testing:** Type of testing which checks whether the system was developed in accordance with standards, procedures and guidelines. It is usually performed by external companies which offer "Certified OGC Compliant" brand.

12. **Destructive Testing:** Type of testing in which the tests are carried out to the specimen's failure, in order to understand a specimen's structural performance or material behavior under different loads. It is usually performed by QA teams.

13. **Dynamic Testing:** Term used in software engineering to describe the testing of the dynamic behavior of code. It is typically performed by testing teams.

14. **Error-Handling Testing:** Software testing type which determines the ability of the system to properly process erroneous transactions. It is usually performed by the testing teams.

15. **Gray Box Testing:** A combination of Black Box and White Box testing methodologies: testing a piece of software against its specification but using some knowledge of its internal workings. It can be performed by either development or testing teams.

16. **Integration Testing:** The phase in software testing in which individual software modules are combined and tested as a group. It is usually conducted by testing teams.

17. **Load Testing:** Testing technique that puts demand on a system or device and measures its response. It is usually conducted by the performance engineers.

18. **Regression Testing:** Type of software testing that seeks to uncover software errors after changes to the program (e.g. bug fixes or new functionality) have been made, by retesting the program. It is performed by the testing teams.

19. **Recovery Testing:** Testing technique which evaluates how well a system recovers from crashes, hardware failures, or other catastrophic problems. It is performed by the testing teams.

20. **Unit Testing:** Software verification and validation method in which a programmer tests if individual units of source code are fit for use. It is usually conducted by the development team.

PRACTICAL NO:-

AIM: To Study Critical Path Method (Cpm), Project Evaluation and Review Technique (Pert), Gantt Chart And Work-Breakdown Structure In Software Projects.

PROJECT EVALUATION AND REVIEW TECHNIQUE (PERT):

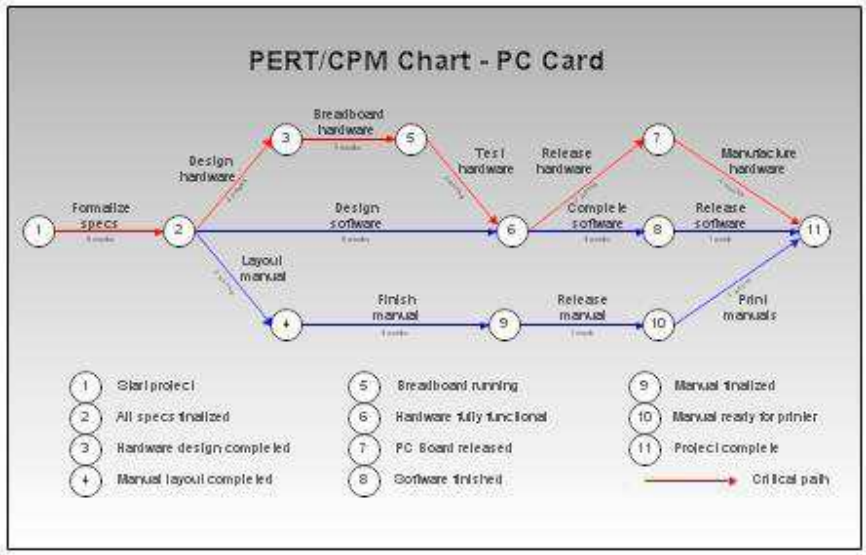
INTRODUCTION:

The Program (or Project) Evaluation and Review Technique, commonly abbreviated PERT, is a statistical tool, used in project management, that is designed to analyze and represent the tasks involved in completing a given project. First developed by the United States Navy in the 1950s, it is commonly used in conjunction with the critical path method (CPM).

TERMINOLOGY:

- **OPTIMISTIC TIME (O):** the minimum possible time required to accomplish a task, assuming everything proceeds better than is normally expected
- **PESSIMISTIC TIME (P):** the maximum possible time required to accomplish a task, assuming everything goes wrong (but excluding major catastrophes).
- **MOST LIKELY TIME (M):** the best estimate of the time required to accomplish a task, assuming everything proceeds as normal.
- **EXPECTED TIME (T_E):** the best estimate of the time required to accomplish a task, accounting for the fact that things don't always proceed as normal (the implication being that the expected time is the average time the task would require if the task were repeated on a number of occasions over an extended period of time).
    - $T_E = (O + 4M + P) \div 6$
- **SLACK:** a measure of the excess time and resources available to complete a task. It is the amount of time that a project task can be delayed without causing a delay in any subsequent tasks (free float) or the whole project (total float). Positive slack would indicate ahead of schedule; negative slack would indicate behind schedule; and zero slack would indicate on schedule.

EXAMPLE:



CRITICAL PATH METHOD (CPM):

INTRODUCTION:

The critical path method (CPM) is a step-by-step technique for process planning that defines critical and non-critical tasks with the goal of preventing time-frame problems and process bottlenecks. The CPM is ideally suited to projects consisting of numerous activities that interact in a complex manner.

Critical path is the sequential activities from start to the end of a project. Although many projects have only one critical path, some projects may have more than one critical path depending on the flow logic used in the project.

If there is a delay in any of the activities under the critical path, there will be a delay of the project deliverables.

Most of the times, if such delay is occurred, project acceleration or re-sequencing is done in order to achieve the deadlines.

Critical path method is based on mathematical calculations. In the critical path method, the critical activities of a program or a project are identified. These are the activities that have a direct impact on the completion date of the project.

EXAMPLE:

CPM calculates the longest path of planned activities to logical end points or to the end of the project, and the earliest and latest that each activity can start and finish without making the project longer. This process determines which activities are "critical" (i.e., on the longest path). In project management, a critical path is the sequence of project network activities which add up to the longest overall duration. This determines the shortest time possible to complete the project. Any delay of an activity on the critical path directly impacts the planned project completion date. A project can have several, parallel, near critical paths. An additional parallel path through the network with the total durations shorter than the critical path is called a sub-critical or non-critical path.

CPM analysis tools allow a user to select a logical end point in a project and quickly identify its longest series of dependent activities (its longest path). These tools can display the critical path (and near critical path activities if desired) as a cascading waterfall that flows from the project's start (or current status date) to the selected logical end point.

In above example RED LINE shows the CPM.

WORK-BREAKDOWN STRUCTURE (WBS):

INTRODUCTION:

A work breakdown structure (WBS), in project management and systems engineering, is a deliverable oriented decomposition of a project into smaller components. It defines and groups a project's discrete work elements in a way that helps organize and define the total work scope of the project.

WBS DEVELOPMENT

A WBS is the cornerstone of effective project planning, execution, controlling, statusing, and reporting. All the work contained within the WBS is to be identified, estimated, scheduled, and budgeted. The WBS is the structure and code that integrates and relates all project work (scope, schedule, and cost). When initial project funding is received, the Project Director (PD) develops a WBS that identifies necessary funds according to the schedule and needs of the tasks in the WBS elements. The WBS is generally a multi-level framework that organizes and graphically displays elements representing work to be accomplished in logical relationships. The PD is to structure the project work into WBS elements (work packages) that are:

Definable: can be described and easily understood by project participants.

Manageable: a meaningful unit of work where specific responsibility and authority can be assigned to a responsible individual.

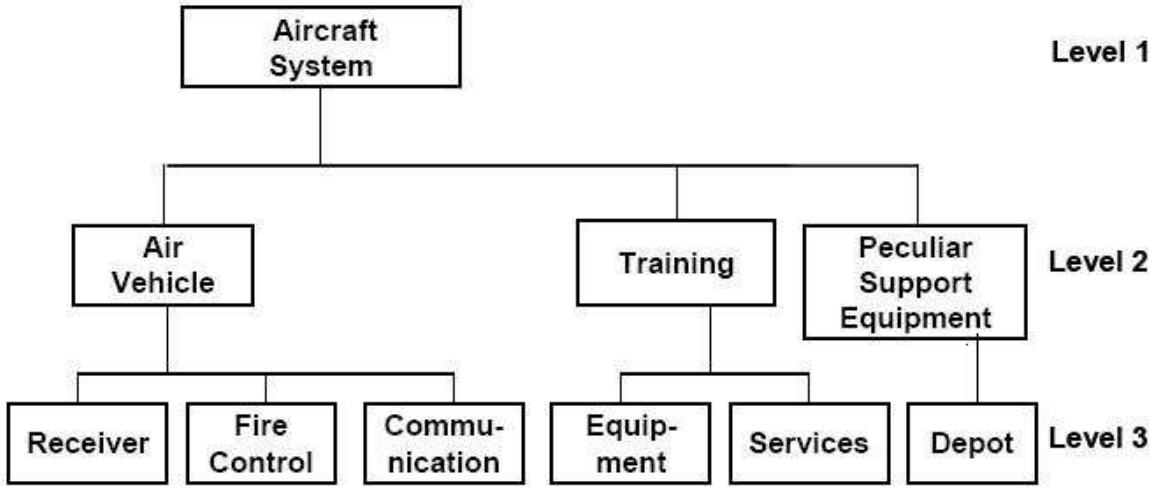
Estimateable: duration can be estimated in time required to complete, and cost can be estimated in resources required to complete.

Independent: minimum interface with or dependence on other ongoing elements (i.e., assignable to a single control account, and clearly distinguishable from other work packages).

Integratable: integrates with other project work elements and with higher level cost estimates and schedules to include the entire project.

Measurable: can be used to measure progress; has start and completion dates and measurable interim milestones.

Adaptable: sufficiently flexible so the addition/elimination of work scope can be readily accommodated in the WBS framework.



Block diag. of work break structure

GANTT CHART

INTRODUCTION:

A Gantt chart is a type of bar chart, developed by Henry Gantt in the 1910s, that illustrates a project schedule. Gantt charts illustrate the start and finish dates of the terminal elements and summary elements of a project. Terminal elements and summary elements comprise the work breakdown structure of the project. Some Gantt charts also show the dependency (i.e. precedence network) relationships between activities. Gantt charts can be used to show current schedule status using percent-complete shadings and a vertical "TODAY" line as shown here.

Although now regarded as a common charting technique, Gantt charts were considered revolutionary when first introduced. [1] In recognition of Henry Gantt's contributions, the Henry Laurence Gantt Medal is awarded for distinguished achievement in management and in

community service. This chart is also used in information technology to represent data that has been collected.

They:

- Help you to plan out the tasks that need to be completed.
- Give you a basis for scheduling when these tasks will be carried out.
- Allow you to plan the allocation of resources needed to complete the project.
- Help you to work out the critical path for a project where you must complete it by a particular date.

When a project is under way, Gantt Charts help you to monitor whether the project is on schedule. If it is not, it allows you to pinpoint the remedial action necessary to put it back on schedule.

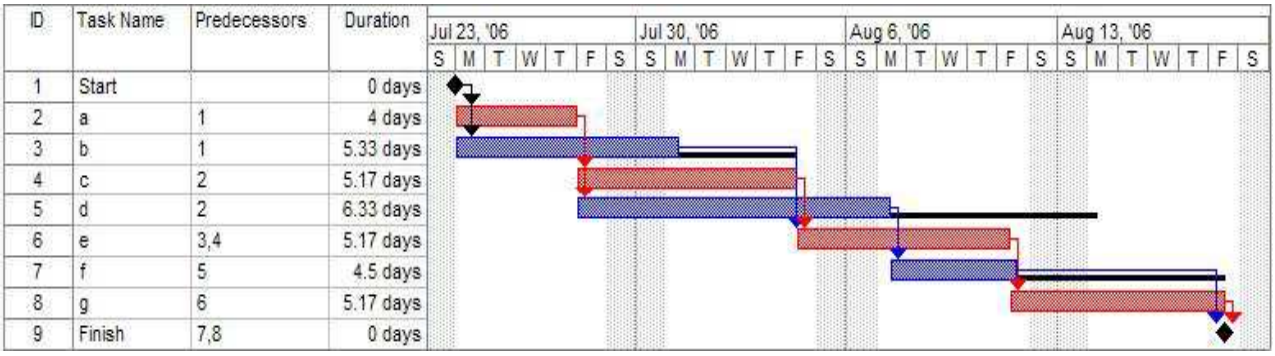
EXAMPLE:

In the following example there are seven tasks, labeled *A* through *G*. Some tasks can be done concurrently (*A* and *B*) while others cannot be done until their predecessor task is complete (cannot begin until *A* is complete). Additionally, each task has three time estimates: the optimistic time estimate (*O*), the most likely or normal time estimate (*M*), and the pessimistic time estimate (*P*). The expected time (T_E) is computed using the beta probability distribution for the time estimates, using the formula $(O + 4M + P) \div 6$.

Activity	Predecessor	Time estimates			Expected time
		Opt. (*O*)	Normal (*M*)	Pess. (*P*)	
A	—	2	4	6	4.00
B	—	3	5	9	5.33
C	*A*	4	5	7	5.17
D	*A*	4	6	10	6.33
E	*B*, *C*	4	5	7	5.17

F	*D*	3	4	8	4.50
G	*E*	3	5	8	5.17

Once this step is complete, one can draw a Gantt chart.



ID	Task Name	Predecessors	Duration
1	Start		0 days
2	a	1	4 days
3	b	1	5.33 days
4	c	2	5.17 days
5	d	2	6.33 days
6	e	3,4	5.17 days
7	f	5	4.5 days
8	g	6	5.17 days
9	Finish	7,8	0 days

Note here (1) the critical path is in red, (2) the slack is the black lines connected to non-critical activities, (3) since Saturday and Sunday are not work days and are thus excluded from the schedule; some bars on the Gantt chart are longer if they cut through a weekend.

PRACTICAL NO:-

AIM: To Study Data Flow Diagrams (Dfds) and Levels In Dfds.

INTRODUCTION:

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modelling its process aspects. Often they are a preliminary step used to create an overview of the system which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design).

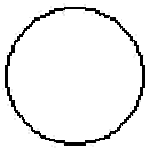
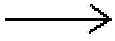
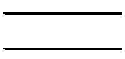
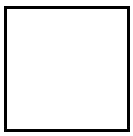
At its simplest, a data flow diagram looks at how data flows through a system. It concerns things like where the data will come from and go to as well as where it will be stored. But you won't find information about the processing timing (e.g. whether the processes happen in sequence or in parallel).

We usually begin with drawing a context diagram, a simple representation of the whole system. To elaborate further from that, we drill down to a level 1 diagram with additional information about the major functions of the system. This could continue to evolve to become a level 2 diagram when further analysis is required.

REPRESENTATION OF COMPONENTS:

DFDs only involve four symbols. They are:

- **Process**
- **Data Object**
- **Data Store**
- **External entity**

	**Process** Transform of incoming data flow(s) to outgoing flow(s).
	**Data Flow** Movement of data in the system.
	**Data Store** Data repositories for data that are not moving. It may be as simple as a buffer or a queue or as sophisticated as a relational database.
	**External Entity** Sources of destinations outside the specified system boundary.

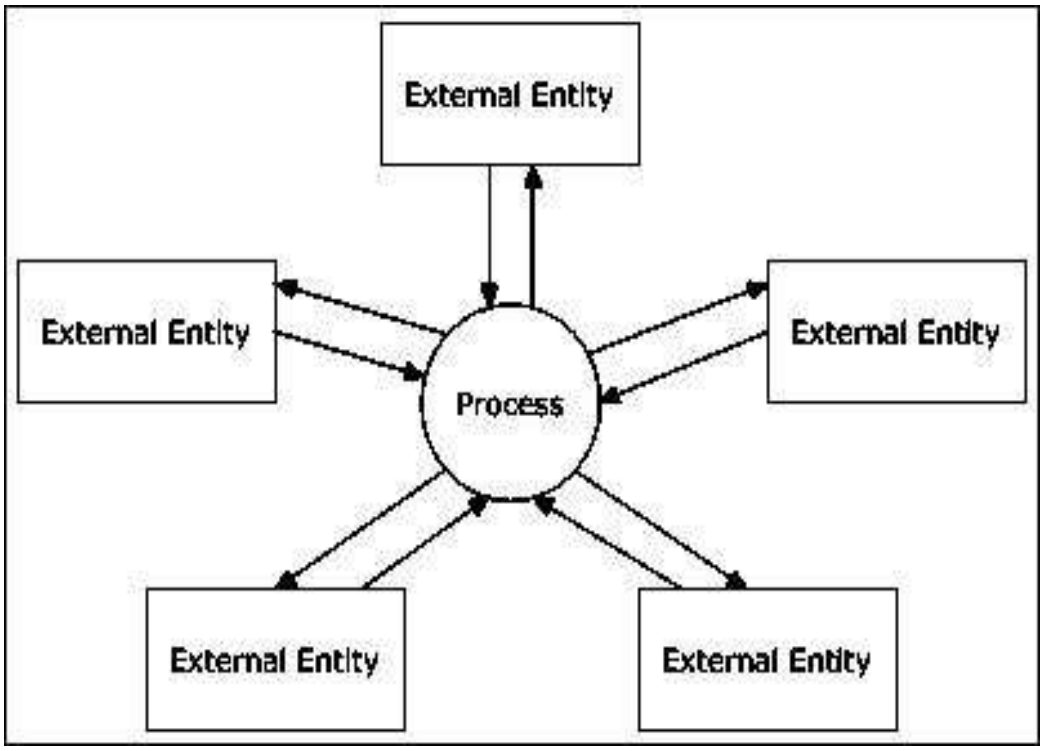
RELATIONSHIP AND RULES

RELATIONSHIP

The DFD may be used for any level of data abstraction. DFD can be partitioned into levels. Each level has more information flow and data functional details than the previous level.

LEVELS IN DFDS:

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

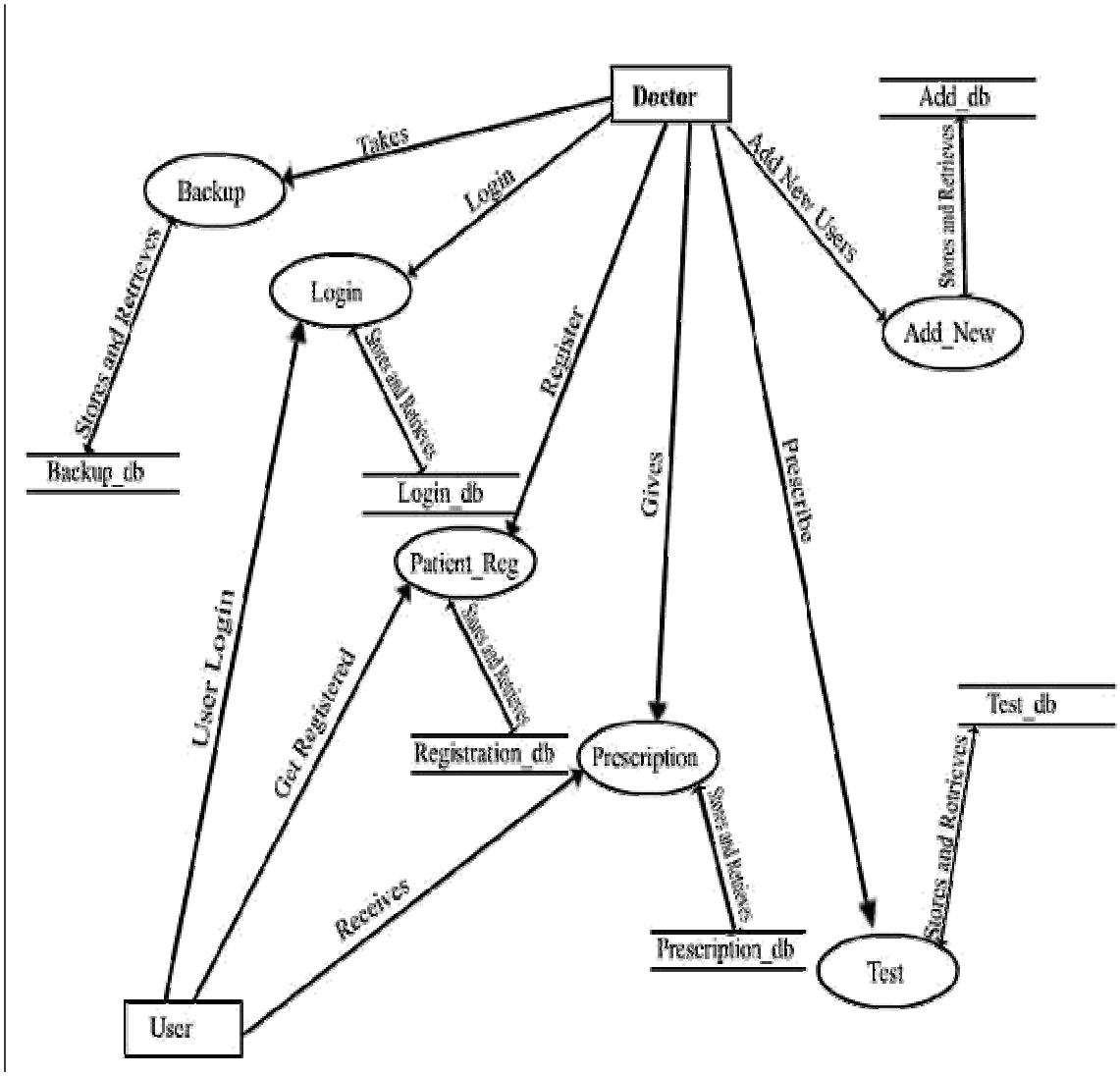


Context Diagram

LEVEL 0 DFD:

Next Level is Level 0 DFD. Some important points are:

- Level 0 DFD must balance with the context diagram it describes.
- Input going into a process are different from outputs leaving the process.
- Data stores are first shown at this level

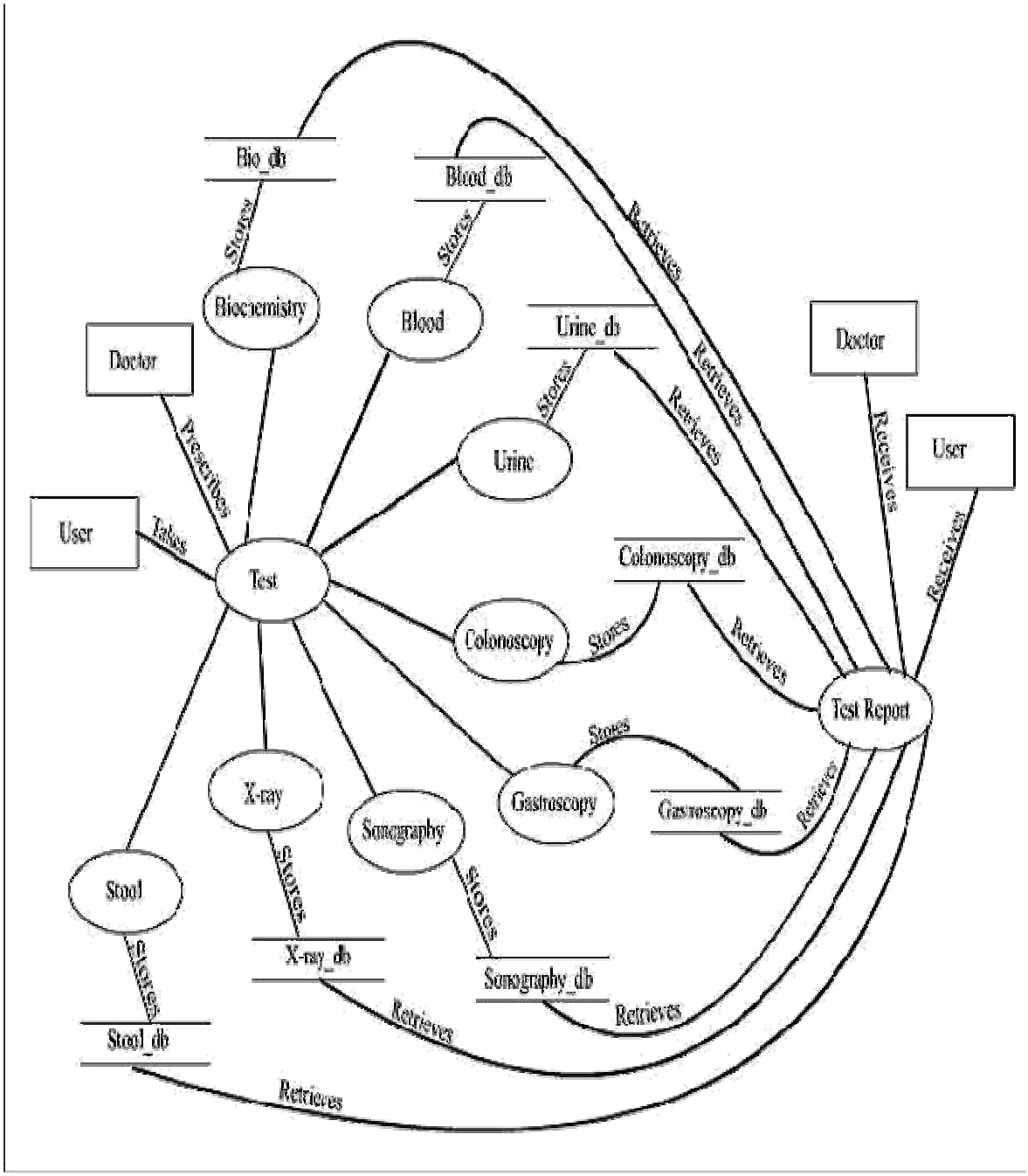


Level 0 DFD

LEVEL 1 DFD:

Next level is Level 1 DFD. Some important points are:

- Level 1 DFD must balance with the Level 0 it describes.
- Input going into a process are different from outputs leaving the process.
- Continue to show data stores.

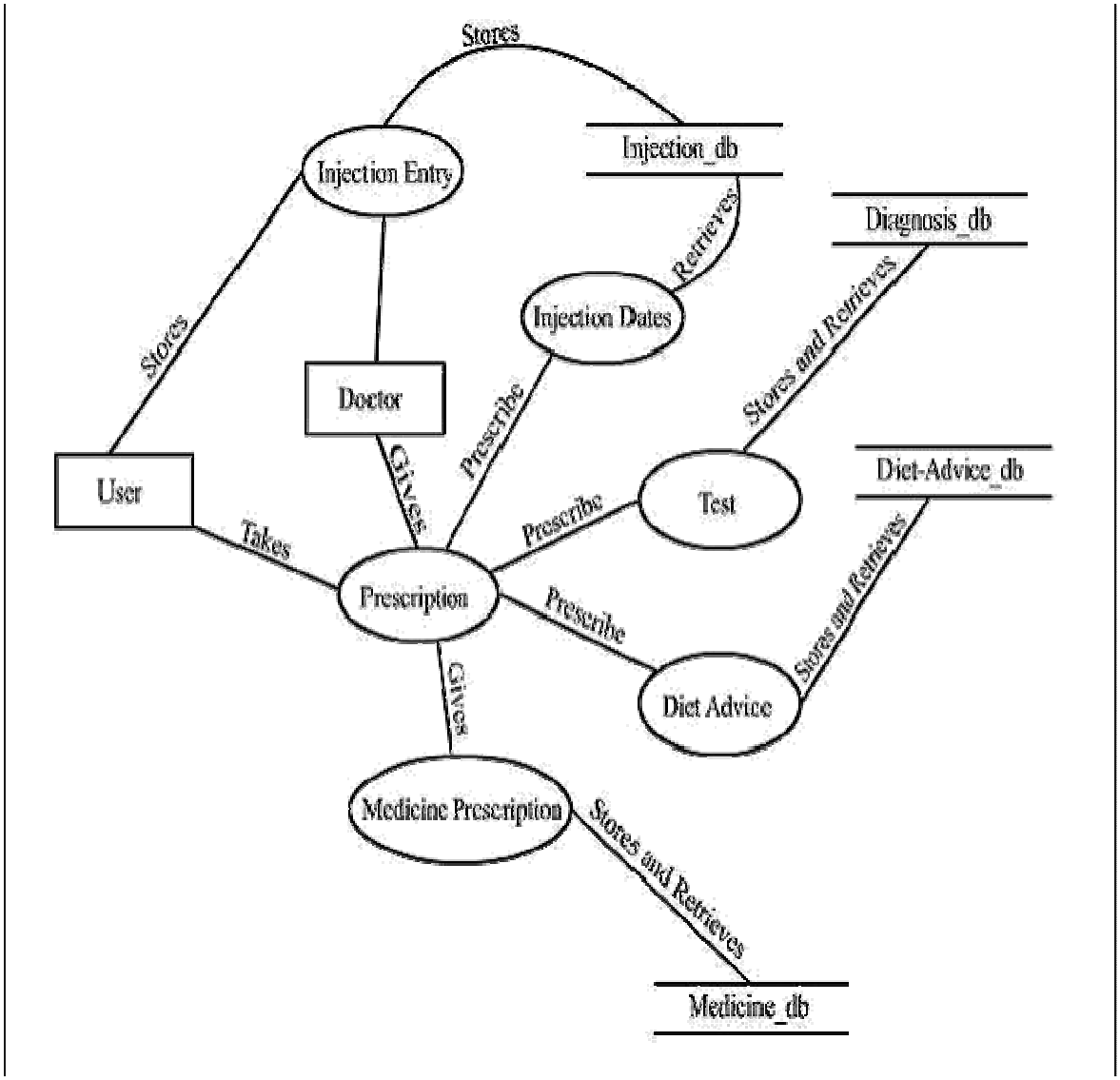


Level 1 DFD

LEVEL 2 DFD:

Next level is Level 1 DFD. Some important points are:

- Level 1 DFD must balance with the Level 0 it describes.



RULES

- In DFDs, all arrows must be labeled.
- The information flow continuity, that is all the input and the output to each refinement, must maintain the same in order to be able to produce a consistent system.

ADVANTAGES:

- DFDs have diagrams that are easy to understand, check and change data.

- DFDs help tremendously in depicting information about how an organization operations.
- They give a very clear and simple look at the organization of the interfaces between an application and the people or other applications that use it.

DISADVANTAGES:

- Modification to a data layout in DFDs may cause the entire layout to be changed. This is because the specific changed data will bring different data to units that it accesses. Therefore, evaluation of the possible of the effect of the modification must be considered first.
- The number of units in a DFD in a large application is high. Therefore, maintenance is harder, more costly and error prone. This is because the ability to access the data is passed explicitly from one component to the other. This is why changes are impractical to be made on DFDs especially in large system.

APPLICATIONS:

- DFDs are excellent guide for validating the compatibility of the process and designs of the system. This is because in order to design applications successfully, especially large ones, the design of both the processes and the data stores is important. In addition, the data must be consistent with each other. For example, there must be process to store the data in the data stores and the data stores must supply the data views accessed by the processes. Since DFDs depict the relationships between processes, data store, and data views, this made DFD the perfect guide for validating compatibility.

- DFDs are appropriate diagrams for designing high-level application architecture. This is because it is a fact that the larger the application is to be developed the more important the architecture is. For example, building a box does not need an architect but a 10-story building does. In most architectural design, they are represented as diagrams because diagrams are the best way to depict multiple relationships among multiple components. This is applicable to software design, too and DFDs helps tremendously in showing the architecture design of the system r application.

- DFDs are especially useful for depicting system flow charts. DFDs are used to show the flows of data among batch-job steps.

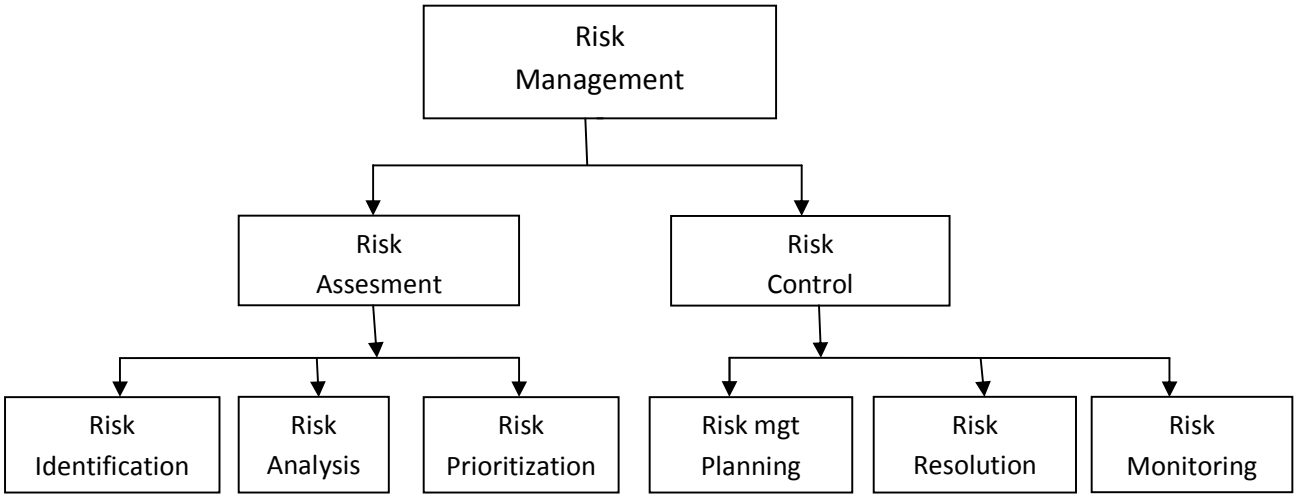
PRACTICAL NO:-

AIM: To Study the Risk Management During The Software Development.

INTRODUCTION:

Risk management is an emerging area that aims to address the problem of identifying and managing the risks associated with a software project. Risk in a project is the possibility that the defined goals are not met. The basic motivation of having risk management is to avoid disasters or heavy losses.

Risk management is the area that tries to ensure that the impact of risks on cost, quality, and schedule is minimal. It can be considered as dealing with possibility and actual occurrence of hose events that are not regular or commonly expected. So, in a sense risk management begins where normal project management ends. Diagrammatically the risk management activities are shown below which clearly shows that it revolves around **Risk Assessment** and **Risk Control**. It can be considered as dealing with the possibility and actual occurrence of those events that are not regular or commonly expected. It deals with events that are infrequent, somewhat out of the control of the project management, and are large enough.



RISK ASSESSMENT

Risk assessment is an activity that must be undertaken during project planning. This involves identifying the risks, analysing them, and prioritizing them on the basis of the analysis. The goal of risk assessment is to prioritize the risks so that risk management can focus attention and resources on the more risky items.

The risk assessment consists of three steps:

- Risk identification

- Risk analysis
- Risk prioritization

RISK IDENTIFICATION is the first step in risk assessment, which identifies all the different risks for a particular project. These risks are project dependent, and their identification is clearly necessary before any risk management can be done for the project.

Based on surveys of experienced project managers, Boehm has produced a list of the top-10 risk items likely to compromise the success of a software project. These risks and management techniques are shown above and the description is given as follows:

- The top ranked risk item is **Personnel Shortfalls**. This involves just having fewer people than necessary or not having people with specific skills that a project may require. Some of the ways to manage these risks is to get the top talent possible and to match the needs of the project with the skills of the available personnel.
- The second item, **Unrealistic Schedules And Budgets**, happens very frequently due to business and other reasons. It is very common that high level management imposes a schedule for a software project that is not based on the characteristics of the project and is unrealistic.
- Project runs the risk of **Developing The Wrong Software** if the requirements analysis is not done properly and if development begins to earlier.
- Similarly, often **Improper User Interface** may be developed. This requires extensive rework of the user interface later or the software benefits are not obtained because users are reluctant to use it.
- Some **Requirements Changes** are to be expected in any project, but some requirements frequent changes are requested, which is often a reflection of the fact that the client has not yet understood or settled on its own requirements.
- **Gold Plating** refers to adding features in the software that are only marginally useful. This adds unnecessary risk to the project because gold plating consumes recourses and time with little return.
- **Performance Shortfalls** are critical in real time systems and poor performance can mean the failure of project.
- The project might be delayed if the **External Component Is Not Available** on time. The project would also suffer if the quality of the external component is poor or if the component turns out to be incompatible with the other project components or with the environment in which the software is developed or is to operate.
- If a project relies on **Technology That Is Not Well Developed**, it may fail. This is a risk due to straining the computer science capabilities.

RISK ANALYSIS include studying the probability and the outcome of possible decisions, understanding the task dependencies to decide critical activities and the probability and cost of their not being completed on time, risks on the various quality factors like reliability and usability, and evaluating the performance early through simulation, etc , if there are strong performance constraints on the system.

One approach for **RISK PRIORITIZATION** is through the concept of risk exposure, which is sometimes called risk impact. RE is defined by the relationship

$$RE=Prob(U\ O)*Loss(U\ O),$$

Where Prob (U O) is the probability of the risk materializing and Loss (U O) is the total loss incurred due to the unsatisfactory outcome.

RISK CONTROL

Risk control includes three tasks which are

- Risk management planning
- Risk resolution
- Risk monitoring

Risk control starts with **Risk Management Planning.** Plans are developed for each identified risk that needs to be controlled. This activity, like other planning activities, is done during the initiation phase. A basic risk management plan has five components. These are

1. Why the risk is important and why it should be managed
2. What should be delivered regarding disk management and when
3. Who is responsible for performing the different risk management activities
4. How will the risk be abetted or the approach is taken.
5. And how many resources are needed.
6. The main focus of risk management planning is to enumerate the risks to be controlled and specify how to deal with a risk.

The actual elimination or reduction is done in the **Risk Resolution** step. Risk resolution is essentially implementation of the risk management plan.
Risk Monitoring is the activity of monitoring the status of various risks and their control activities. Like project monitoring, it is performed through the entire duration of the project.