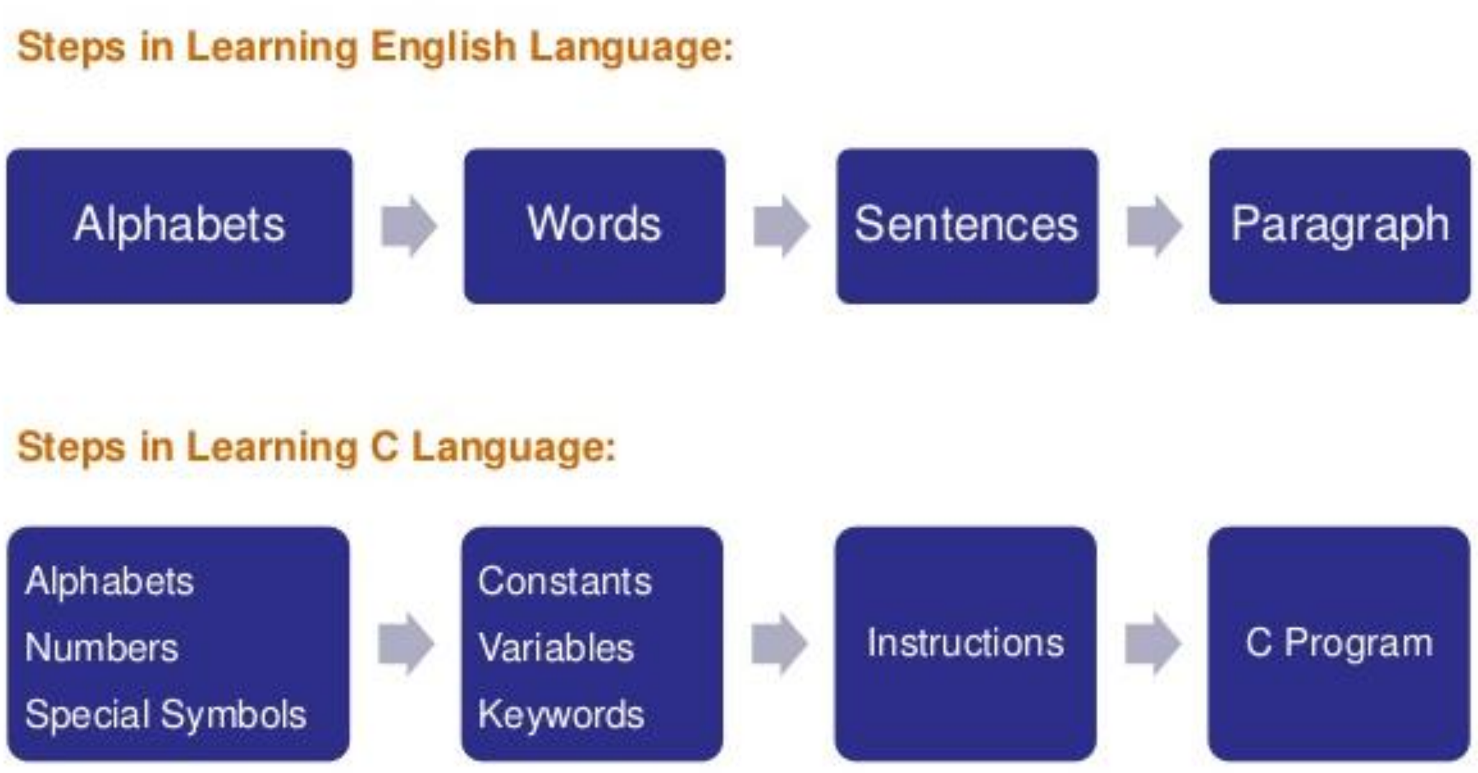
Unit 1 : Programming Language

1.1. Introduction to Programming Language

- **C** is a programming language developed at AT & T's Bell Laboratories of USA in 1972. It was designed and written by a man named Dennis Ritchie.
- In the late seventies C began to replace the more familiar languages of that time like PL/I, ALGOL, etc
- It was initially designed for programming UNIX operating system. Now the software tool as well as the C compiler is written in C. Major parts of popular operating systems like Windows, UNIX, Linux is still written in C.
- This is because even today when it comes to performance (speed of execution) nothing beats C.
- Moreover, if one is to extend the operating system to work with new devices one needs to write device driver programs. These programs are exclusively written in C.
- C seems so popular is because it is **reliable**, **simple** and **easy** to use. Often heard today is – "C has been already superceded by languages like C++, C# and Java.

1.1. Introduction to Programming Language



Steps in Learning English Language:

Alphabets → Words → Sentences → Paragraph

Steps in Learning C Language:

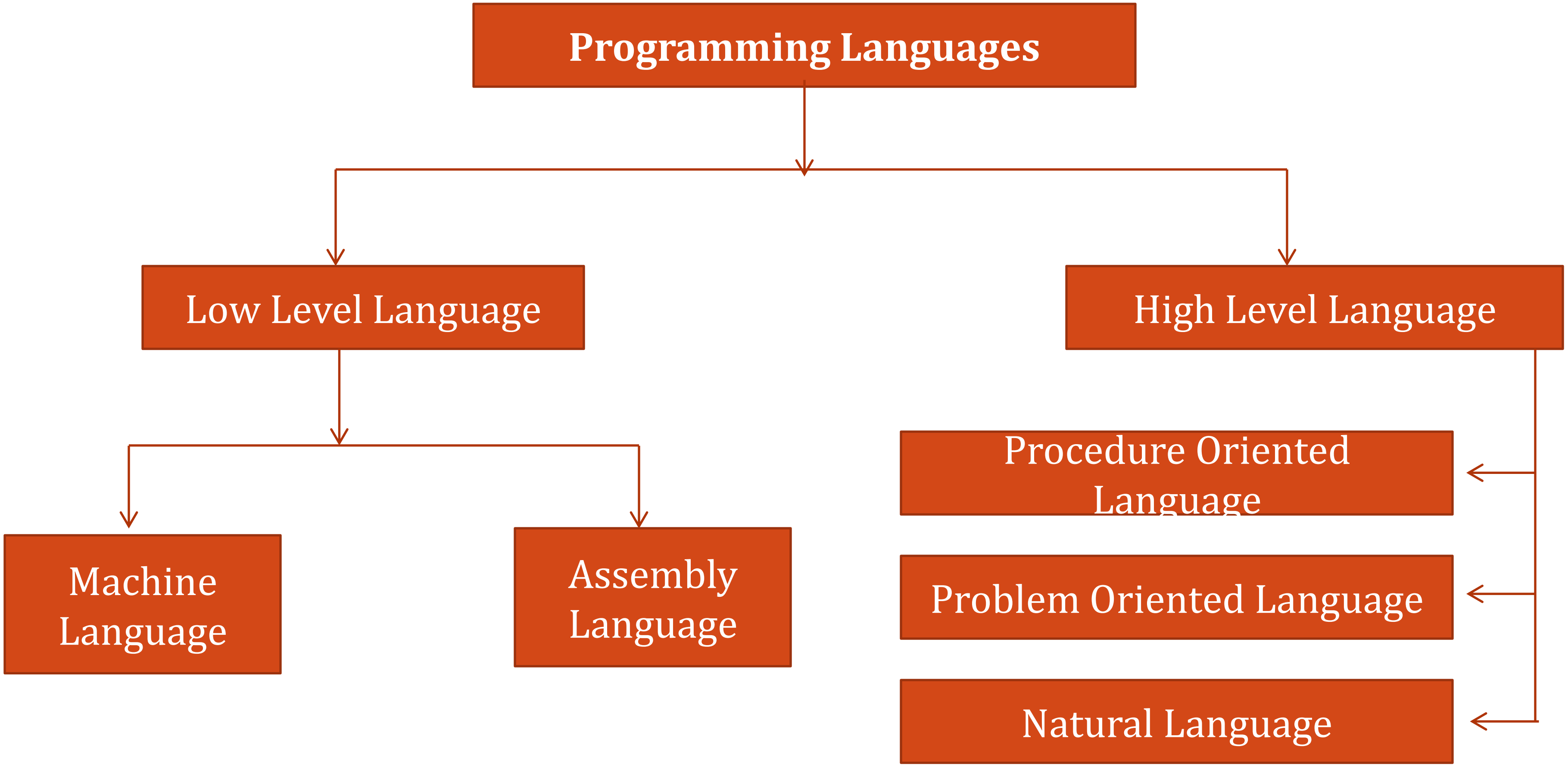
Alphabets
Numbers
Special Symbols → Constants
Variables
Keywords → Instructions → C Program

Fig. 1.1. Steps in C program

1.1. Introduction to Programming Language

- Learning C is simple and easier.
- Instead of straight-away learning how to write programs, we must first know what **alphabets**, **numbers** and **special symbols** are used in C, then how using them **constants**, **variables** and **keywords** are constructed, and finally how are these combined to form an **instruction**.
- A group of instructions would be combined later on to form a **program**.
- So a computer ***program*** is just a collection of the instructions necessary to solve a specific problem.
- The basic operations of a computer system form what is known as the computer's ***instruction set.*** And the approach or method that is used to solve the problem is known as an ***algorithm***.

1.2. Types of Programming Language



Programming Languages

Low Level Language

High Level Language

Machine Language

Assembly Language

Procedure Oriented Language

Problem Oriented Language

Natural Language

1.2. Types of Programming Language

So for as programming language concern these are of two types.

- **Low level language**
- **High level language**

1.2. Types of Programming Language

1. Low level language:

- Low level languages are **machine level** and **assembly level language**.
- In machine level language computer only understand digital numbers i.e. in the form of 0 and 1. So, instruction given to the computer is in the form binary digit, which is difficult to implement instruction in binary code.
- This type of program is not portable, difficult to maintain and also error prone. The **assembly language** is on other hand modified version of machine level language.
- Where instructions are given in English like word as ADD, SUM, MOV etc. It is easy to write and understand but not understand by the machine. So the translator used here is assembler to translate into machine level.

1.2. Types of Programming Language

2. High level language:

- These languages are machine independent, means it is portable. The language in this category is Pascal, Cobol, Fortran etc.
- High level languages are not understood by the machine. So they need to be translated by the translator into machine level.
- A translator is software which is used to translate high level language as well as low level language in to machine level language.

1.3. Language Processor

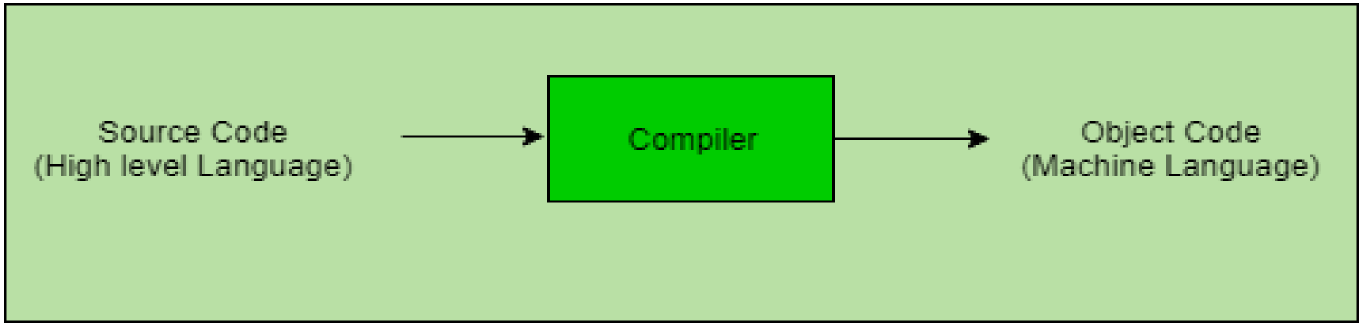
Three types of translator are there:

- **Compiler**
- **Interpreter**
- **Assembler**

1.3. Language Processor

1. Compiler

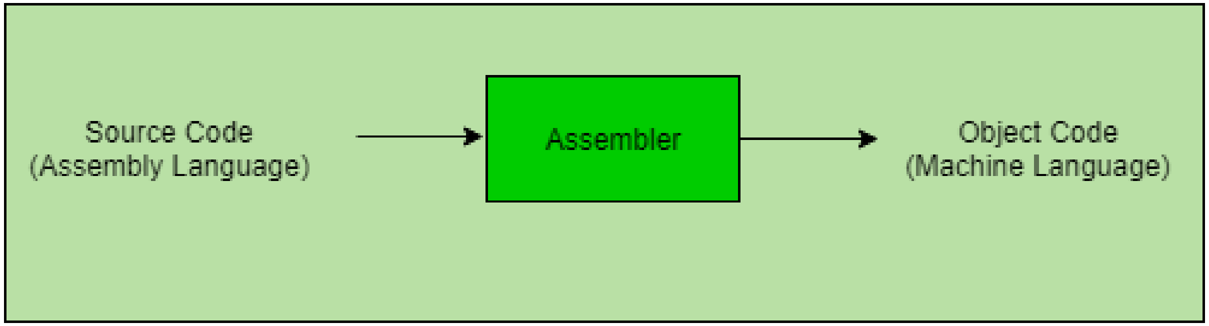
- The language processor that reads the complete source program written in high level language as a whole in one go and translates it into an equivalent program in machine language is called as a Compiler.
- **Example:** C, C++, C#, JavaIn a compiler, the source code is translated to object code successfully if it is free of errors. The compiler specifies the errors at the end of compilation with line numbers when there are any errors in the source code. The errors must be removed before the compiler can successfully recompile the source code again.



1.3. Language Processor

2. Assembler

- The Assembler is used to translate the program written in Assembly language into machine code.
- The source program is a input of assembler that contains assembly language instructions.
- The output generated by assembler is the object code or machine code understandable by the computer.



1.3. Language Processor

3. Interpreter

- The translation of single statement of source program into machine code is done by language processor and executes it immediately before moving on to the next line is called an interpreter.
- If there is an error in the statement, the interpreter terminates its translating process at that statement and displays an error message.
- The interpreter moves on to the next line for execution only after removal of the error.
- An Interpreter directly executes instructions written in a programming or scripting language without previously converting them to an object code or machine code. **Example:** Perl, Python and Matlab.

1.3. Language Processor

COMPILER	INTERPRETER
1. A compiler is a program which coverts the entire source code of a programming language into executable machine code for a CPU.	1. interpreter takes a source program and runs it line by line, translating each line as it comes to it.
2. Compiler takes large amount of time to analyze the entire source code but the overall execution time of the program is comparatively faster.	2. Interpreter takes less amount of time to analyze the source code but the overall execution time of the program is slower.
3. Compiler generates the error message only after scanning the whole program, so debugging is comparatively hard as the error can be present any where in the program.	3. Its Debugging is easier as it continues translating the program until the error is met
4. Generates intermediate object code.	4. No intermediate object code is generated.
5. Examples: C, C++, Java	5. Examples: Python, Perl

1.4. Program Errors

- Error is an illegal operation performed by the user which results in abnormal working of the program.
- Programming errors often remain undetected until the program is compiled or executed.
- Some of the errors inhibit the program from getting compiled or executed. Thus errors should be removed before compiling and executing.
- The most common errors can be broadly classified as follows.
  - Syntax Errors
  - Logical Errors
  - Runtime Errors

1.4. Program Errors

1. Syntax errors:

- Errors that occur when you **violate the rules** of writing C/C++ syntax are known as syntax errors. This compiler error indicates something that must be fixed before the code can be compiled. All these errors are detected by compiler and thus are known as compile-time errors. Most frequent syntax errors are:
  - Missing Parenthesis (})
  - Printing the value of variable without declaring it
  - Missing semicolon like this:// C program to illustrate

```
// syntax error
#include<stdio.h>
int main()
{
   int x = 10;
   int y = 15;
   printf("%d", (x, y)) // semicolon missed
    retutn 0;
} Error:
```

1.4. Program Errors

2. Run-time Errors :

- Errors which occur during program execution(run-time) after successful compilation are called run-time errors. One of the most common run-time error is division by zero also known as Division error. These types of error are hard to find as the compiler doesn’t point to the line at which the error occurs. For more understanding run the example given below.

```
#include<stdio.h>
int main()
{
    int n = 9, div = 0;
     // wrong logic
    // number is divided by 0,
    // so this program abnormally terminates
    div = n/0;
     printf("resut = %d", div);
     return 0;
}
```

1.4. Program Errors

3. Logical Errors :

- On compilation and execution of a program, desired output is not obtained when certain input values are given. These types of errors which provide incorrect output but appears to be error free are called logical errors. These are one of the most common errors done by beginners of programming. These errors solely depend on the logical thinking of the programmer and are easy to detect if we follow the line of execution and determine why the program takes that path of execution.

```
// C program to illustrate
// logical error
int main()
{
   int i = 0;
     // logical error : a semicolon after loop
   for(i = 0; i < 3; i++);
   {
      printf(“%d“, i);

   }
   getchar();
   return 0;
}
```

1.5. Features of good Program

- Every computer requires appropriate instruction set (programs) to perform the required task.
- The quality of the processing depends upon the given instructions.
- If the instructions are improper or incorrect, then it is obvious that the result will be superfluous.
- Therefore, proper and correct instructions should be provided to the computer so that it can provide the desired output.
- Hence, a program should be developed in such a way that it ensures proper functionality of the computer. In addition, a program should be written in such a manner that it is easier to understand the underlying logic.

1.5. Features of good Program

- A good computer program should have following characteristics:

1. **Portability**
2. **Readability**
3. **Efficiency**
4. **Structural**
5. **Flexibility**
6. **Generality**
7. **Documentation**

1.5. Features of good Program

1. Portability:

- Portability refers to the ability of an application to run on different platforms (operating systems) with or without minimal changes.
- Due to rapid development in the hardware and the software, nowadays platform change is a common phenomenon.
- Hence, if a program is developed for a particular platform, then the life span of the program is severely affected.

1.5. Features of good Program

2. Readability:

- The program should be written in such a way that it makes other programmers or users to follow the logic of the program without much effort.
- If a program is written structurally, it helps the programmers to understand their own program in a better way.
- Even if some computational efficiency needs to be sacrificed for better readability, it is advisable to use a more user-friendly approach, unless the processing of an application is of utmost importance.

1.5. Features of good Program

3. Efficiency:

- Every program requires certain processing time and memory to process the instructions and data.
- As the processing power and memory are the most precious resources of a computer, a program should be laid out in such a manner that it utilizes the least amount of memory and processing time.

1.5. Features of good Program

4. Structural:

- To develop a program, the task must be broken down into a number of subtasks.
- These subtasks are developed independently, and each subtask is able to perform the assigned job without the help of any other subtask.
- If a program is developed structurally, it becomes more readable, and the testing and documentation process also gets easier.

1.5. Features of good Program

5. Flexibility:

- A program should be flexible enough to handle most of the changes without having to rewrite the entire program.
- Most of the programs are developed for a certain period and they require modifications from time to time.
- For example, in case of payroll management, as the time progresses, some employees may leave the company while some others may join.
- Hence, the payroll application should be flexible enough to incorporate all the changes without having to reconstruct the entire application.

1.5. Features of good Program

6. Generality:

- Apart from flexibility, the program should also be general. Generality means that if a program is developed for a particular task, then it should also be used for all similar tasks of the same domain.
- For example, if a program is developed for a particular organization, then it should suit all the other similar organizations.

1.5. Features of good Program

7. Documentation:

- Documentation is one of the most important components of an application development.
- Even if a program is developed following the best programming practices, it will be rendered useless if the end user is not able to fully utilize the functionality of the application.
- A well-documented application is also useful for other programmers because even in the absence of the author, they can understand it.

1.6. Program Paradigm

- **Paradigm** It is a method to solve some problem or do some task.
- Programming paradigm is an approach to solve problem using some programming language or also we can say it is a method to solve a problem using tools and techniques that are available to us following some approach.
- Programming paradigms are not languages or tools. You can't "build" anything with a paradigm. They're more like a set of ideals and guidelines that many people have agreed on, followed, and expanded upon.

1.6. Program Paradigm (Programming Language Paradigm

- Imperative or Procedural Language
  (C, Fortan, Pascal)
- Applicative or Functional Language (LISP)
- Object Oriented Language (C++, Java, C#)

1.6. Program Paradigm

1. **Imperative or Procedural Language**: Imperative or procedural languages are statement-oriented language. A program consists of a sequence of statement and the execution of each statement cause the computer to change the value of one or more location in its memory.

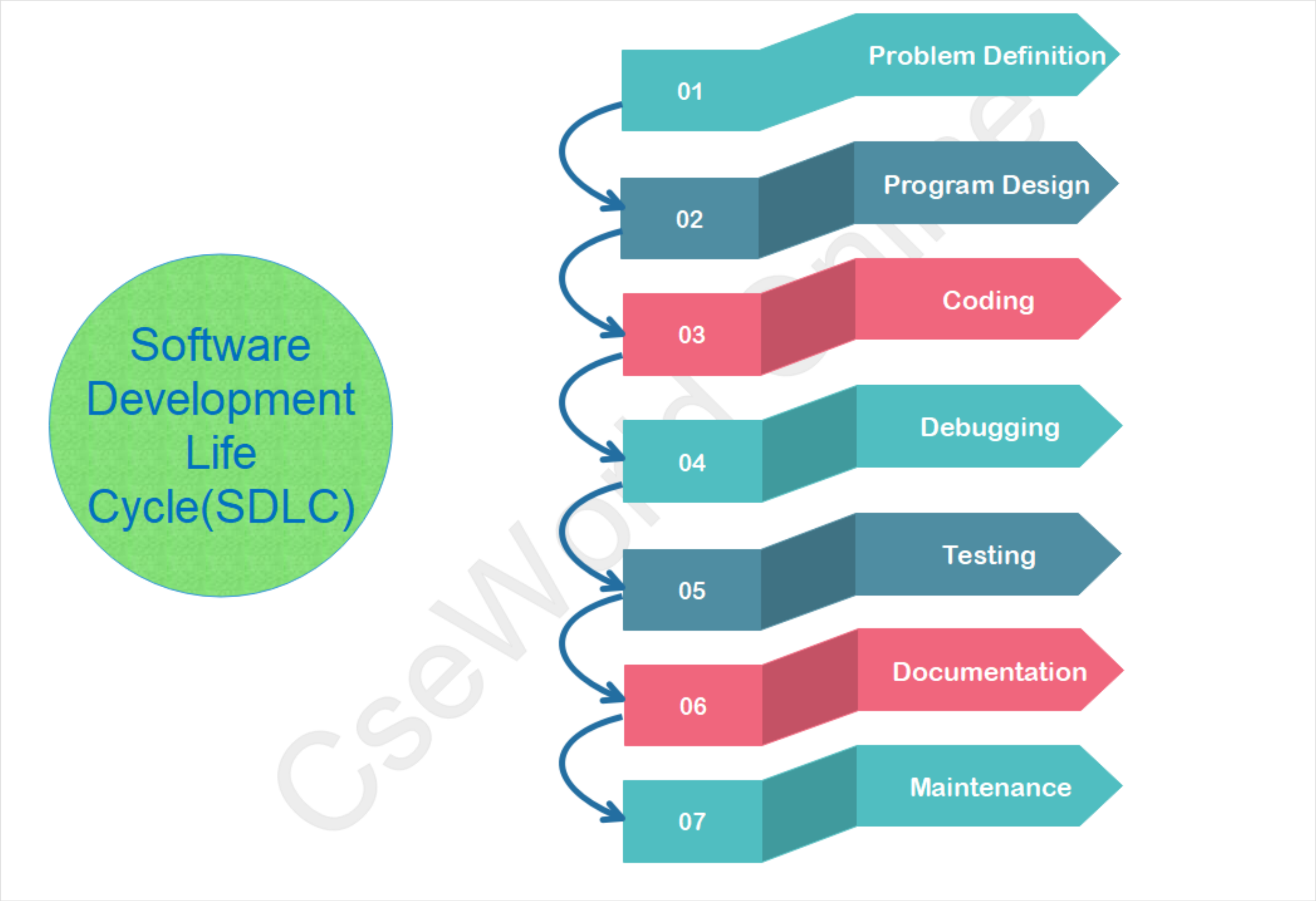
2. **Applicative or Functional Language:** Applicative or functional programming language is designed to support the development of program by the use of functions.

3. **Object Oriented Language:** It is based on the concept of object which contains data in the form of attributes and code in the form of method. Object Oriented Programming allows decomposition of a problem into a number of entities called objects and then build data and function around these objects.

1.7. Program Development life cycle

- When we want to develop a program using any program using any programming language, we follow a sequence of steps.
- These steps are called phases in program development.
- The various stages in the development of a computer program are :
  1. Problem Definition
  2. Program Design
  3. Coding
  4. Debugging
  5. Testing
  6. Documentation
  7. Maintenance

1.7. Program Development life cycle



1.7. Program Development life cycle

1. Problem Definition:

- The first step in the process of program development is the thorough understanding and identification of the problem for which is the program or software is to be developed.
- In this step the problem has to be defined formally.
- All the factors like Input/output, processing requirement, memory requirements, error handling, interfacing with other programs have to be taken into consideration in this stage.

1.7. Program Development life cycle

2. Program Design:

- The next stage is the program design. The software developer makes use of tools like algorithms and flowcharts to develop the design of the program.
  - Algorithm
  - Flowchart

1.7. Program Development life cycle

2. Program Design:

1. **Definition of Algorithm**

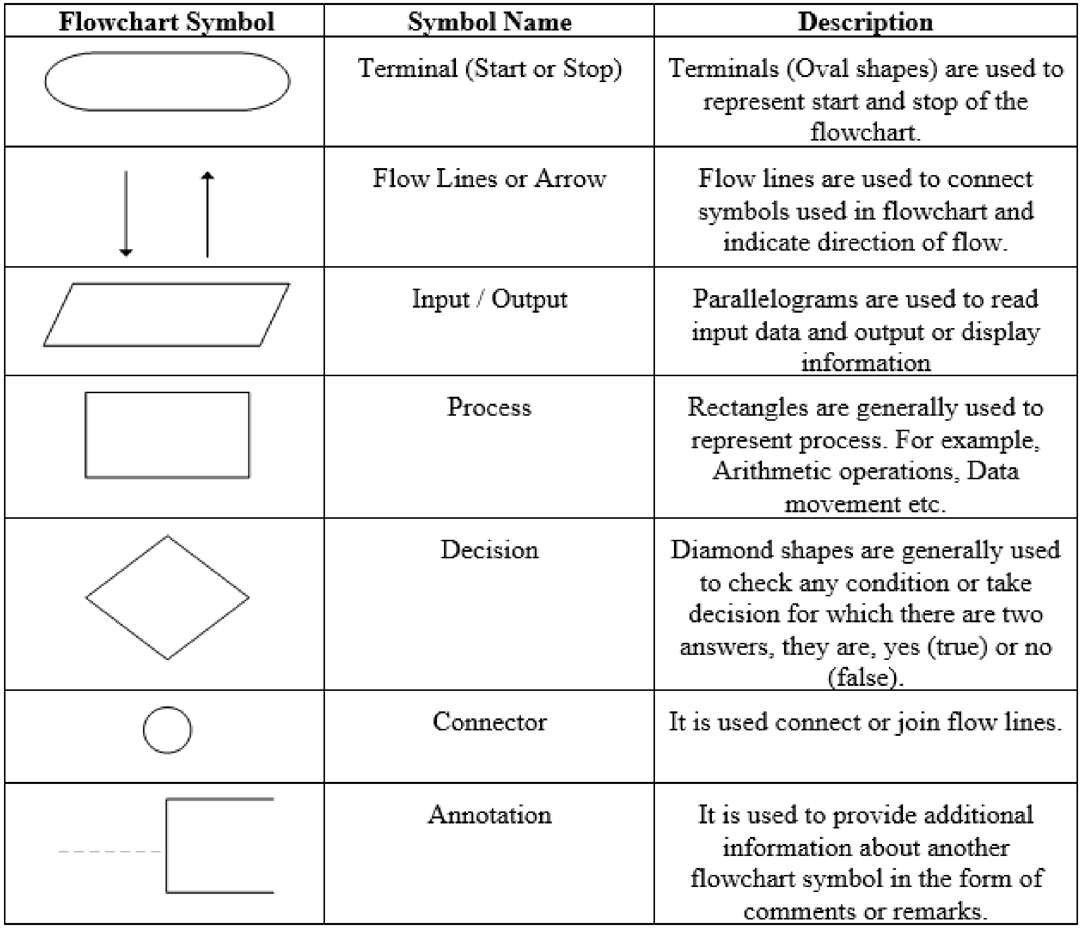
- To write a logical step-by-step method to solve the problem is called the algorithm; in other words, an algorithm is a procedure for solving problems. In order to solve a mathematical or computer problem, this is the first step in the process. An algorithm includes calculations, reasoning, and data processing. Algorithms can be presented by natural languages, pseudocode, and flowcharts, etc.

2. **Definition of Flowchart**

- A flowchart is the graphical or pictorial representation of an algorithm with the help of different symbols, shapes, and arrows to demonstrate a process or a program. With algorithms, we can easily understand a program. The main purpose of using a flowchart is to analyze different methods. Several standard symbols are applied in a flowchart:

1.7. Program Development life cycle

Definition of Flowchart



Flowchart Symbol	Symbol Name	Description
	Terminal (Start or Stop)	Terminals (Oval shapes) are used to represent start and stop of the flowchart.
	Flow Lines or Arrow	Flow lines are used to connect symbols used in flowchart and indicate direction of flow.
	Input / Output	Parallelograms are used to read input data and output or display information
	Process	Rectangles are generally used to represent process. For example, Arithmetic operations, Data movement etc.
	Decision	Diamond shapes are generally used to check any condition or take decision for which there are two answers, they are, yes (true) or no (false).
	Connector	It is used connect or join flow lines.
	Annotation	It is used to provide additional information about another flowchart symbol in the form of comments or remarks.

Start and stop

Flow or connection

Input or output

process

condition

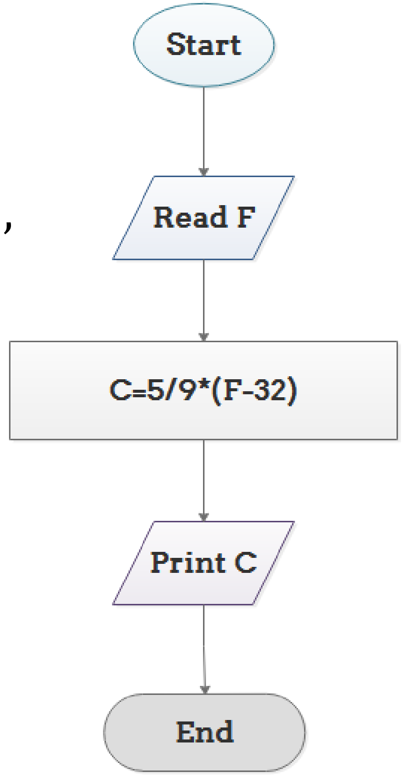
1.7. Program Development life cycle

Example 1: Convert Temperature from Fahrenheit (^{0}F) to Celsius (^{0}C)

Flowchart:

Algorithm:

- Step 1: Start
- Step 2: Read temperature in Fahrenheit,
- Step 3: Calculate temperature with formula C=5/9*(F-32),
- Step 4: Print C
- Step 5 : End



1.7. Program Development life cycle

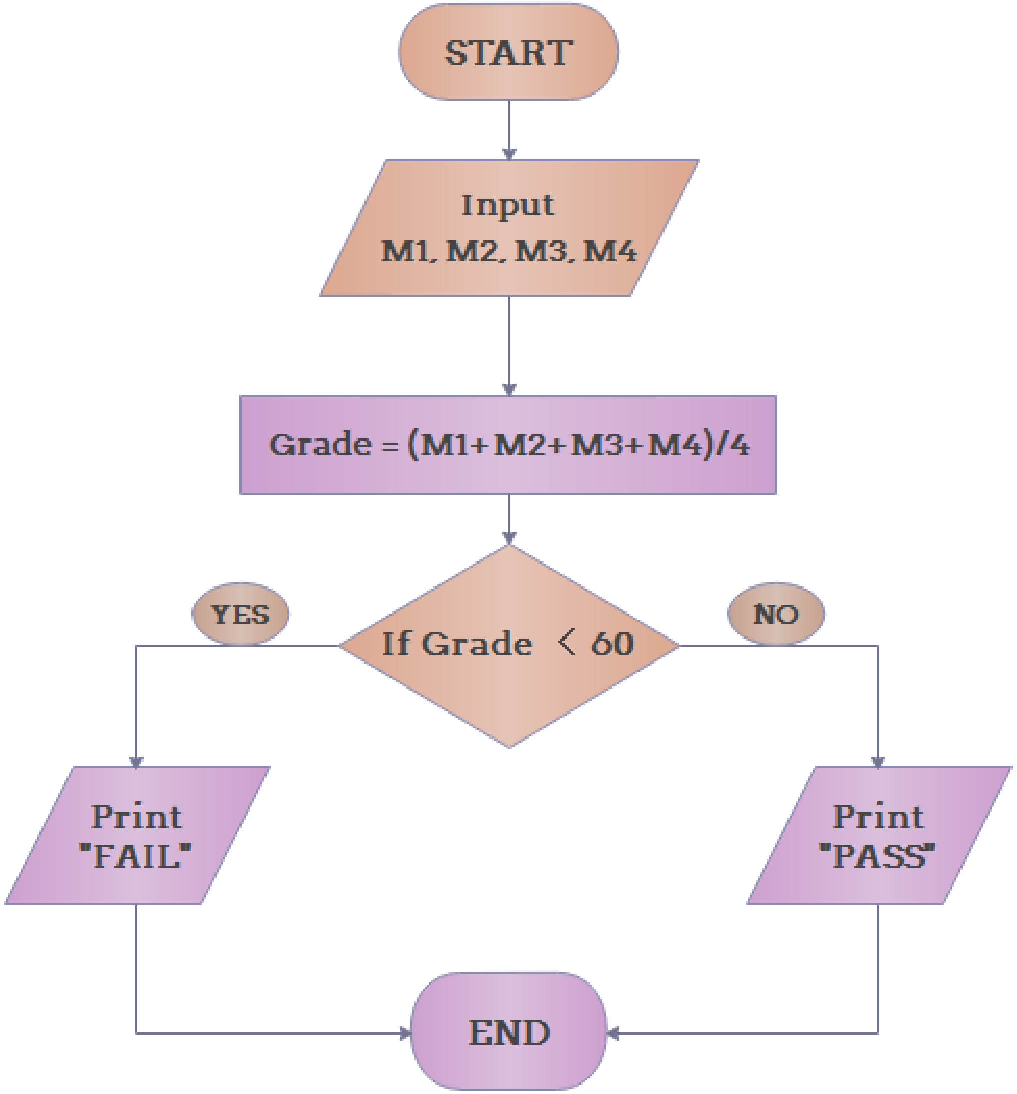
Example 2: Determine Whether A Student Passed the Exam or Not:

Algorithm:

- Step 1: Start
- Step 2: Input grades of 4 courses M1, M2, M3 and M4,
- Step 3: Calculate the average grade with formula "Grade=(M1+M2+M3+M4)/4"
- Step 4: If the average grade is less than 60, print "FAIL", else print "PASS".
- Step 5: End

Con....

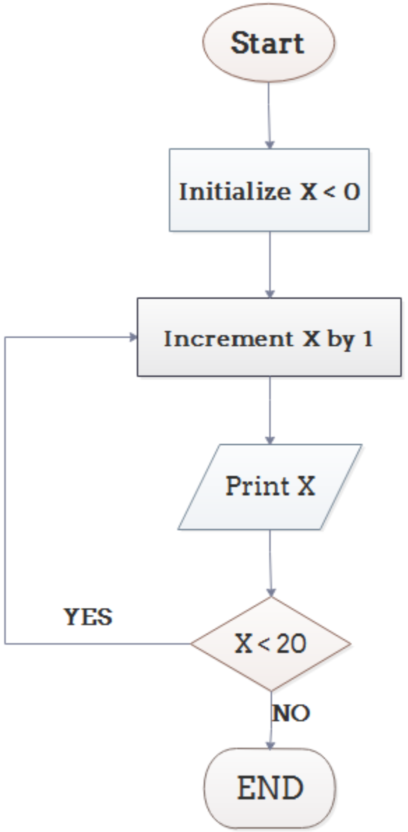
Flowchart:



1.7. Program Development life cycle

Example 3: Print 1 to 20

Flowchart:



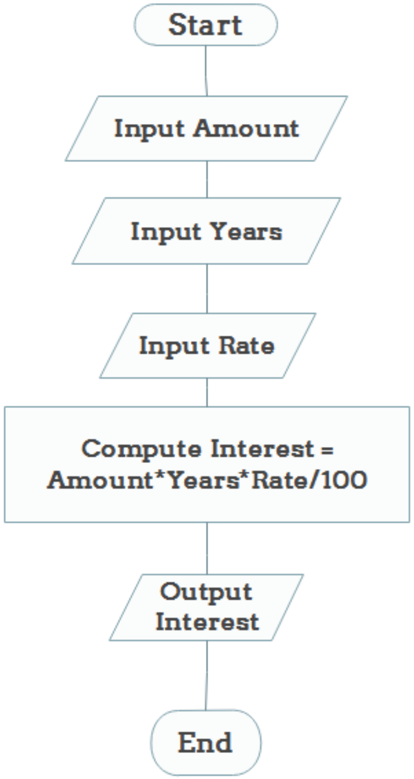
Algorithm:

- Step 1: Start
- Step 2: Initialize X as 0,
- Step 3: Increment X by 1,
- Step 4: Print X,
- Step 5: If X is less than 20 then go back to step 2.
- Step 6: End

1.7. Program Development life cycle

Example 4: Calculate the Interest of a Bank Deposit

Flowchart:



Algorithm:

- Start 1: Start
- Step 2: Read amount,
- Step 3: Read years,
- Step 4: Read rate,
- Step 5: Calculate the interest with the formula "Interest=Amount*Years*Rate/100
- Step 6: Print interest,
- Step 7: End

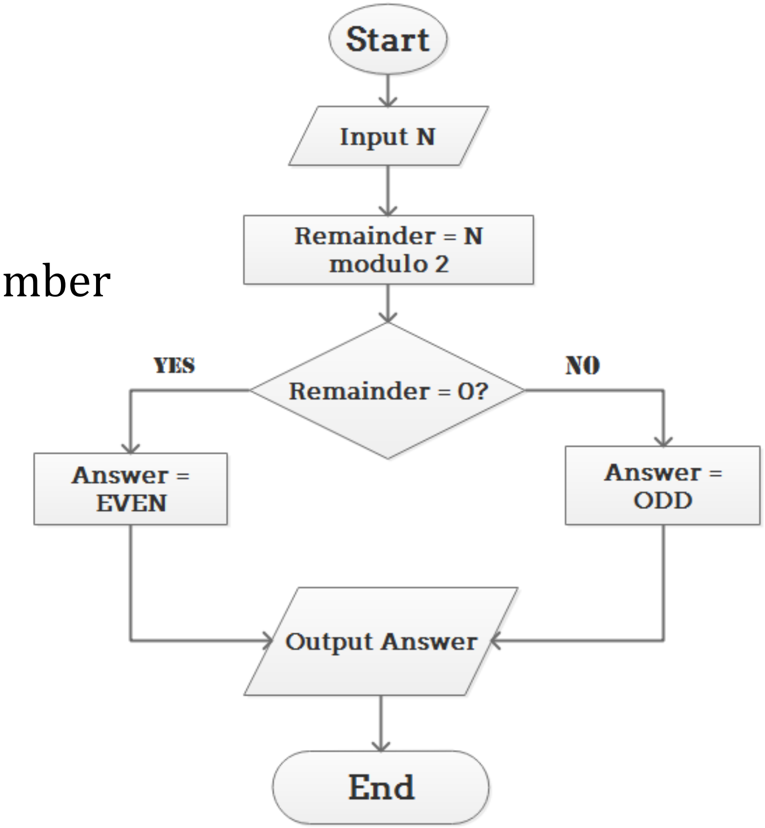
1.7. Program Development life cycle

Example 5: Determine and Output Whether Number N is Even or Odd

Algorithm:

- Step 1: Start
- Step 2: Read number N,
- Step 3: Set remainder as N modulo 2,
- Step 4: If the remainder is equal to 0 then number N is even, else number N is odd,
- Step 5: Print output.
- Step 6: End

Flowchart:



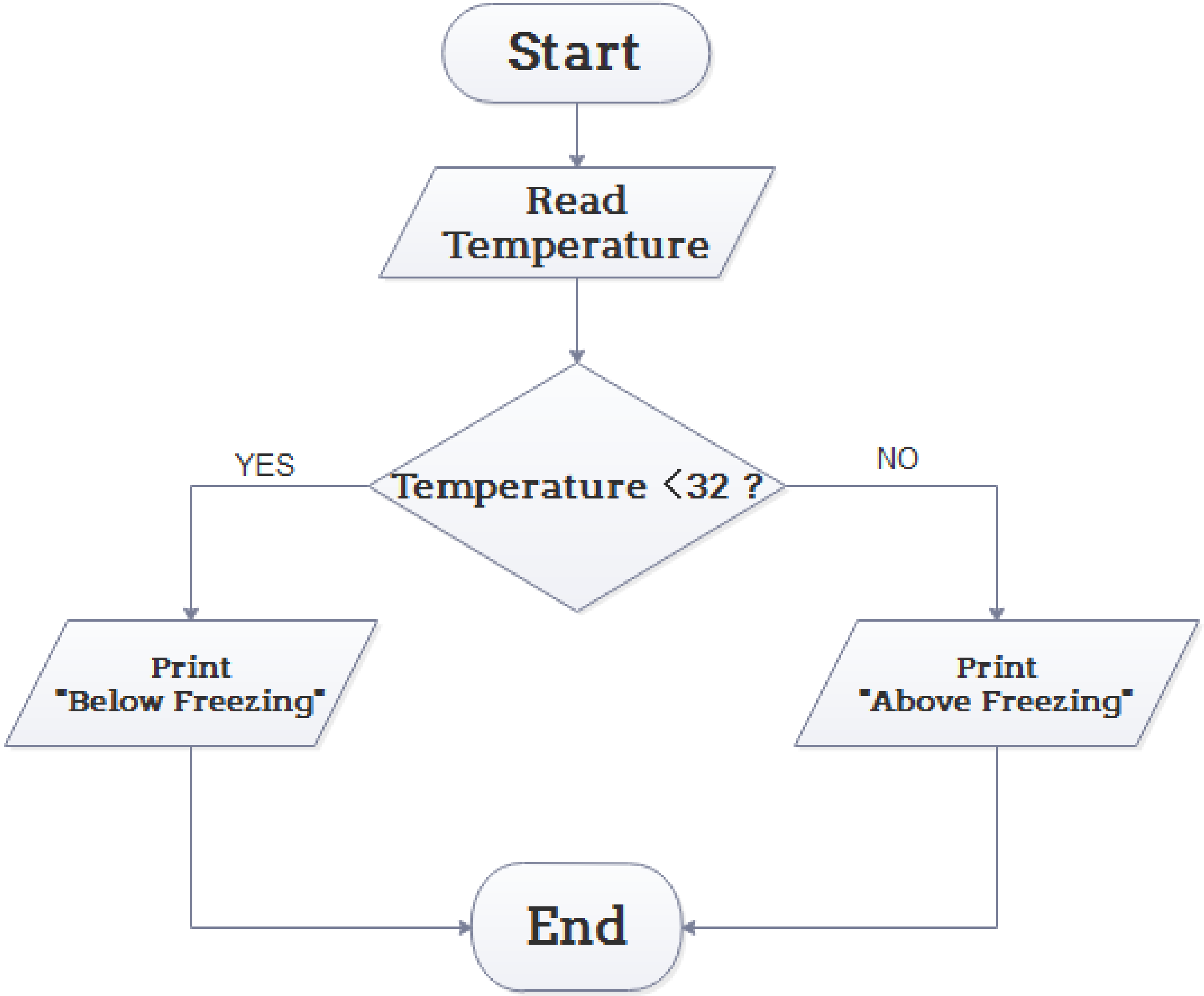
1.7. Program Development life cycle

Example 6: Determine Whether a Temperature is Below or Above the Freezing Point

Algorithm:

- Step 1:Start
- Step 2: Input temperature,
- Step 3: If it is less than 32, then print "below freezing point", otherwise print "above freezing point".
- Step 4: End

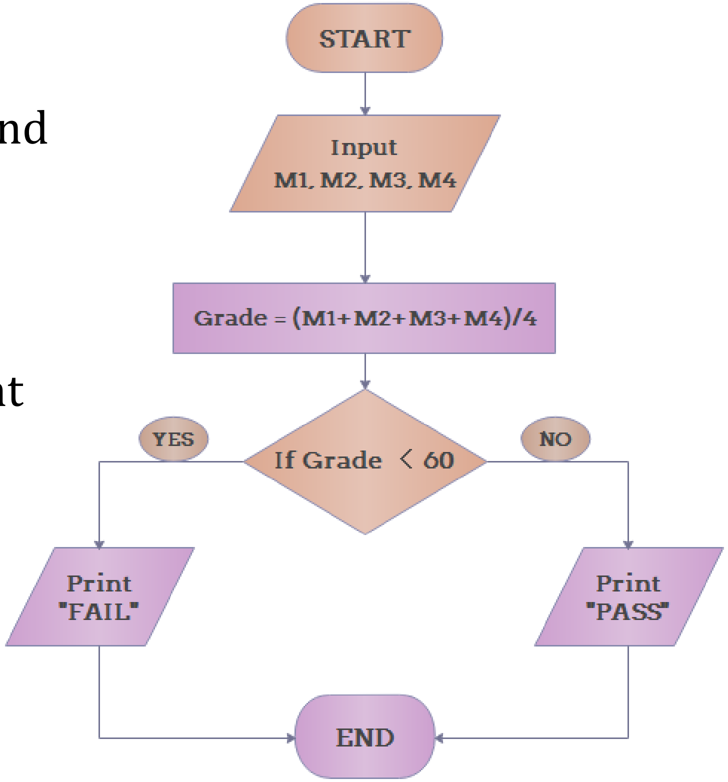
Flowchart:


Start
Read Temperature
Temperature <32 ?
YES
NO
Print "Below Freezing"
Print "Above Freezing"
End

1.7. Program Development life cycle

Example 7: Determine Whether A Student Passed the Exam or Not:

Flowchart:



Algorithm:

- Step 1:Start
- Step 2: Input grades of 4 courses M1, M2, M3 and M4,
- Step 3: Calculate the average grade with the formula "Grade=(M1+M2+M3+M4)/4"
- Step 4: If the average grade is less than 60, print "FAIL", else print "PASS".
- Step 5: End

1.7. Program Development life cycle

3. Coding:

- Once the design process is complete, the actual computer program is written, i.e. the instructions are written in a computer language.
- Coding is generally a very small part of the entire program development process and also a less time consuming activity in reality.
- In this process all the syntax errors i.e. errors related to spelling, missing commas, undefined labels etc. are eliminated.
- For effective coding some of the guide lines which are applied are :
  - Use of meaningful names and labels of variables,
  - Simple and clear expressions,
  - Modularity with emphasis on making modules generalized,
  - Making use of comments and indenting the code properly,
  - Avoiding jumps in the program to transfer control.

1.7. Program Development life cycle

4. Debugging:

- At this stage the errors in the programs are detected and corrected.
- This stage of program development is an important process. Debugging is also known as program validation.
- Some common errors which might occur in the programs include:
  - Un initialization of variables.
  - Reversing of order of operands.
  - Confusion of numbers and characters.

1.7. Program Development life cycle

5. Testing:

- The program is tested on a number of suitable test cases.
- A test plan of the program has to be done at the stage of the program design itself.
- This ensures a thorough understanding of the specifications.
- The most trivial and the most special cases should be identified and tested.
- It is always useful to include the maximum and minimum values of all variables as test data.

1.7. Program Development life cycle

6. Documentation:

- Documentation is a very essential step in the program development.
- Documentation help the users and the people who maintain the software.
- This ensures that future modification if required can be done easily. Also it is required during redesigning and maintenance.

1.7. Program Development life cycle

7. Maintenance:

- Updating and correction of the program for changed conditions and field experience is accounted for in maintenance.
- Maintenance becomes essential in following situations:
  - Change in specification,
  - Change in equipment,
  - Errors which are found during the actual execution of the program.

1.8. Software development Model

- Software development life cycle (**SDLC**) is a series of phases that provide a common understanding of the software building process.
- The good software engineer should have enough knowledge on how to choose the SDLC model based on the project context and the business requirements.
- Therefore, it may be required to choose the right SDLC model according to the specific concerns and requirements of the project to ensure its success.

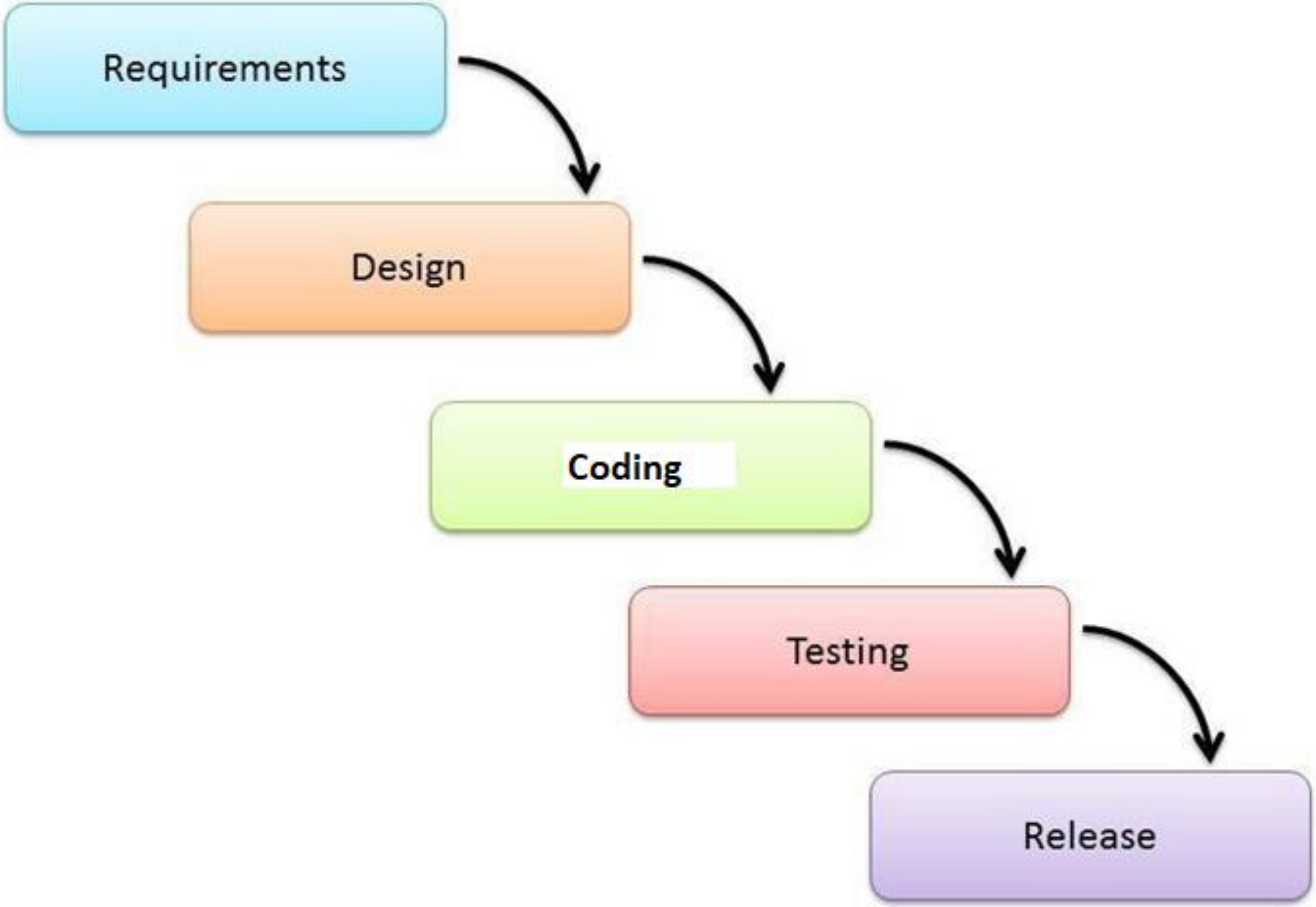
1.8. Software development Model

1.Waterfall Model

- Waterfall model is the earliest SDLC approach that was used for software development.
- Waterfall model is an example of a sequential model.
- It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin.
- The waterfall approach does not define the process to go back to the previous phase to handle changes in requirement.

1.8. Software development Model

1.Waterfall Model



1.8. Software development Model

1.Waterfall Model

Advantages and Disadvantages of waterfall model

Advantages

- Simple and easy to understand and use.
- Easy to manage.
- Works well for smaller and low budget projects where requirements are very well understood.
- Clearly defines stages and well understood.
- Process and results are well documented.
- **Disadvantages**
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex projects.
- Poor model for long and ongoing projects.
- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.

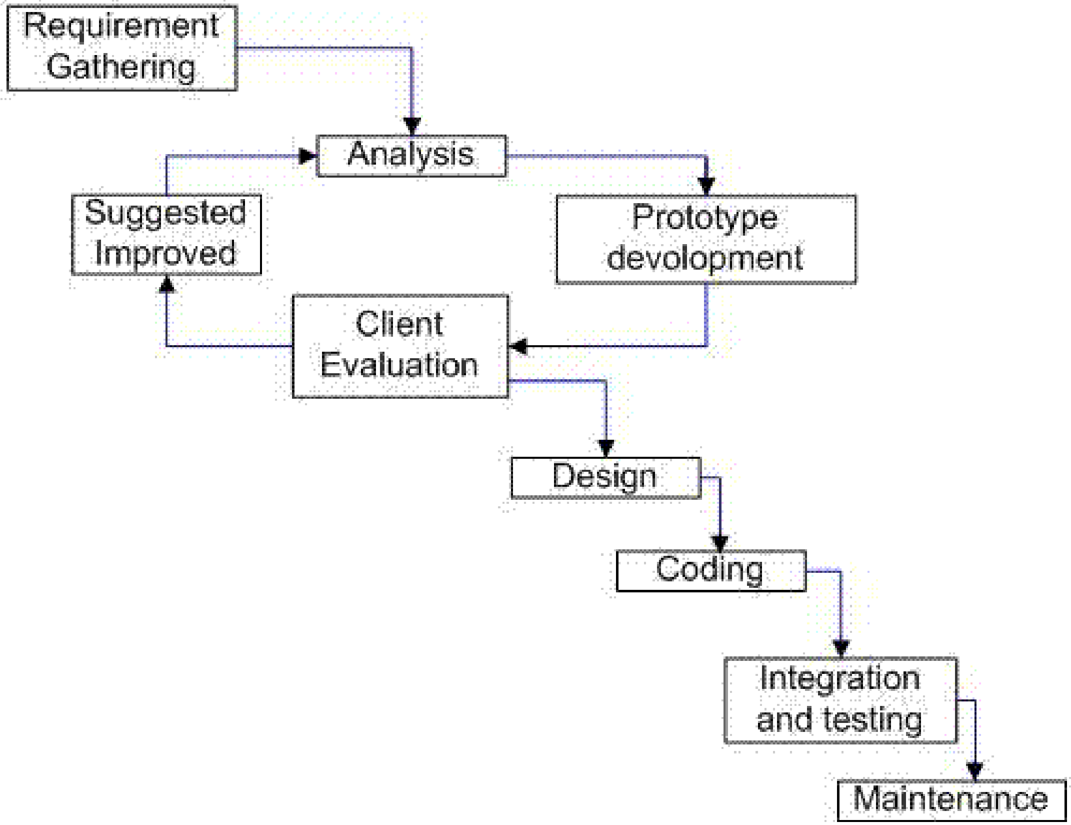
1.8. Software development Model

2.Prototyping Model

- Prototype is a working model of software with some limited functionality.
- In Prototyping Model, we do not develop the full software, firstly we develop a prototype of the software including some favorable functions in the product.
- After this, the software is handed over to the customer and the customer will use the prototype product and check any possible problems.
- If any problem is identified or if the customer is not satisfied then we remove the problems in the product and we create a new version of prototype of the product.
- Prototyping model should be used when the desired system needs to have a lot of interaction with the end users.

1.8. Software development Model

2.Prototyping Model



1.8. Software development Model

2.Prototyping Model

Advantages

- Users are actively involved in the development.
- More accurate user requirements are obtained.
- Errors can be detected much earlier.
- Quick customer feedback provides a better idea of customer needs.
- Missing functionality can be identified easily.

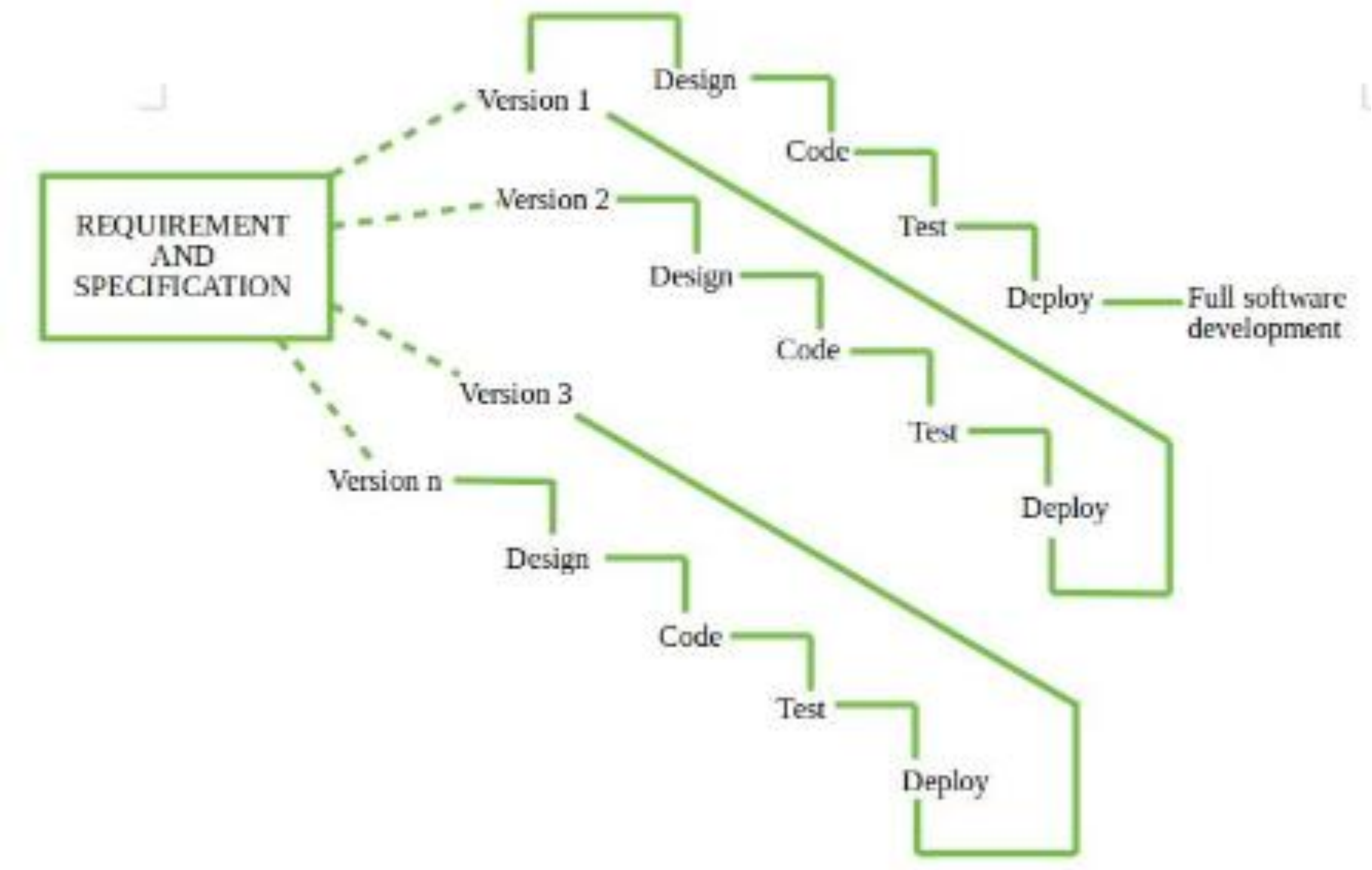
Disadvantages

- Time consuming and expensive as several new prototypes of the product should be developed if customer is not satisfied.
- Customer may demand the actual system to be delivered soon after seeing an early prototype.
- If customer is not satisfied in the initial prototype, he/she may lose interest in the project.

1.8. Software development Model

- **3. Incremental Model**
- Also known as the Successive version model.
- First, a simple working system implementing only a few basic features is built and then that is delivered to the customer. Then thereafter many successive iterations/ versions are implemented and delivered to the customer until the desired system is released.

- Incremental Model



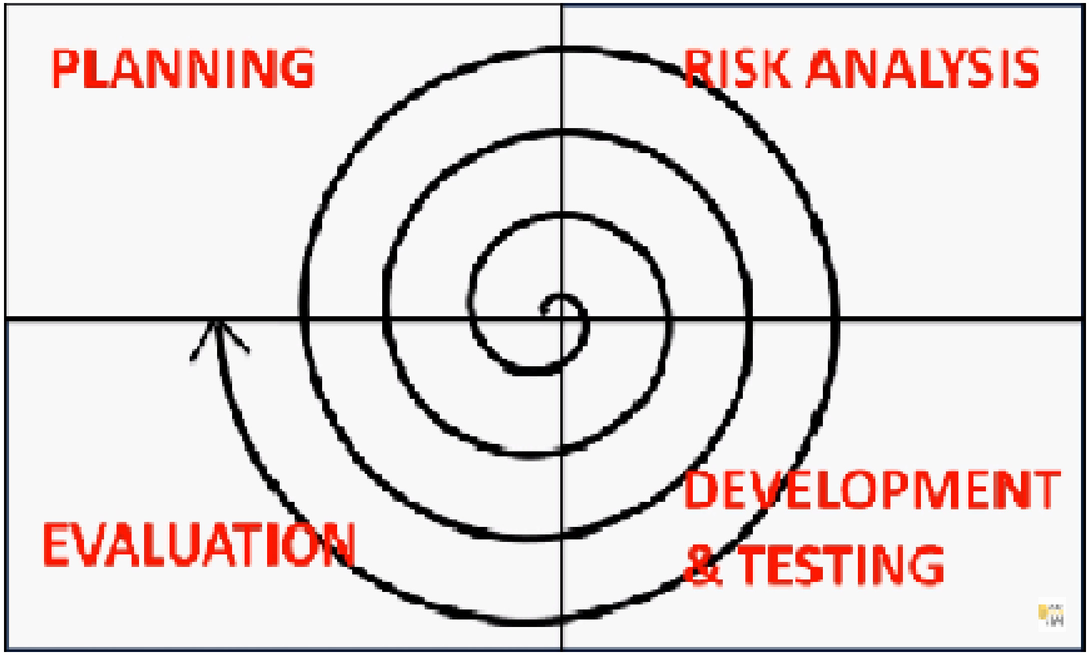
- **Advantages Of Incremental Model**
  - Prepares the software fast.
  - Clients have a clear idea of the project.
  - Changes are easy to implement.
  - Provides risk handling support, because of its iterations.

Disadvantages Of Incremental Model

- Because of its continuous iterations the cost increases.
- Issues may arise from the system design if all needs are not gathered upfront throughout the duration of the program lifecycle.
- Every iteration step is distinct and does not flow into the next.
- It takes a lot of time and effort to fix an issue in one unit if it needs to be corrected in all the units.

1.8. Software development Model

4.Spiral Model



1.8. Software development Model

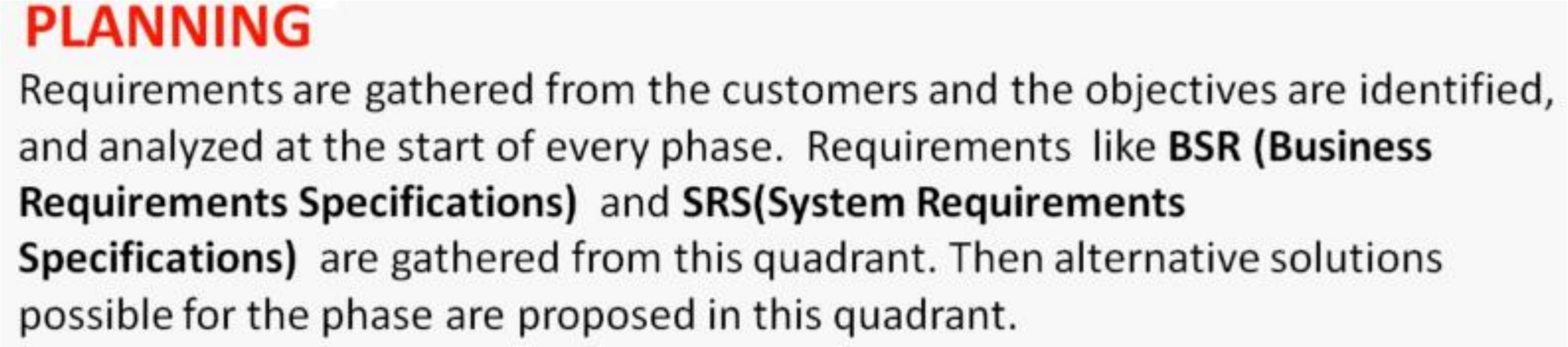
3.Spiral Model

- The spiral model is a systems development lifecycle (SDLC) method used for risk management that combines the iterative development process model with elements of the Waterfall model. The spiral model is used by software engineers and is favored for large, expensive and complicated projects.
- The spiral model has 4 phases:

  Planning, Risk Analysis, Development & Testing, Evaluation
- In its diagrammatic representation, it looks like a spiral.

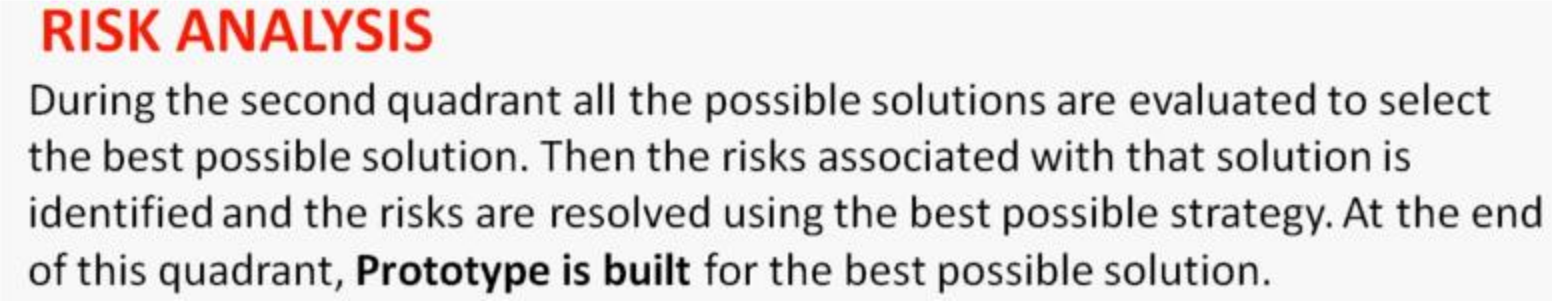
1.8. Software development Model

- Spiral Model



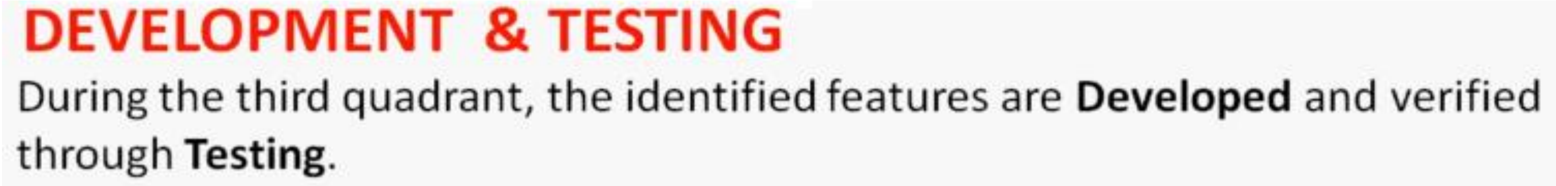
PLANNING

Requirements are gathered from the customers and the objectives are identified, and analyzed at the start of every phase. Requirements like **BSR (Business Requirements Specifications)** and **SRS(System Requirements Specifications)** are gathered from this quadrant. Then alternative solutions possible for the phase are proposed in this quadrant.



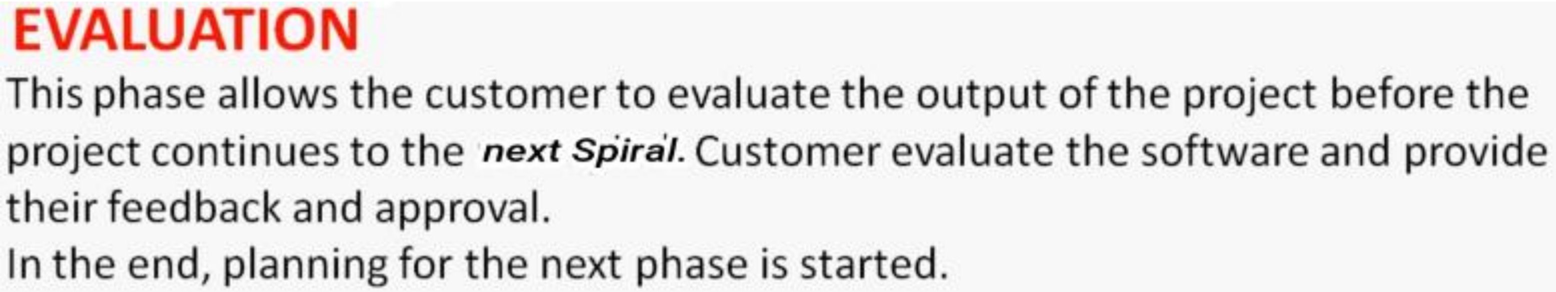
RISK ANALYSIS

During the second quadrant all the possible solutions are evaluated to select the best possible solution. Then the risks associated with that solution is identified and the risks are resolved using the best possible strategy. At the end of this quadrant, **Prototype is built** for the best possible solution.



DEVELOPMENT & TESTING

During the third quadrant, the identified features are **Developed** and verified through **Testing**.



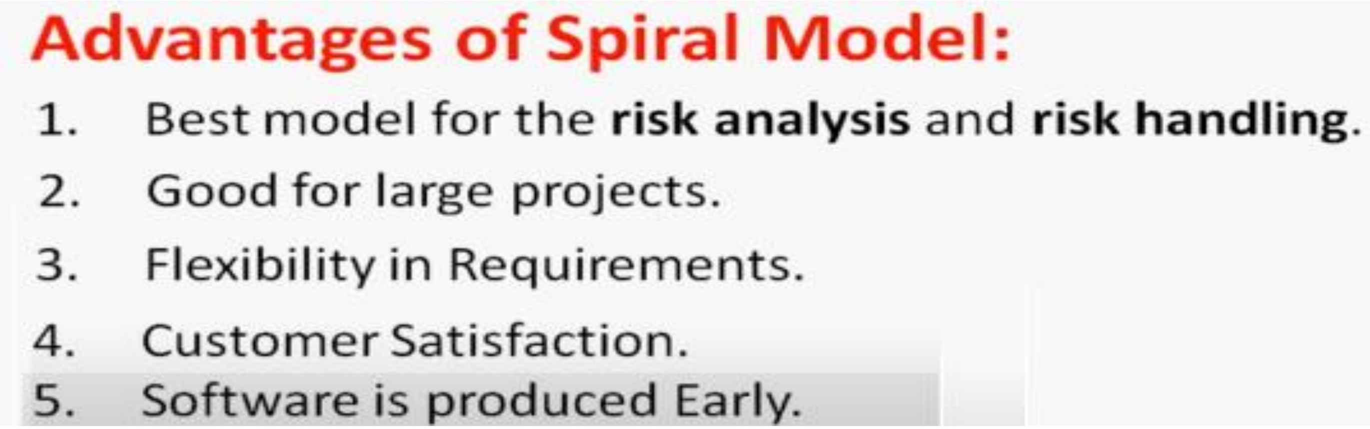
EVALUATION

This phase allows the customer to evaluate the output of the project before the project continues to the ***next Spiral.*** Customer evaluate the software and provide their feedback and approval.

In the end, planning for the next phase is started.

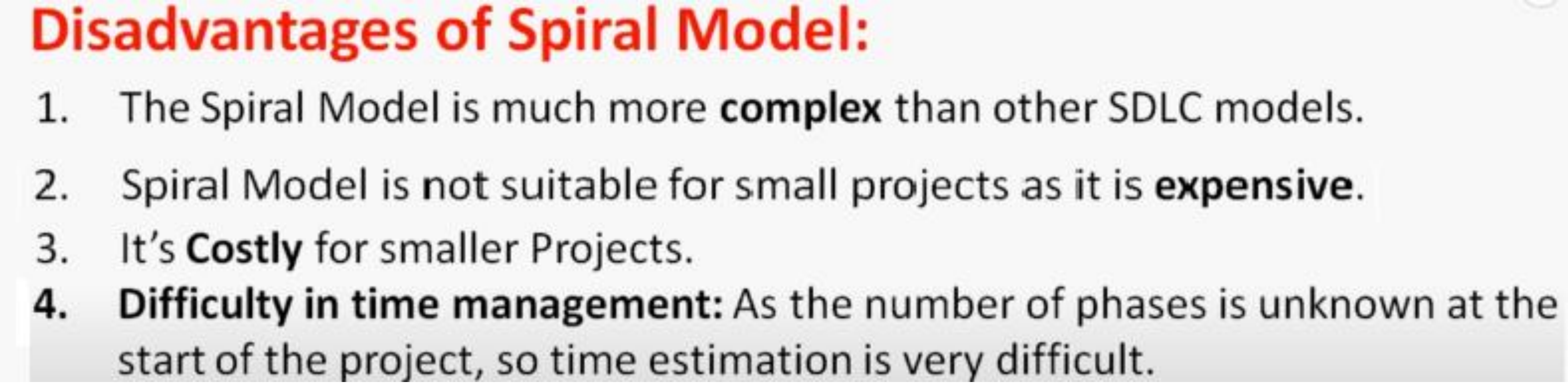
1.8. Software development Model

Spiral Model



Advantages of Spiral Model:

1. Best model for the **risk analysis** and **risk handling**.
2. Good for large projects.
3. Flexibility in Requirements.
4. Customer Satisfaction.
5. Software is produced Early.



Disadvantages of Spiral Model:

1. The Spiral Model is much more **complex** than other SDLC models.
2. Spiral Model is not suitable for small projects as it is **expensive**.
3. It's **Costly** for smaller Projects.
4. **Difficulty in time management:** As the number of phases is unknown at the start of the project, so time estimation is very difficult.

1.9. System Design Tools

- System analysis can be complex and confusing work.
- The analyst should be able to deal with a large amount of highly detailed and often conflicting information.
- The analyst needs a way to organize the information, determine where there are gaps in understanding and identify areas of conflicting.
- Modeling techniques provide solutions for the system analyst.
- The modeling techniques used in system development are:
  - Context diagram
  - Data Flow Diagram (DFD)
  - E-R diagrams

1.9. System Design Tools

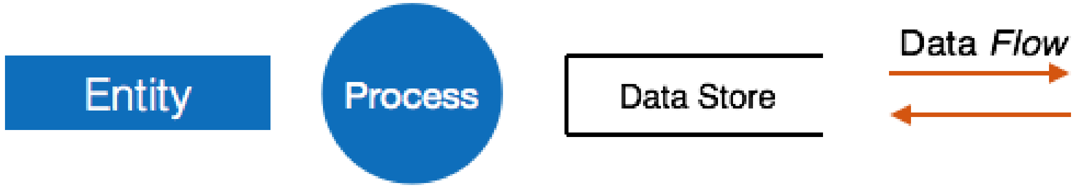
1. Data Flow Diagram(DFD)

- Data flow diagram is graphical representation of flow of data in an information system.
- It is capable of depicting incoming data flow, outgoing data flow and stored data.
- The DFD does not mention anything about how data flows through the system.
- There is a prominent difference between DFD and Flowchart.
- The flowchart depicts flow of control in program modules. DFDs depict flow of data in the system at various levels.
- DFD does not contain any control or branch elements.

1.9. System Design Tools

1. Data Flow Diagram(DFD)

- DFD can represent Source, destination, storage and flow of data using the following set of components –

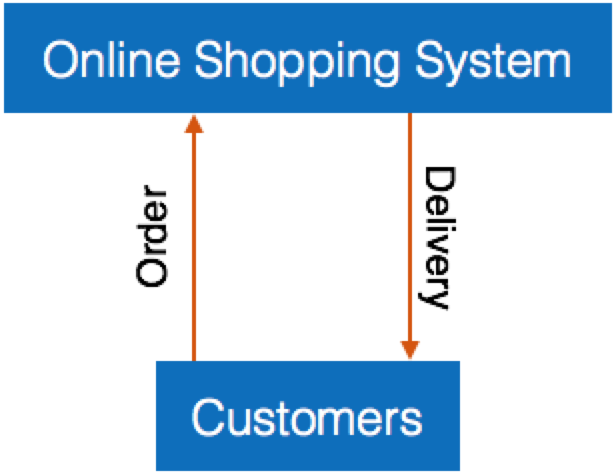


- **Entities** - Entities are source and destination of information data. Entities are represented by a rectangles with their respective names.
- **Process** - Activities and action taken on the data are represented by Circle or Round-edged rectangles.
- **Data Storage** - There are two variants of data storage - it can either be represented as a rectangle with absence of both smaller sides or as an open-sided rectangle with only one side missing.
- **Data Flow** - Movement of data is shown by pointed arrows. Data movement is shown from the base of arrow as its source towards head of the arrow as destination.

1.9. System Design Tools

1. Data Flow Diagram(DFD)

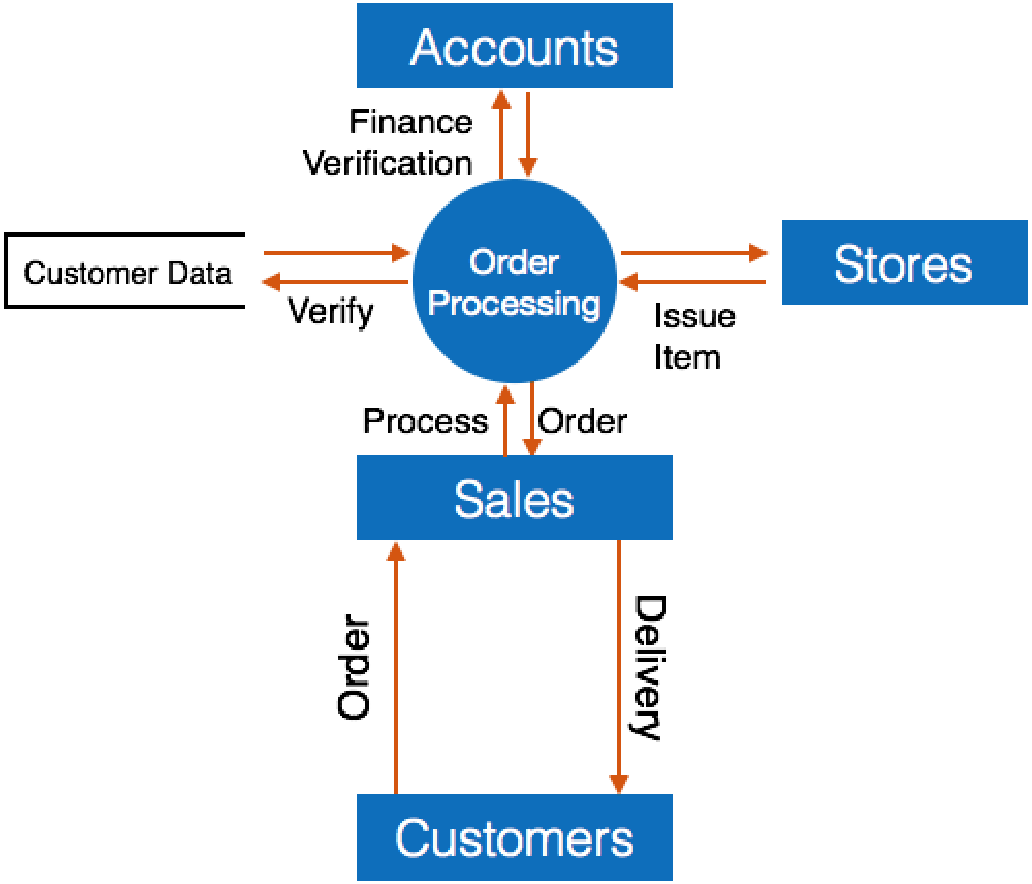
- **Level 0** - Highest abstraction level DFD is known as Level 0 DFD, which depicts the entire information system as one diagram concealing all the underlying details. Level 0 DFDs are also known as context level DFDs.



1.9. System Design Tools

1. Data Flow Diagram(DFD)

- **Level 1** - The Level 0 DFD is broken down into more specific, Level 1 DFD. Level 1 DFD depicts basic modules in the system and flow of data among various modules. Level 1 DFD also mentions basic processes and sources of information.



1.9. System Design Tools

2. Context Diagram

- The context diagram is used to establish the context and boundaries of the system to be modelled: which things are inside and outside of the system being modelled, and what is the relationship of the system with these external entities.
- A context diagram, sometimes called a level 0 data-flow diagram, is drawn in order to define and clarify the boundaries of the software system. It identifies the flows of information between the system and external entities. The entire software system is shown as a single process.

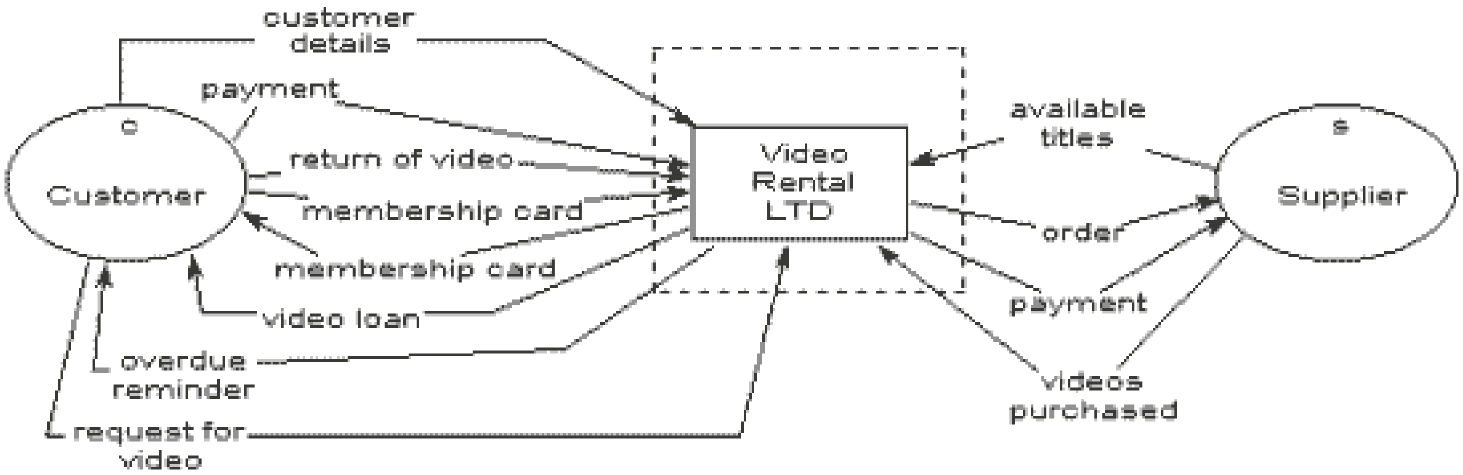


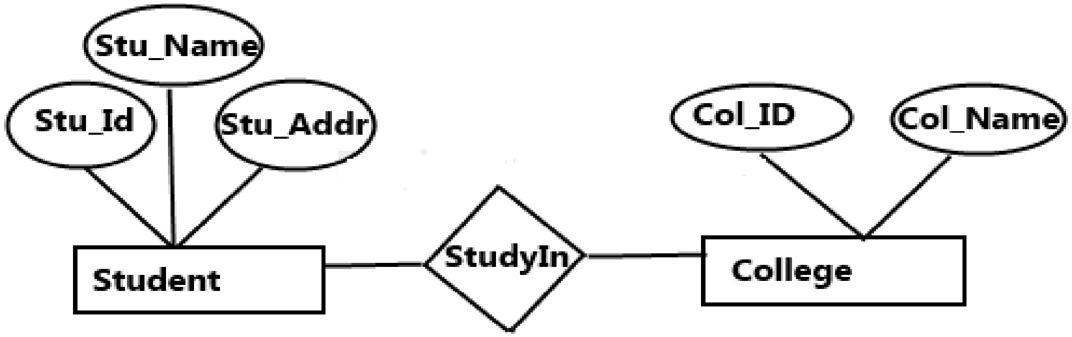
Figure :Context diagram for Video-Rental LTD

1.9. System Design Tools

3.Entity Relationship Diagrams (E-R Diagram)

- Entity Relationship Diagrams (ERD) are graphic illustration used to display object or events within a system and their relationships to one another.
- It 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 and the **inter-relationship among these entities**.



Sample E-R Diagram

Rectangle represents Entity, Ellipse represents Attribute, Diamond represents Relationship between the entities.

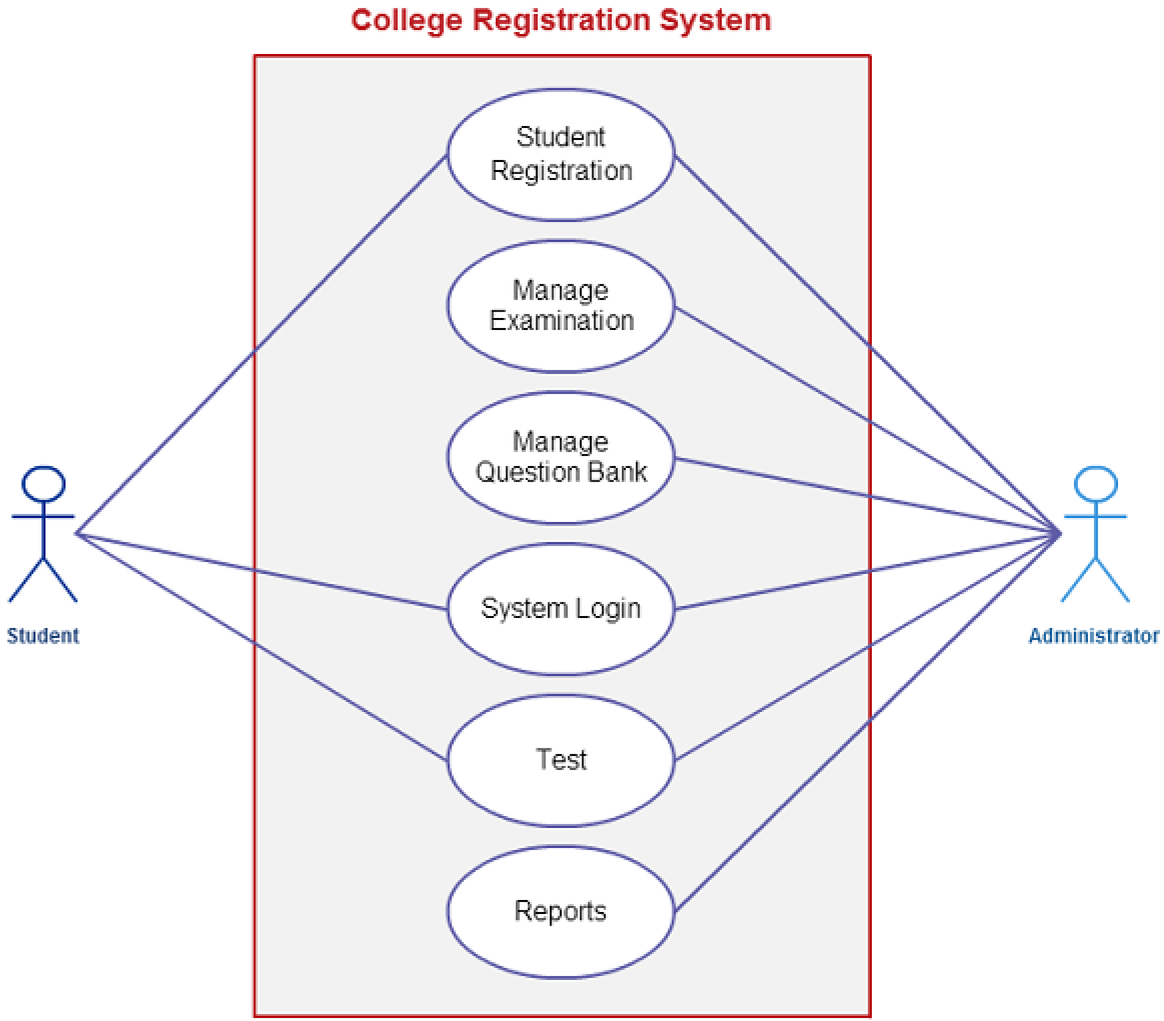
1.9. System Design Tools

4. Use Case Diagram

- A **use case diagram** is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved.
- A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.
- The use cases are represented by either circles or ellipses.

1.9. System Design Tools

- **4. Use Case Diagram**



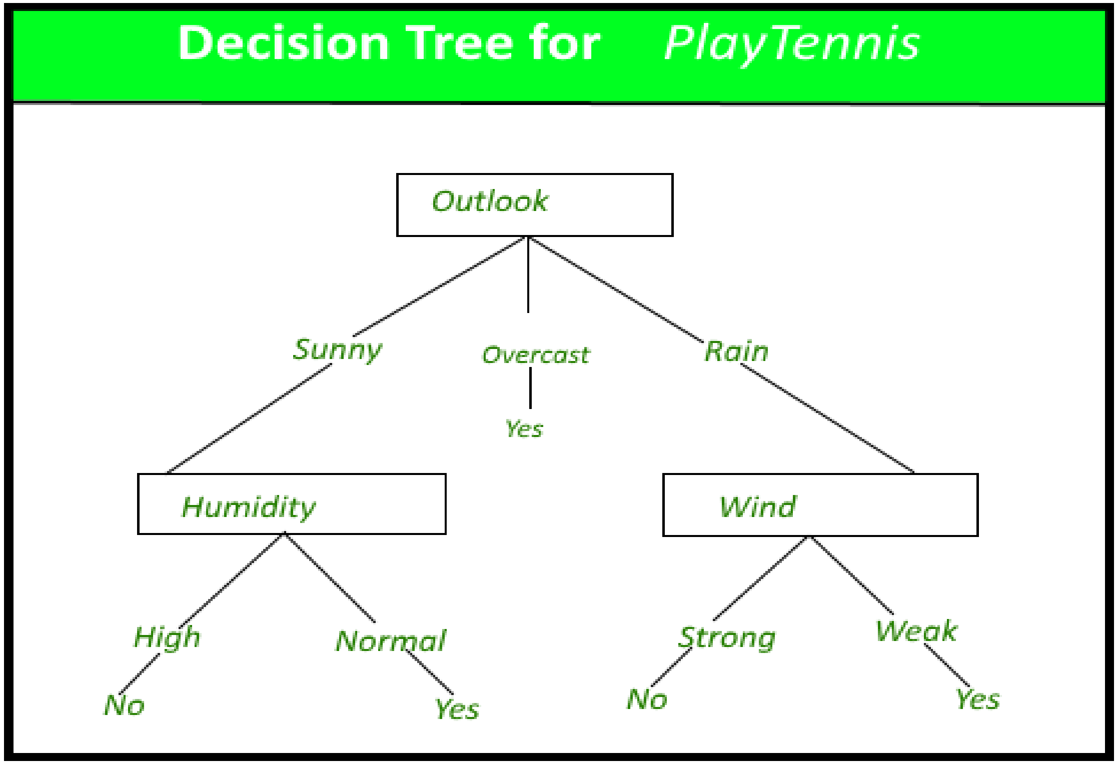
1.9. System Design Tools

5. Decision Tree

- Decision Tree provide a graphic representation of decision logic that helps non-computer people easy to understand. The principles for the development of decision tree are relatively forward.
    1. Identify all conditions
    2. Find out values these conditions may take or assume
    3. List all possible outcomes
- Decision Tree are graphical representations of the decision table. These are also available and aid the construction of decision tables.
- A decision tree helps to show the paths that are possible in a decision following an action or decision by the user.

1.9. System Design Tools

- **5. Decision Tree**



1.9. System Design Tools

6. Decision Table

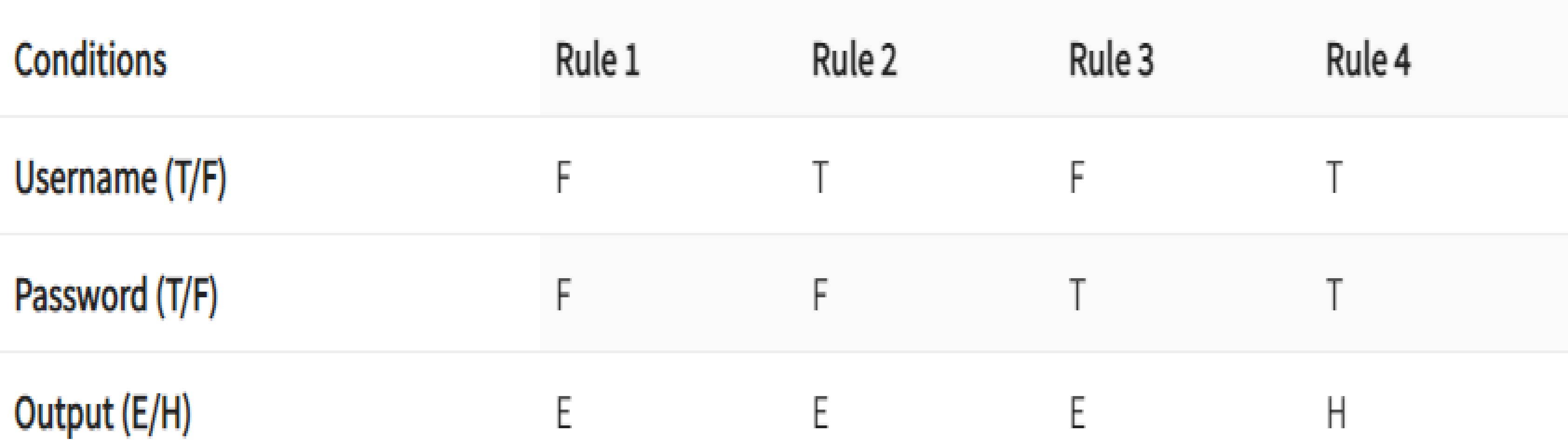
- Using decision tables, decision trees conditions and outcomes are listed in the form of two-dimensional tables. A decision table, as compared to a decision tree, checks all the possible combinations that might arise for all conditions.

General Format of Decision Tables

- List of conditions
  - Columns representing logical combinations of conditional value
- Lit of outcomes
  - Resulting outcome for each set of conditions
- Decision table is a tabular method for describing the logic of the decisions to be taken.

1.9. System Design Tools

- **6. Decision Table For a Login System that displays error message or navigates to home page.**



Conditions	Rule 1	Rule 2	Rule 3	Rule 4
Username (T/F)	F	T	F	T
Password (T/F)	F	F	T	T
Output (E/H)	E	E	E	H

1.9. System Design Tools

S.NO.	DECISION TABLE	DECISION TREE
1.	Decision Tables are tabular representation of conditions and actions.	Decision Trees are graphical representation of every possible outcome of a decision.
2.	We can derive decision table from decision tree.	We can not derive decision tree from decision table.
3.	It helps to clarify the criteria.	It helps to take into account the possible relevant outcomes of decision.
4.	In Decision Tables, we can include more than one 'or' condition.	In Decision Trees, we can not include more than one 'or' condition.
5.	It is used when there are small number of properties.	It is used when there are more number of properties.
6.	It is used for simple logic only.	It can be used for complex logic as well.
7.	It is constructed of rows and tables.	It is constructed of branches and nodes.