Unit 1: Introduction to C Programming

Agendas

- **Introduction to Programming Language**
- **Programming Approach: Top down and Bottom up Approach**
- **Structured Programming**
- **History of C**
- **Algorithms**
- **Pseudocode and Flowchart**
- **Coding**
- **Compilation and Execution, Structure of C program**
- **Debugging.**

Introduction to Programming Language

- **Program is the specific set of instructions, understandable to computer.**

- **A programming language is a computer language that is used by programmers (developers) to communicate with computers. It is a set of instructions written in any specific language (C, C++, Java, Python) to perform a specific task.**

- **The computer has its own language and any communication with the computer must be in its language or translated into this language.**

- ***Programming is an art, skill, poetry that is mastered through immense practice, patience, and experience.***

Types of Programming Language

The primary classifications of programming languages are:

1. Low-Level Language:

- **Machine Languages**
- **Assembly Languages**

2. High level Languages:

- **Low-level languages are closer to the language used by a computer, while high-level languages are closer to human languages.**

Machine level language

- **Computers are made of electronic devices and they can understand only electronic pulse and no-pulse (or ‘1’ and ‘0’) conditions. Therefore, all instructions and data should be written using binary codes 1 and 0.**

- **Machine language is a collection of binary digits or bits that the computer reads and interprets.**
- **Machine languages are the only languages understood by computers.**
- **While easily understood by computers, machine languages are almost impossible for humans to use because they consist entirely of numbers.**

Machine Instruction	Machine Operation
0000 0000	Stop
0000 0001	Rotate bristles left
0000 0010	Rotate bristles right
0000 0100	Go back to start of program
0000 1000	Skip next instruction if switch is off

Assembly level language

- **A program written in assembly language consists of a series of instructions mnemonics that correspond to a stream of executable instructions, when translated by an assembler, that can be loaded into memory and executed.**

- **Assembly languages use keywords and symbols, much like English, to form a programming language but at the same time introduce a new problem.**
- **The problem is that the computer doesn't understand the assembly code, so we need a way to convert it to machine code, which the computer does understand.**

- **Assembly language programs are translated into machine language by a program called an assembler.**

Example:

- **Machine language :**

  10110000 01100001

- **Assembly language :**

  mov a1, #061h

– Meaning:

Move the hexadecimal value 61 into the processor register named "a1".

High Level Language

- **High-level languages allow us to write computer code using instructions resembling everyday spoken language (for example: print, if, while) which are then translated into machine language to be executed.**
- **Programs written in a high-level language need to be translated into machine language before they can be executed.**

- **Some programming languages use a compiler to perform this translation and others use an interpreter.**

- **Examples of High-level Language:**
- **C++**
- **JAVA**
- **PHYTON eg: print(2+3)**

Language Processor

- **Most programs are written in a high-level language such as C, Java, or Python. These languages are designed more for people, rather than machines.**

- **Simply put, high-level languages simplify the job of telling a computer what to do. However, since computers only understand instructions in machine code (in the form of 1's and 0's), we can not properly communicate with them without some sort of a translator.**

This is why *language processors* exist.

- **The language processor is a special translator system used to turn a program written in a high-level language or assembly language, which we call "source code", into machine code, which we call "object program" or "object code".**

- **There are three types of language processors:**

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

Assembler

- **An assembler is a computer program that translates assembly language statements into machine language codes.**
- **The assembler takes each of the assembly language statements from the source code and generates a corresponding bit stream using 0s and 1s.**
- **The output of the assembler in the form of sequence of 0s and 1’s is called object code or machine code.**
- **This machine code is finally executed to obtain the results.**

Compiler

- **The compiler is a computer program that translates the source code written in a high-level language into the corresponding object code of the low-level language.**

- **This translation process is called compilation. The entire high-level program is converted into the executable machine code file.**

- **Compiled languages include COBOL, FORTRAN,C, C++, etc.**

Interpreter

- **The interpreter is a translation program that converts each high-level program statement into the corresponding machine code.**
-
- **This translation process is carried out just before the program statement is executed.**
- **Instead of the entire program, one statement at a time is translated and executed immediately.**
- **The commonly used interpreted language is Python and PERL.**

Assembler	Compiler	Interpreter
1.Assembler converts source code written in assembly language into machine code and then that machine code is executed by a computer.	1.A compiler converts high-level language program code into machine language and then executes it.	1.Interpreter converts source code into the intermediate form and then convert that intermed-iate code into machine language.
2.Assembler converts Assembly language to machine language at once.	2.Compiler scans the entire program first before translating into machine code.	2.Interpreter scans and translates the program line by line to equivalent machine code.
3.Input to the assembler is assembly language code.	3.Compiler takes entire program as input.	3.Interpreter takes single instruction as input.
4.It is difficult to debugging.	4.compiler is slow for debugging because errors are displayed after entire program has been checked.	4.Interpreter is good for fast debugging.
5.Example:GAS,GNU.	5.C,C++,Java,C#.	5.Python,perl,VB,LISP.

Compiler	Interpreter
The compiler converts a program into machine code as a whole	Interpreter converts the program into machine code statement by statement
The compiler creates object code file	Interpreter does not create object code file
A compiler converts a high-level program that can be executed many times	An interpreter converts high-level program each time it is executed.
Programs execution is fast	Programs execution is slow
Compiler displays syntax errors after compiling the whole program	Interpreter displays syntax error on each statement of program

Program development Life cycle

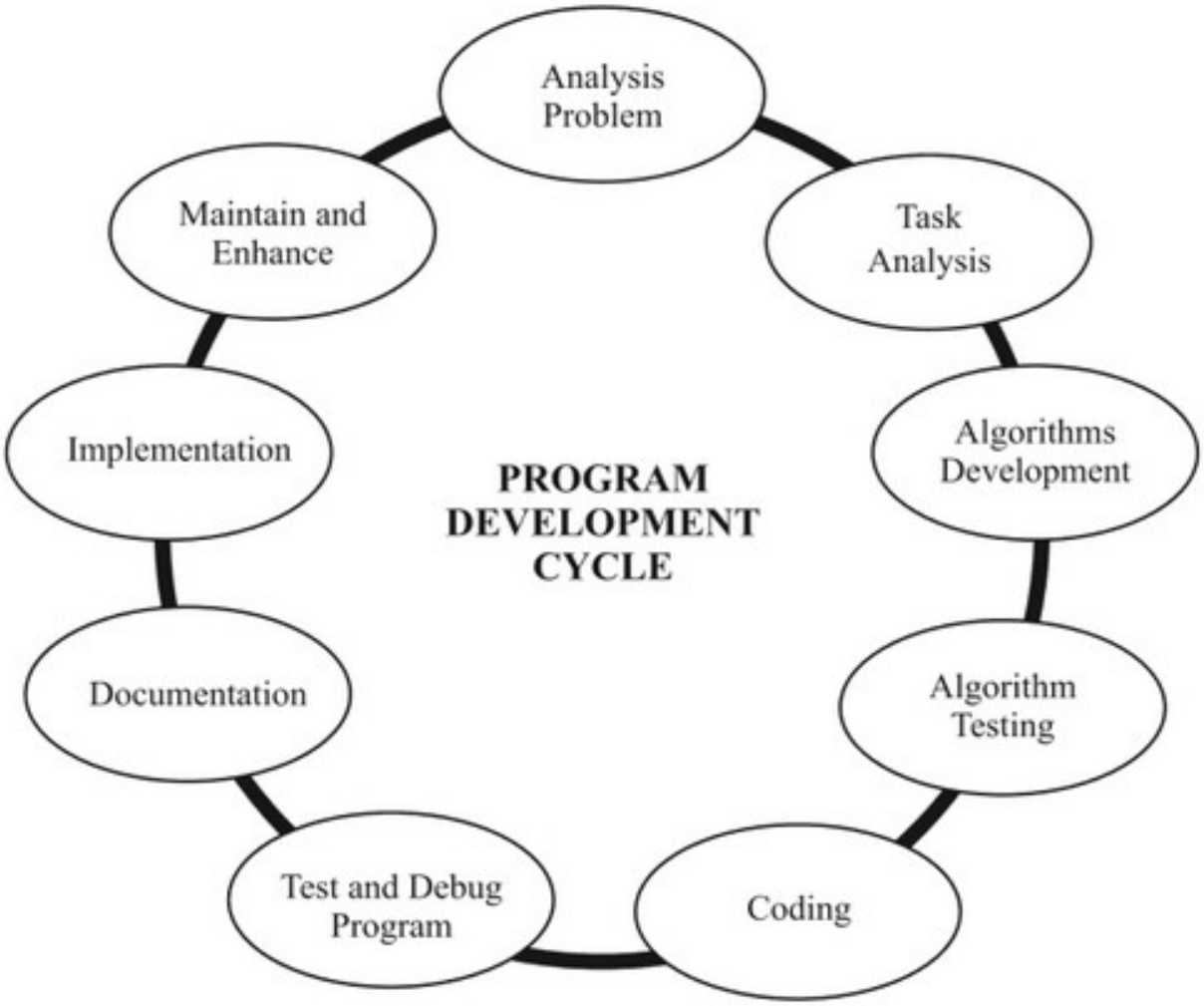
- 

Figure 1.2: Program Development Cycle

Program Development Life Cycle

Before starting the process of writing a program (coding), the programmer has to determine the problem that needs to be solved. There are different approaches to problem solving. Most require breaking the problem into a series of smaller steps.

Development cycle of a program includes the following phases:

1. Analyze/Define the Problem: Firstly, the problem is analyzed precisely and completely and then defined.

2. Task Analysis: After analyzing the problem the developer needs to develop various solutions to solve the given problem. From these solutions, the optimum solution (by experimenting with all the solution) is chosen, which can solve the problem comfortably and economically.

Program Development Life Cycle

3. Developing Algorithm: After selecting the appropriate solution, algorithm is developed to depict the basic logic of the selected solution. An algorithm depicts the solution in logical steps (sequence of instructions). Further algorithm is represented by flowcharts. These tools make program logic clear and they eventually help in coding.

4. Algorithm Testing: Before converting the algorithms into actual code it should be checked for accuracy. The main propose of checking algorithm is to identity major logical errors at an early stage because logical errors are often difficult to detect and correct at later stages. The testing also ensures that the algorithm is the 'true" one and it should work for both normal as well as unusual data.

Program Development Life Cycle

5. Coding: After meeting all the design considerations the actual coding of the program takes place in the chosen programming language. Depending upon application domain available resources, a program can be written by using computer languages of different levels such as machine, assembly or high level languages.

6. Test and Debug the Program: It Is common for the initial program code to contain errors. A program compiler and programmer-designed test data machine tests the code for syntax errors. The results obtained are compared with results calculated manually from this test.

Program Development Life Cycle

7. Documentation: Once the program is free from all the errors, it is the duty of the program developers to ensure that the program is supported by suitable documentation. These documentation should be supplied to the program users. Documenting a program enables the user to operate the program correctly. It also enables other persons to understand the program clearly so that it, if necessary be modified, or corrected by someone other than the original programmer.

8. Implementation: After performing all the above mentioned steps, the program is Installed on the end user's machine. At this stage, users are also provided with all the essential documents so that they can understand how the program works. The implementation can be viewed as the final testing because only after using the program the user can point out the drawbacks.

Program Development Life Cycle

9. Maintenance and Enhancement: After the program is implemented, it should be properly maintained by taking care of the changing requirements of its users and system. The program should be regularly enhanced by adding additional capabilities. This phase is also concerned with detecting and fixing the errors which were missed in testing phase. Since this step generates user feedback the programming cycle continues as the program is modified or reconstructed to meet the changing needs.

Software Development Model

- Software development life cycle (SDLC) model show the ways to navigate through the complex and demanding process of software building. A project’s quality, time-frames, budget, and ability to meet the stakeholders’ expectations largely depend on the chosen model.
- Waterfall Model
- V-Model
- Incremental Model
- RAD Model
- Agile Model
- Iterative Model
- Spiral Model

Waterfall Model

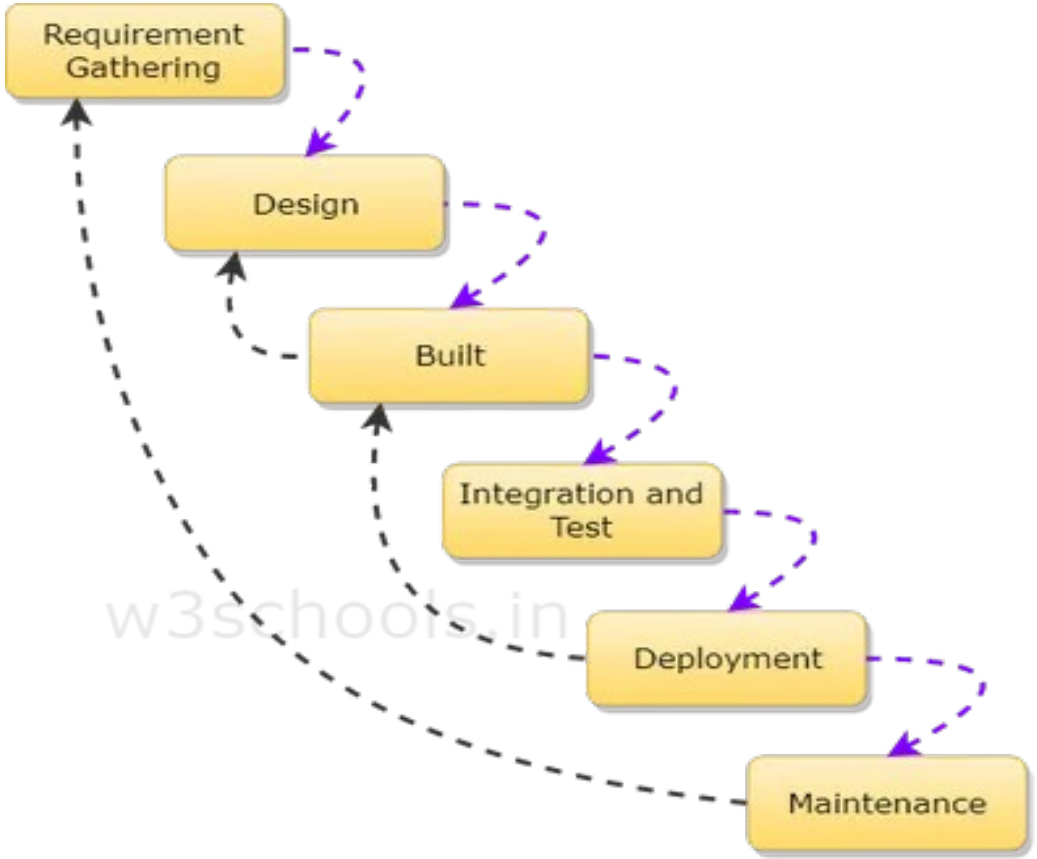


Fig: SDLC Waterfall Model

V-Model

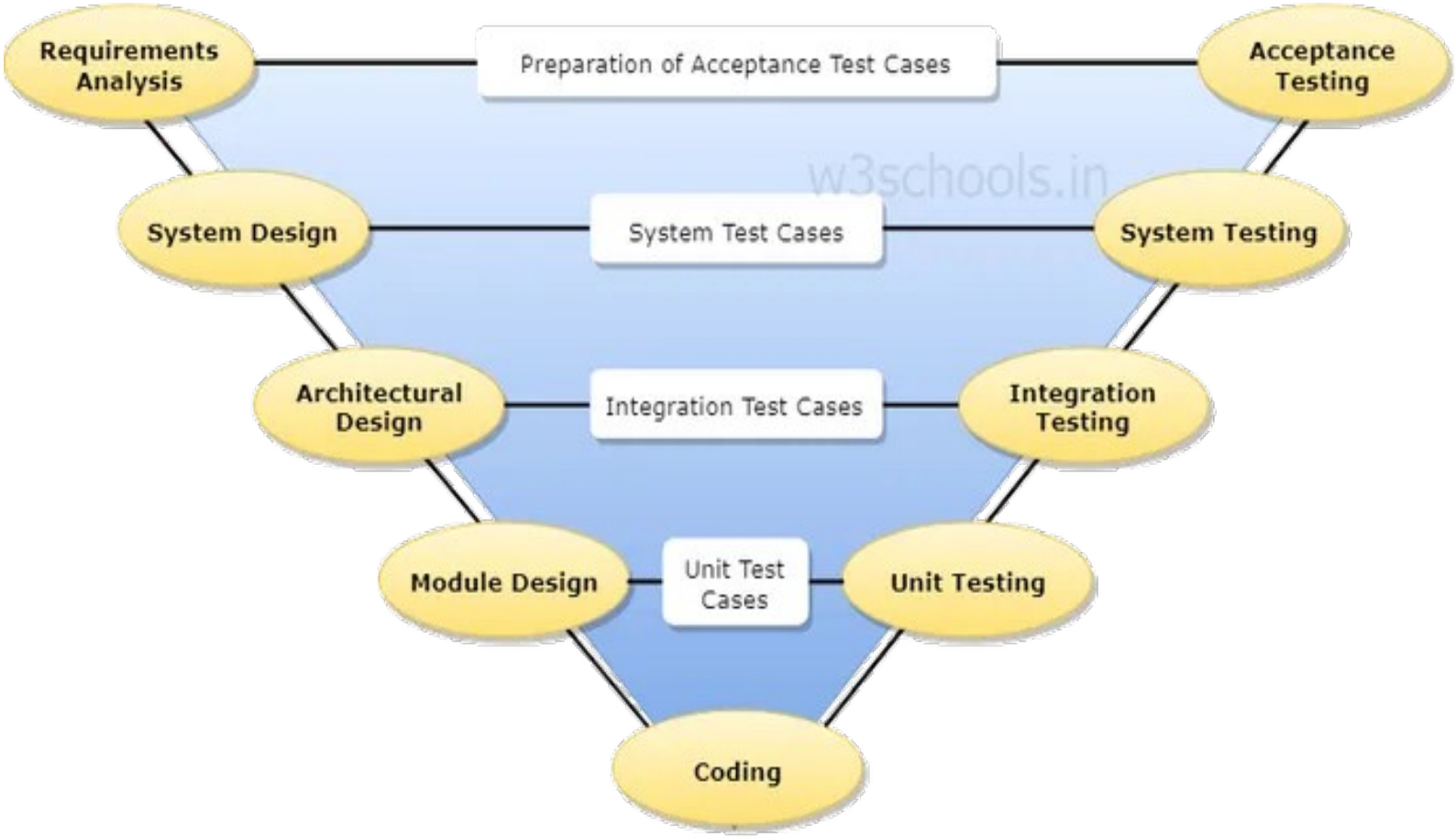
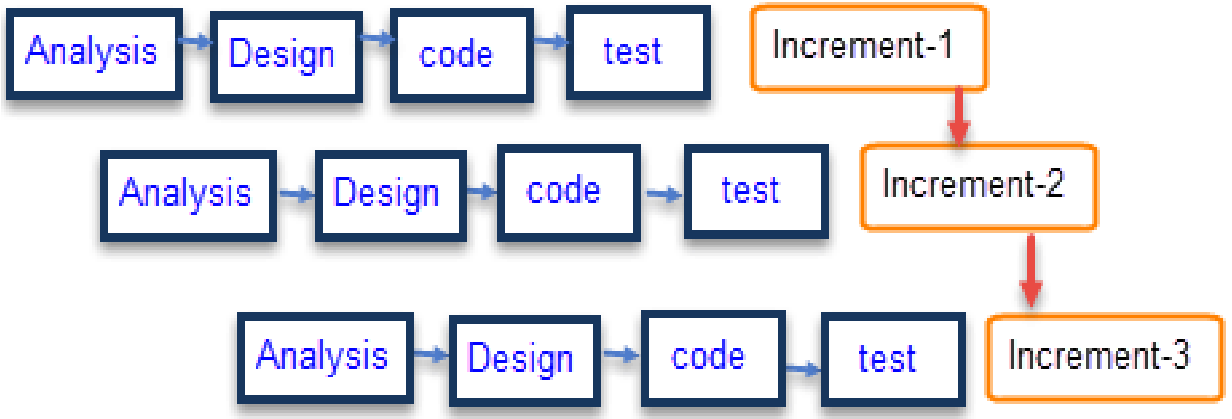


Fig: The Design of the SDLC V-Model

Incremental Model



Incremental Model

RAD Model

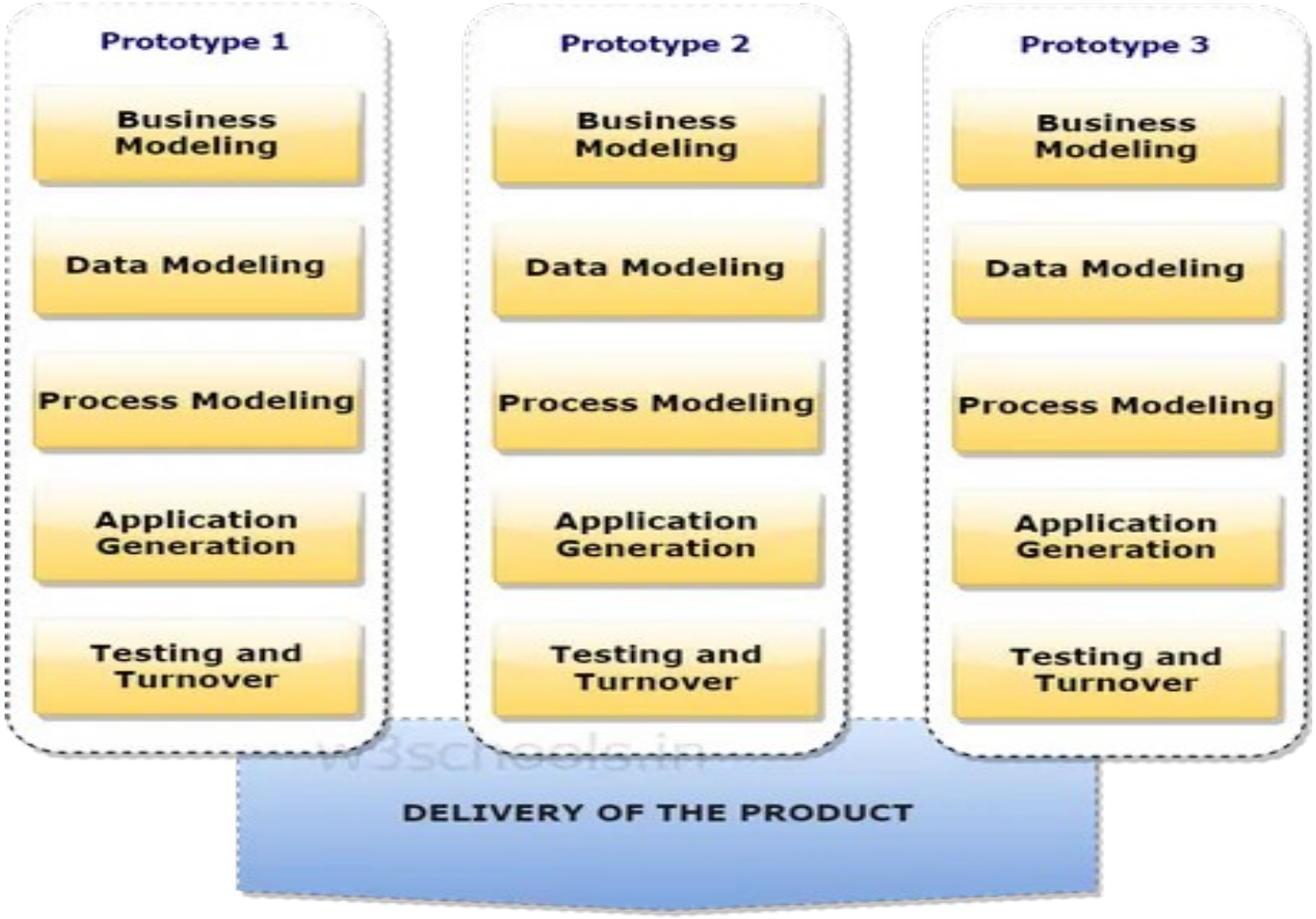


Fig: SDLC RAD Model

Agile Model

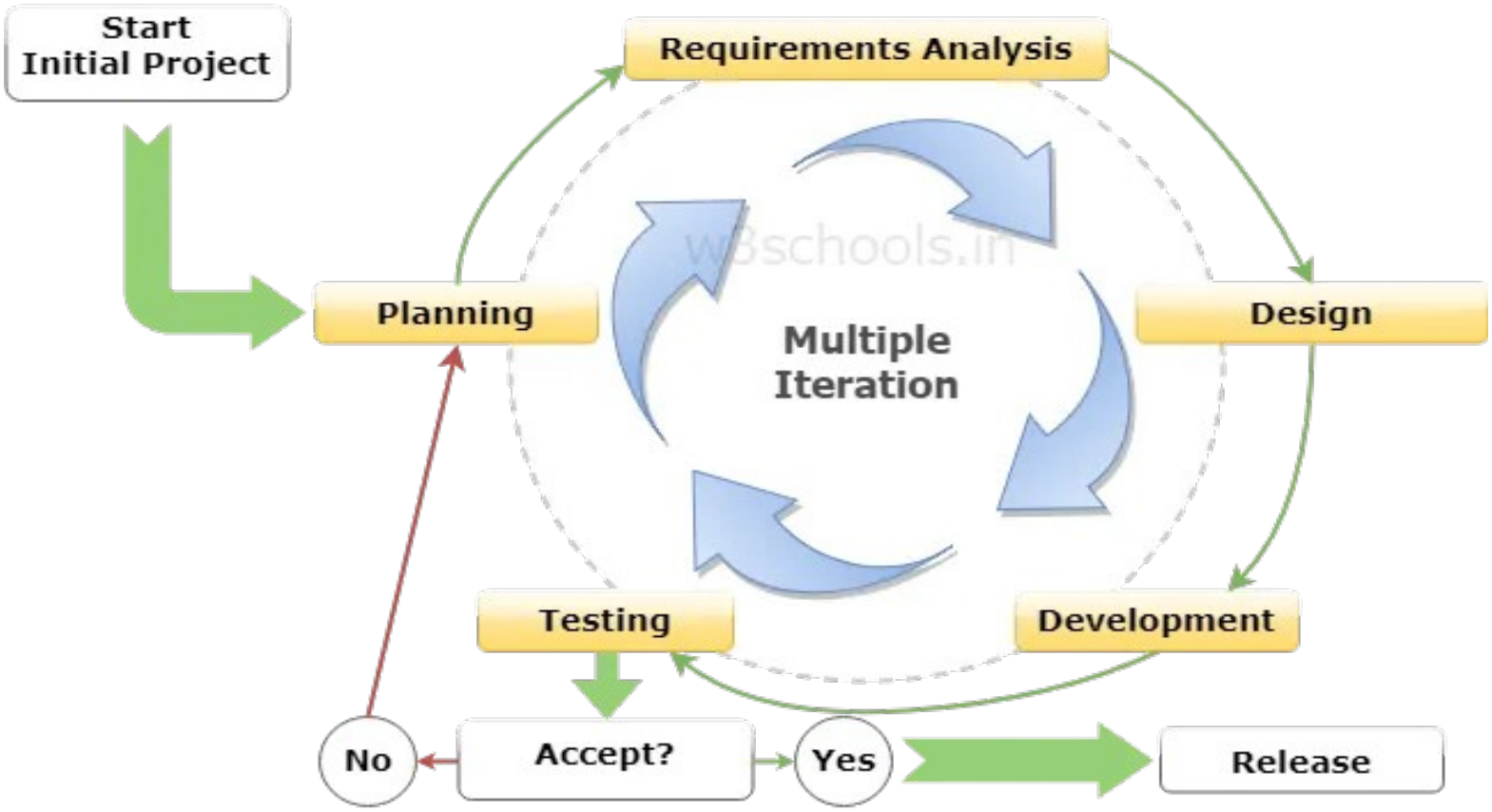


Fig: SDLC Agile Software Development Model

Iterative Model

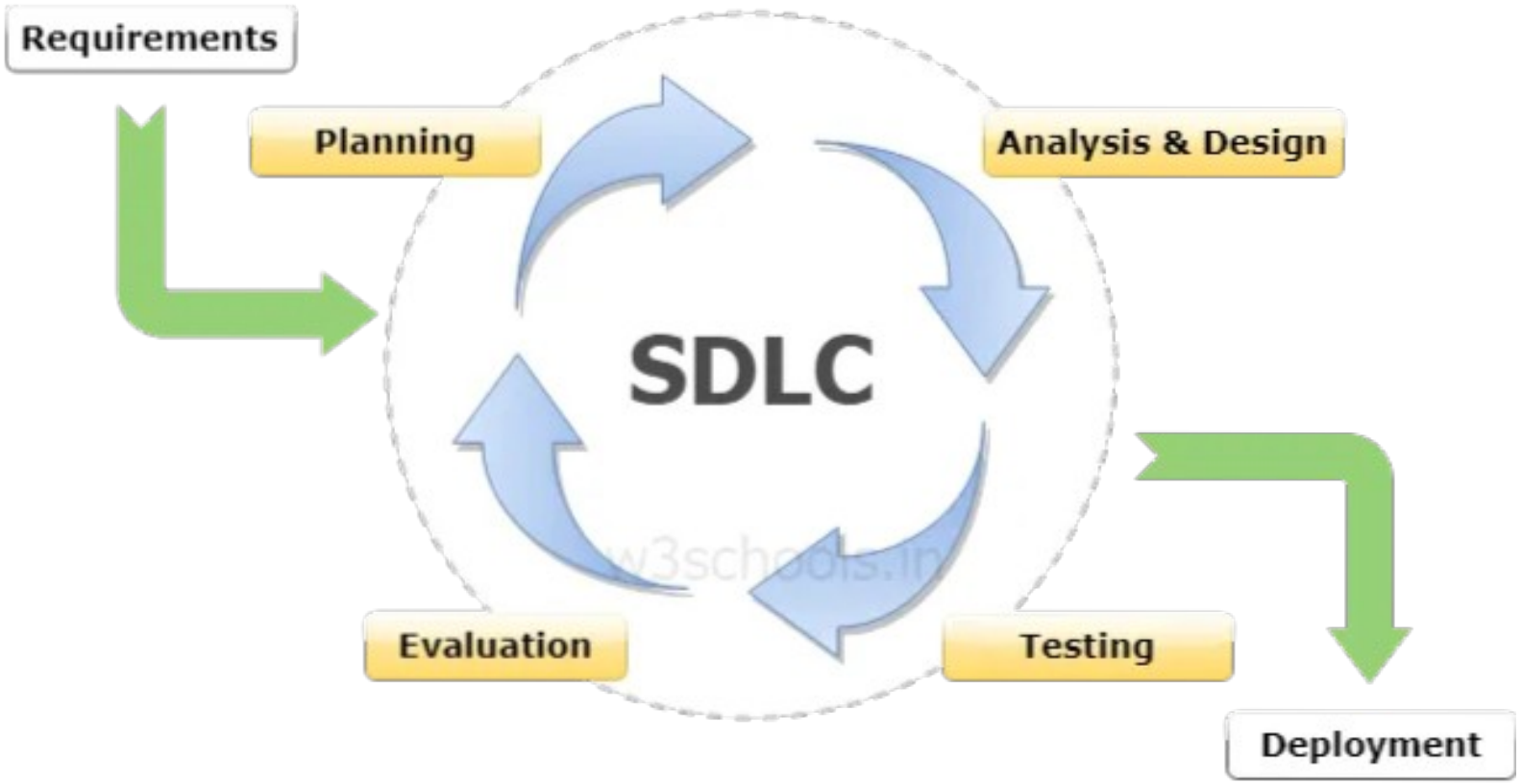
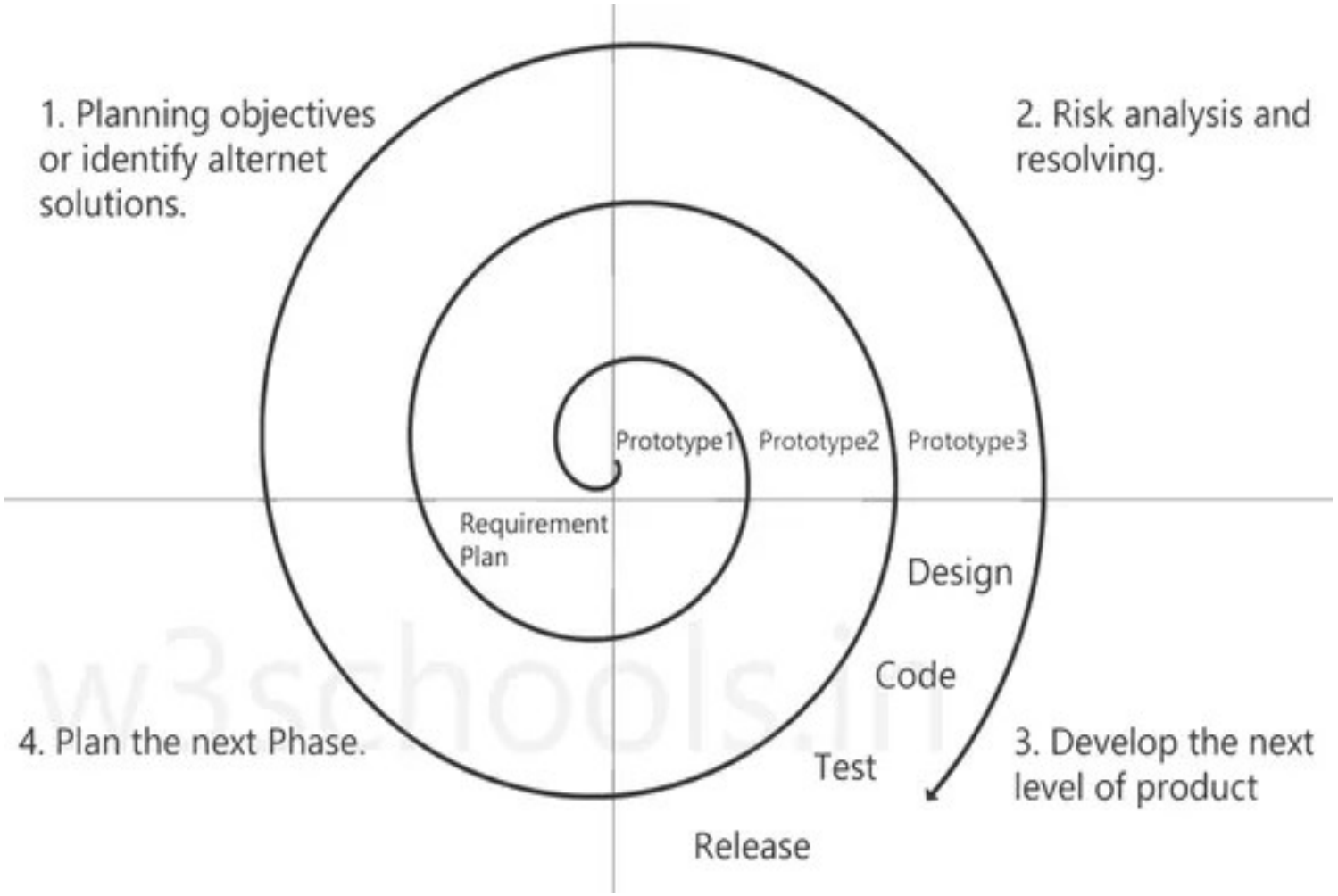
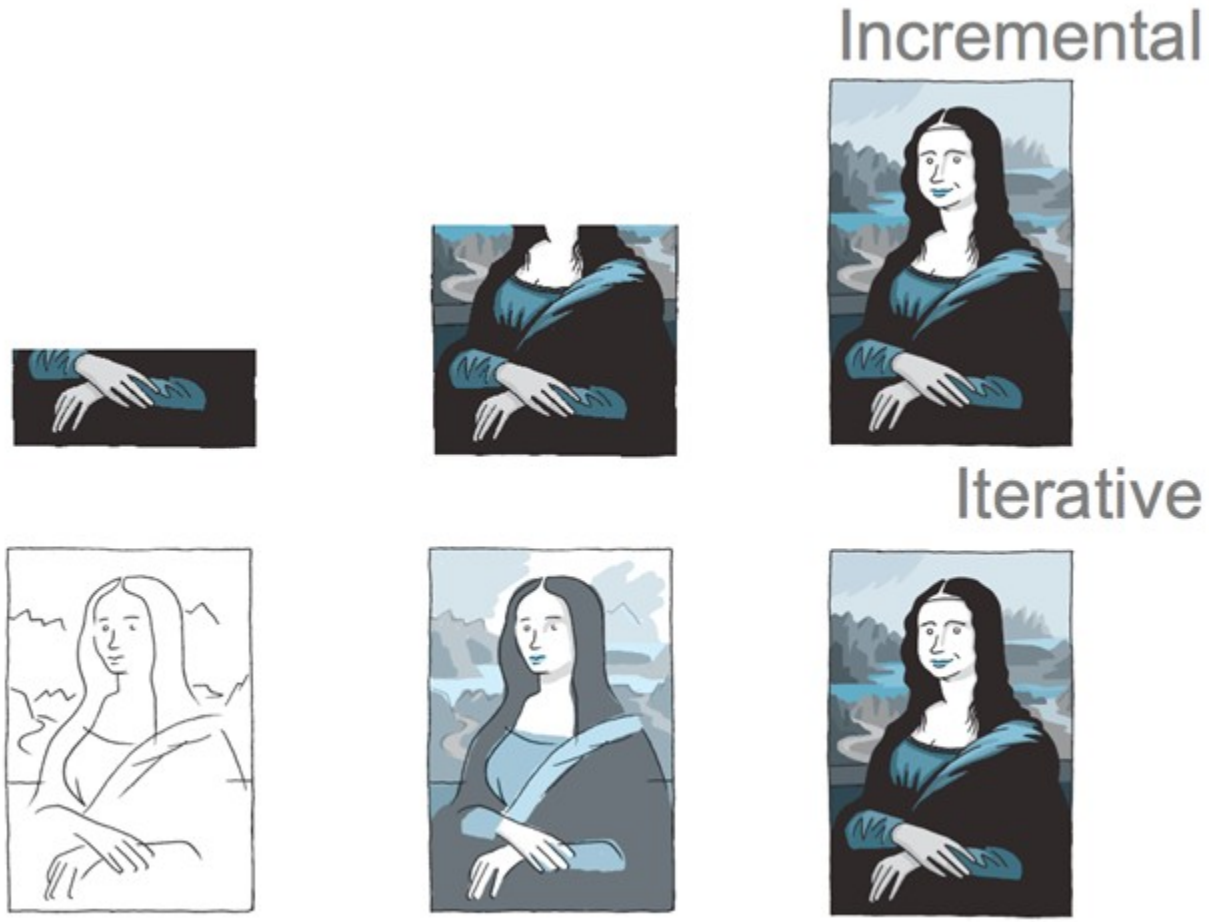


Fig: SDLC Iterative Model

Spiral Model



Incremental vs Iterative

- 

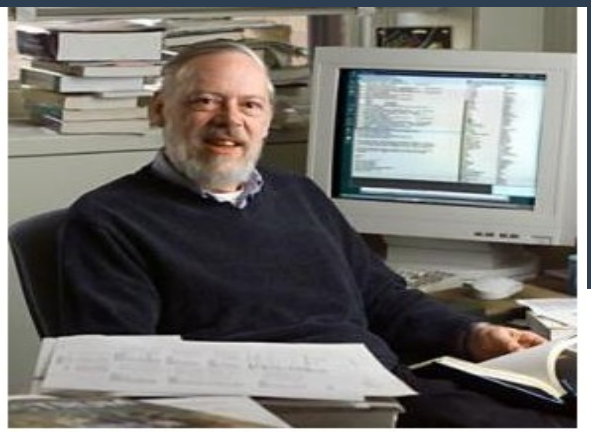
Programming Approach: Top down and Bottom up Approach

A system consists of components, which have components of their own; indeed a system is a hierarchy of components. The highest-level component correspond to the total system. To design such a hierarchy there are two possible approaches: top-down and bottom-up.

The top-down approach (C language) starts from the highest-level component of the hierarchy and proceeds through to lower levels. By contrast, a bottom-up approach starts with the lowest-level component of the hierarchy and proceeds through progressively higher levels to the top-level component.

In top down approach, main() function is written first and all sub functions are called from main function. Then, sub functions are written based on the requirement. Whereas, in bottom up approach, code is developed for modules and then these modules are integrated with main() function.

Introduction

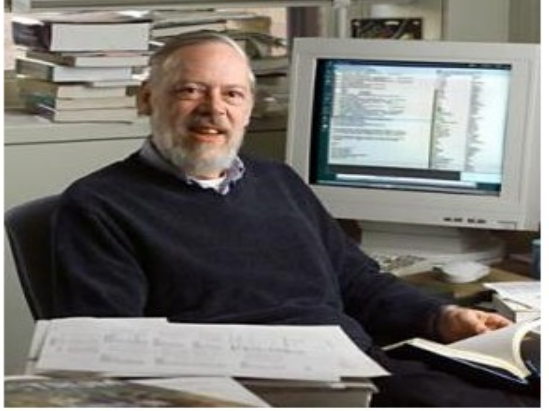


Dennis Ritchie (1941-2011)

- **C is a mid-level structured oriented programming language, used in general-purpose programming, developed by Dennis Ritchie at AT&T Bell Labs, the USA, between 1969 and 1973.**

- **Any programming Language can be divided in to two categories. (High level language) and (Low level language)**
- **But C is considered as a Middle level Language because it can be used for developing system programming as well as application programming.**

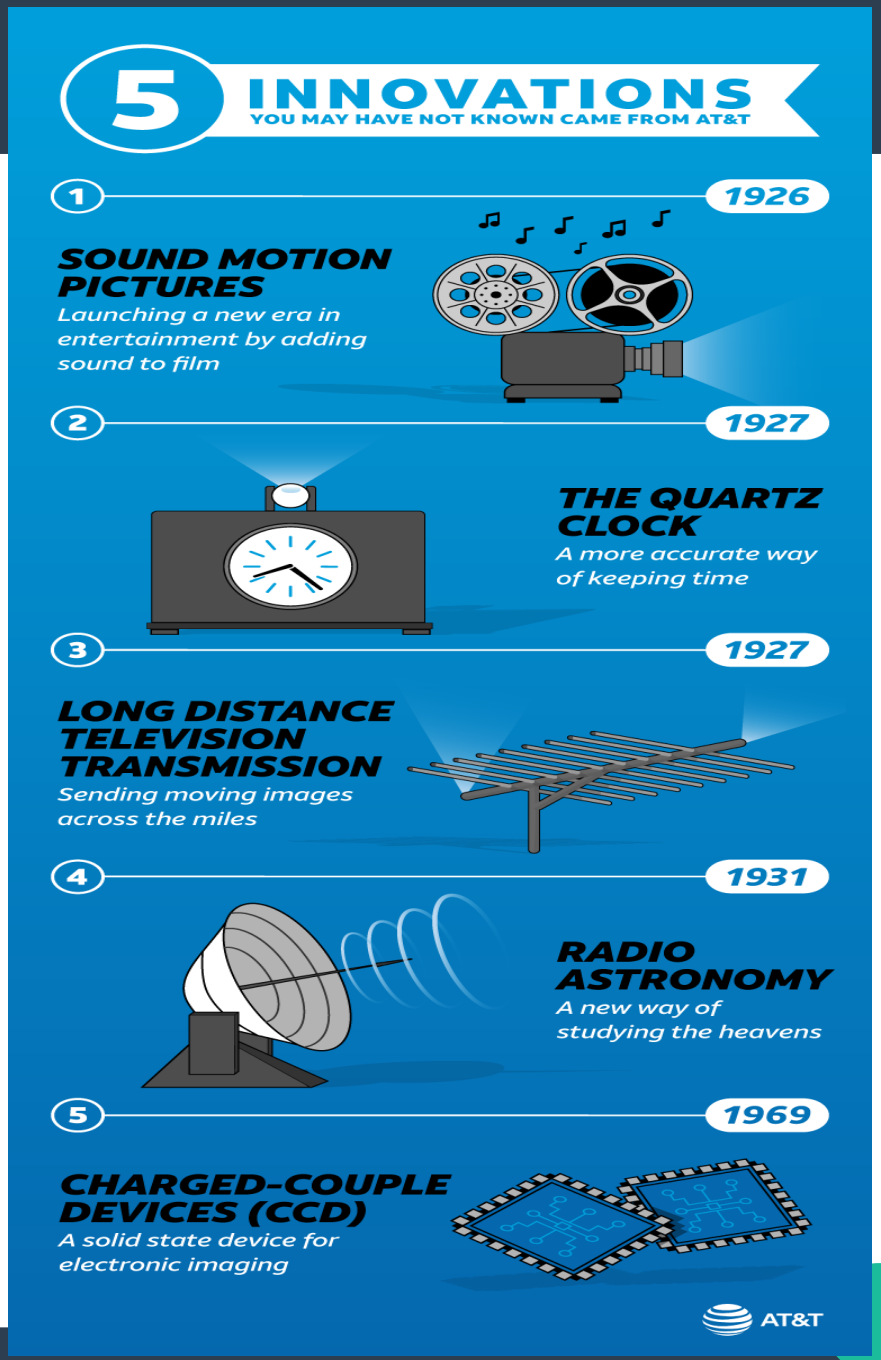
Introduction



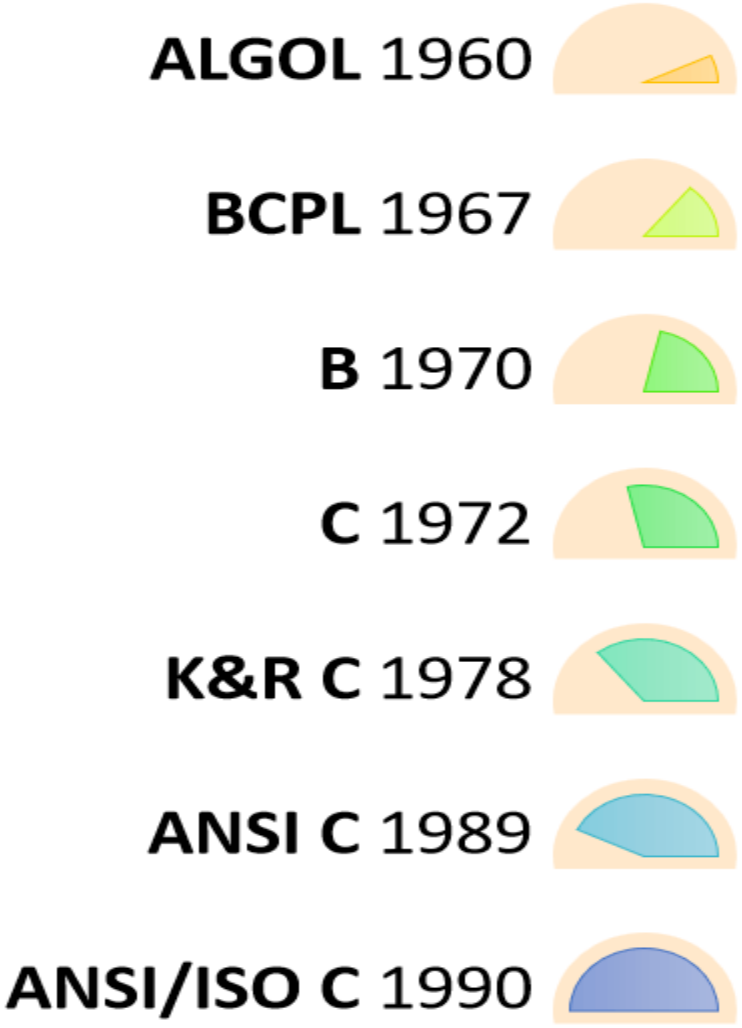
Dennis Ritchie (1941-2011)

- In 1988, the American National Standards Institute (ANSI) had formalized the C language.
- C was invented to write operating systems.
- C has been written in assembly language.
- Different operating Systems like Windows, Linux and many other complex programs like Oracle database, Git, Python interpreter, and more are written in C.

Inventions at AT & T Bell Labs (Nokia Bell Labs)



History of C



History of C

History of C

- **The base or father of programming languages is ‘ALGOL.’ It was first introduced in 1960. ‘ALGOL’ was used on a large basis in European countries. ‘ALGOL’ introduced the concept of structured programming to the developer community.**
- **In 1967, a new computer programming language was announced called as ‘BCPL’ which stands for Basic Combined Programming Language. BCPL was designed and developed by Martin Richards, especially for writing system software. This was the era of programming languages.**
- **Just after three years, in 1970 a new programming language called ‘B’ was introduced by Ken Thompson that contained multiple features of ‘BCPL.’ This programming language was created using UNIX operating system at AT&T and Bell Laboratories. Both the ‘BCPL’ and ‘B’ were system programming languages.**

History of C

- **In 1972, a great computer scientist Dennis Ritchie created a new programming language called ‘C’ at the Bell Laboratories.**
- **It was created from ‘ALGOL’, ‘BCPL’ and ‘B’ programming languages. ‘C’ programming language contains all the features of these languages and many more additional concepts that make it unique from other languages.**

History of C

- **‘C’ is a powerful programming language which is strongly associated with the UNIX operating system. Even most of the UNIX operating system is coded in ‘C’.**

- **Initially ‘C’ programming was limited to the UNIX operating system, but as it started spreading around the world, it became commercial, and many compilers were released for cross-platform systems.**

History of C

- **Today ‘C’ runs under a variety of operating systems and hardware platforms. As it started evolving many different versions of the language were released.**
- **It became difficult for the developers to keep up with the latest version as the systems were running under the older versions. To assure that ‘C’ language will remain standard, American National Standards Institute (ANSI) defined a commercial standard for ‘C’ language in 1989. Later, it was approved by the International Standards Organization (ISO) in 1990. ‘C’ programming language is also called as ‘ANSI C’.**

Algorithms Pseudocode and Flowchart Coding

There are different ways of representing the logical steps for finding a solution of a given problem.

They are as follows:

(i) Algorithm

(ii) Flowchart

(iii) Pseudo-code

(iv) Coding

Pseudocode

```
Pseudocode

BEGIN

    DECLARE Integer a, b, sum

    OUTPUT("Input number1:")

    INPUT a

    OUTPUT("Input number2:")

    INPUT b

    sum=a+b

    OUTPUT sum

END
```

Pseudocode is a generic way of describing an algorithm without using any specific programming language-related notations.

It is an outline of a program, written in a form, which can easily be converted into real programming statements.

The meaning of pseudocode is ‘falsecode.’

Algorithm

Step 1: Start

Step 2: Declare variables a,b,sum

Step 3: Input values for a and b

Step 4: Calculate sum using sum= a+b

Step 5: Display sum

Step 6: Stop

An algorithm is defined as a finite sequence of explicit instructions, which when provided with a set of input values produces an output and then terminates.

To be an algorithm, the steps must be unambiguous and after a finite number of steps, the solution of the problem is achieved

Algorithm

For example, to establish a telephonic communication between two subscribers, following steps

are to be followed:

(i) Dial a phone number

(ii) Phone rings at the called party

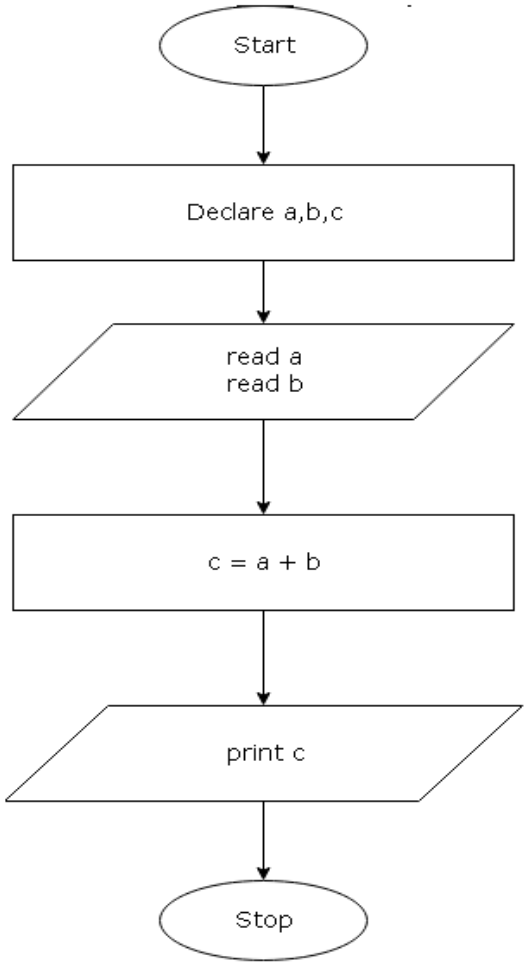
(iii) Caller waits for the response

(iv) Called party picks up the phone

(v) Conversation begins between them

(vi) After the conversation, both disconnect the call

Flow-Chart



A flowchart is a pictorial representation of an algorithm in which the steps are drawn in the form of different shapes of boxes and the logical flow is indicated by interconnecting arrows.

The boxes represent operations and the arrows represent the sequence in which the operations are implemented.

Coding

Coding is the process of writing series of instructions by converting the algorithm in a specific programming language to get computer perform specific task for us.

Coding can be done in various programming languages , depending upon the need and interest of a developer.

```
//WAP in C to add two integer number
#include <stdio.h>

int main() {
    int a,b,sum;
    printf("Enter two values:\n");
    scanf("%d%d", &a,&b);
    sum = a+b;
    printf("The sum is %d", sum);
    return 0;
}
```

Programming Paradigm

Programming paradigm refers to how a program is written in order to solve a problem. Programming can be classified into three categories:

1. Unstructured Programming

2. Structured Programming

3. Object-Oriented Programming

Unstructured Programming

- In unstructured programming language, the whole program must be written in single continuous way; there is no stop or broken block.
- Also known as Linear programing is a method for straightforward programming in a sequential manner. This type of programming does not involve any decision making. A general model of these linear programs is:

• Read a data value.

• Compute an intermediate result.

• Use the intermediate result to compute the desired answer.

• Print the answer.

• Stop.

- Unstructured Programming is a type of programming that generally executes in sequential order i.e., these programs just not jumped from any line of code and each line gets executed sequentially.

Structured Programming

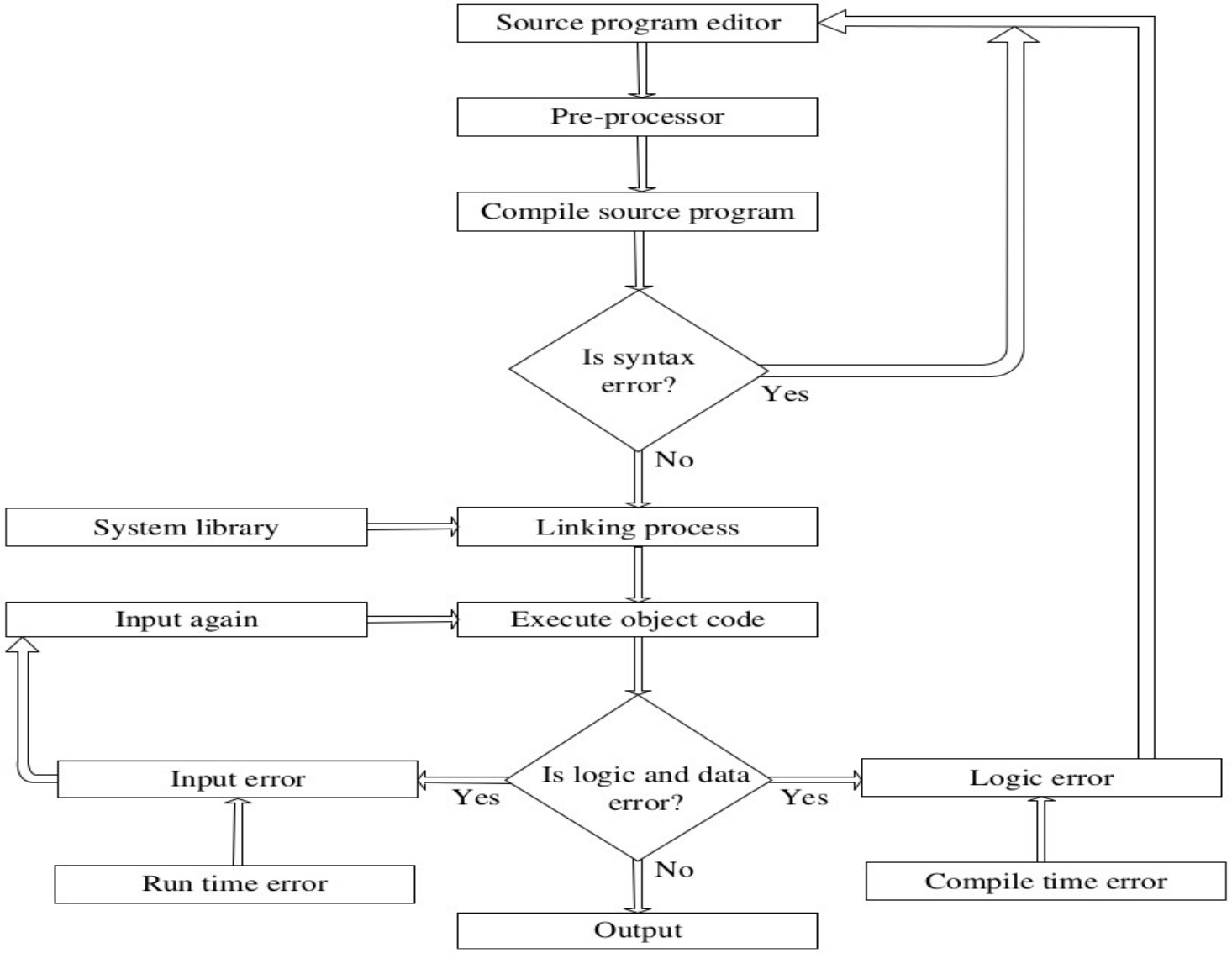
- Structured programming refers to the process in which we break the overall job down into separate pieces of modules/blocks such as:
- Decision making blocks like if-else-elseif, switch-cases,
- Repetitive blocks like For-loop, While-loop, Do-while loop etc
- subroutines/procedures - functions

- C is called a structured programming language because to solve a large problem, C programming language divides the problem into smaller structural blocks (as mentioned above) each of which has a particular responsibility.

Object-Oriented Programming

- **Object-oriented programming is style of computer programming, which promotes building of independent pieces of code that interact with each other.**
- **Object Oriented programming (OOP) is a programming paradigm that relies on the concept of classes and objects.**
- **It allows pieces of programming code to be reused and interchanged between programs.**

How C file are created, compiled and executed ??



Various Step of C Program

C program must go through various phases such as creating a program with editor, execution of preprocessor program, compilation, linking, loading and executing the program. The following steps are essential in C when a program is to be executed in Windows:

(i) Creation of a Program: The program should be written in C editor. The default extension is '.C'. The C program includes preprocessor directives.

(ii) Execution of a Preprocessor Program: After writing a program with C editor the programmer has to compile the program. The preprocessor program executes first automatically before the compilation of the program. A programmer can include other files in the current file. Inclusion of other files is done initially in the preprocessor section.

Various Step of C Program

(iii) Compilation and Linking of a Program: The source program contains statements that are to be translated into object codes. These object codes are suitable for execution by the computer. If a program contains errors the programmer should correct them. If there is no error in the program, compilation proceeds and translated program is stored in another file with the same file name with extension '.o'. This object file is stored on the secondary storage device such as a disc.

Linking is also an essential process. It puts together all other program files and functions that are required by the program. For example, if the programmer is using pow() function, then the object code of this function should be brought from math.h library of the system and linked to the main() program. After linking, the program is stored on the disc and .exe file (for windows os) is created.

(iv) Executing the Program: After the compilation the executable object code will be loaded in the computer's main memory and the program is executed. The loader performs this function. All the above steps/phases of C program can be performed using menu options of the editor. The pre-processor programs are executed before compilation of the main program.

Structure of C program

```
Include header file
Global Declaration
/*     comments    */
/*Function name */
main ()
{
      /*    comments */
      Declaration part
      Executable part
      Function call

}

User-defined functions
```

Documentation Section

Link Section

Definition Section

Global Declaration Section

main ( ) Function Section

{

Declaration part

Executable part

}

Subprogram section

Function **1**

Function **2**

–

–

Function **n**

(User-defined functions)

Structure of C program

Debugging

- ✓ **Debugging is the process of detecting and removing of existing and potential errors in codes/programs.**
- ✓ **When an execution error(run-time error) and logical errors occur, we must first determine its location (where it occurs) within the program. Once the location of the execution error has been identified, the source of the error (why it occurs) must be determined. Knowing where the error occurred often assists in debugging.**
- ✓ **Different techniques are available for finding the location of execution errors and logical errors within a program. Such methods are generally referred to as debugging techniques.**

Debugging

1. Error Isolation: Temporarily deleting or commenting out the line of code to test if the source of errors originate from those lines of codes.

2. Tracing: Tracing involves the use of printf statements to display the values assigned to certain key variables, or to display the values that are calculated internally at various locations within the program.

Most contemporary C compilers include an interactive debugger

3. Watch Values: A watch value is the value of a variable or an expression which is displayed continuously as the program executes.

4. Break Point: A breakpoint is a temporary stopping point within a program. Each breakpoint is associated with a particular instruction within the program. When the program is executed, the program execution will temporarily stop at the breakpoint, before the instruction is executed.

5. Stepping :Stepping refers to the execution of one instruction at a time, typically by pressing a function key to execute each instruction.