C Programming

Facilitator: Krishna Prasad Acharya
Mechi Multiple Campus
BCA Programme

Unit-1 Programming language 10 Hrs

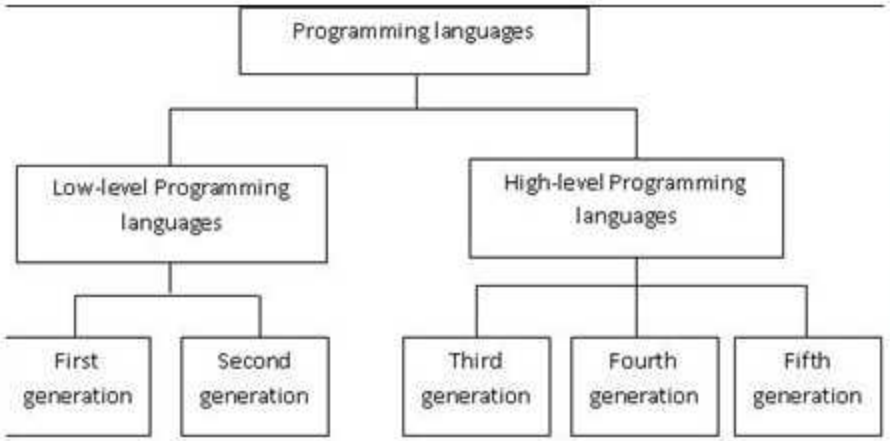
- Introduction to programming language.
- Types of programming language.
- Language processor.
- Types of errors.
- Features of good program.
- Different programming paradigm
- Software development model
- Software/Program development life cycle
- System design tools

Introduction of Language

- A programming language is a set of rules that provides a way of telling a computer what operations to perform.
- A programming language is a set of rules for communicating an algorithm
- It provides a linguistic framework for describing computations
- A programming language is a notational system for describing computation in a machine-readable and human-readable form. A programming language is a notational system for describing computation in a machine-readable and human-readable form.
- A programming language is a tool for developing executable models for a class of problem domains
- English is a natural language. It has words, symbols and grammatical rules. A programming language also has words, symbols and rules of grammar. The grammatical rules are called syntax. Each programming language has a different set of syntax rules

Introduction of Language

- Programming languages can be used to create programs that control the behavior of a computer and serve any purpose.
- Programming languages have evolved over time as better ways have been developed to design them. First programming languages were developed in the 1950s Since then thousands of languages have been developed. Different programming languages are designed for different types of programs.
- In 1954, language **FORTRAN** was invented at IBM by a team led by John Backus; it was the first widely used high level general purpose programming language to have a functional **implementation**.
- Computer languages have some form of written specification of their **Syntax** (Rules) and **Semantic** (Words, phrase & sentences)



Types of Programming language

```
0001001001000101
0010010011101100
10101101001...
```

1.Machine language or 1st GL (Low level):

- Machine language is the only language that can be understood by the computer.
- It contains two symbols,0 and 1.
- Each command in a machine language contains a series of 0's and 1's.
- Programming in machine language is much more complex because we have to remember each and every binary code for the programming language commands.
- A machine language program executes much faster compare to the programs written in other languages.
- It does not require translator program to convert from programming language to machine language code.
- It is difficult to perform debugging process because we have to check entire program code to find out errors.
- It is also difficult to perform modification in the program written in machine language.

Types of Programming language

1. Assimbly language: or 2nd GL (Low level):

- A low level programming language is one step above the machine language. In assembly language we use pre-defined keywords instead of binary code.
- These keywords are known as ***pneumonic op-code***.
- Pneumonic op-codes are easier to remember compare to machine code, because they are in a string format.
- It needs an assembler program to convert pneumonic op-code to its binary code.
- An assembler performs translation of pneumonic op-code into its binary representation at compile time.
- Since program has to convert into its machine code, execution speed of an assembly program is slower than machine language program.
- Programming in assembly language is easy compare to machine language because programmer does not need to remember each and every binary code.
- Simple pneumonic op-code such as ***move***, ***add***, ***copy*** are use which are easy to remember.
- Debugging process is also easy compare to machine language.
- Modification in program is also easy.

```
LOAD r1,b
LOAD r2,h
MUL r1,r2
DIV r1,#2
RET
```

Types of Programming language

3.High Level language: or 3rd Generation of language.

- High level programming language is one step above the low level language.
- It is much more easy compare to machine level language and low level language.
- It uses simple *English* like statements.
- Finding and solving error is also easy in high level language.
- Modification can be performing easily without looking to entire program.
- It needs to be converted from high level language to machine level language for the execution.
- A program such as compiler and interpreter are required to convert high level program into machine language.
- Program written in high level language has the slowest execution speed.
- Examples of high level programming languages are *C, C++, Java, PHP*, etc.

Types of Programming language

3.1 Fourth Generation of language:

- Fourth-generation language is designed to be closer to natural language than a 3GL language.
- A high level language (4GL) that requires fewer instructions to accomplish a task than a third generation language.
- Fourth-generation languages attempt to make communicating with computers as much like the processes of thinking and talking to other people as possible.
- Fourth-generation languages typically consist of English like words and phrases, some of these languages include graphic devices such as icons and onscreen push buttons for use during programming and when running the application.
- Many fourth-generation languages use VB.NET, Access and SQL that support stand-alone as well as web applications.
- Used with databases : Query languages ,Report generators, Forms designers, Application generators

Types of Programming language

3.5 Fifth Generation of language:

- Visual programming allows you to see object oriented structures and drag icons to assemble program blocks.
- A fifth-generation programming language (5GL) is a programming language based on problem solving using constraints given to the program, rather than using an algorithm written by a programmer.
- Fifth-generation languages are used mainly in artificial intelligence research. **Prolog** , **Smalltalk** ,**OPS5** and **Mercury** are examples of fifth-generation languages.
- They are used in the areas of research, robotics, defense system and different types of expert systems.
- Natural languages such as English, Nepali and other languages would be used in the computer making them more intelligent and user-friendly.

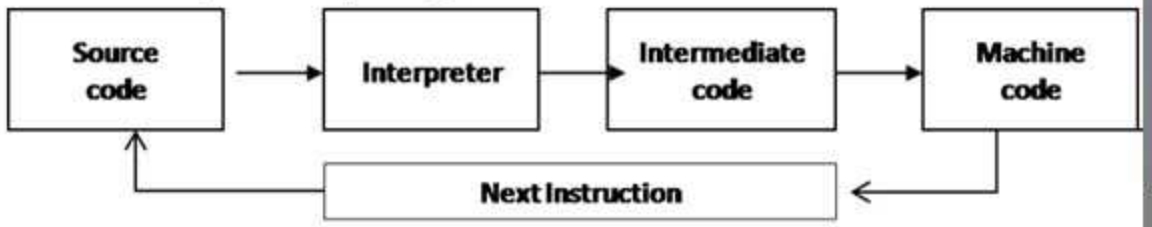
Language Processor

Language processor(Translator)

- A **language processor** is a software program designed or used to perform tasks, such as processing program code to machine code.
- A language processor is a special type of a computer software that has the capacity of translator the source code or program codes into machine codes. The following are different types of language processors are:

1. **Interpreter:**

- An Interpreter is a computer program that performs translation of high level programming code into the machine language.
- It converts one line at a time and then executes it.
- First it converts source code into and intermediary code and then executes it statement by statement.
- An Interpreter program executes slower than compiled program.
- If an error is found, it stops translation.



Interpretation of a program

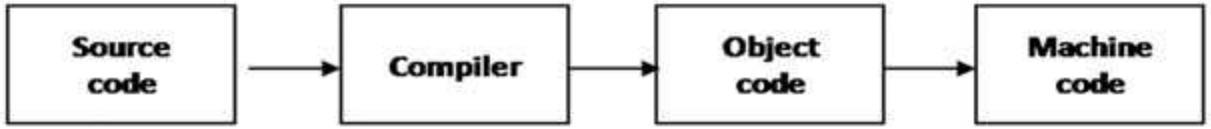
Language Processor

2. **Assembler:**

- An assembler is a one type of translator program.
- It converts assembly language program into the machine language code.
- It scans all the mnemonic op-code into the machine code.
- An assembler generates object code from the assembly language code.

3. **Compiler:**

- A Compiler is a computer program that converts high level language code in to machine code.
- It translates the whole program at a time.
- Since, the whole program is converted at a time, it executes much faster.
- After translation, the compiler generates error report if any error is found.
- Converted program code is also known as object code.



Compilation of a program

Compiler vs interpreter

Compiler

1. It compiles the whole program at a time.
2. Its execution speed is faster compare to interpreter.
3. It first converts whole program and shows error reports at last if any error is found.
4. Performance of the compiled program is better than the interpreted program.
5. It does not provide facility to stop and resume compilation process.

Interpreter

1. it converts source program one line at a time.
2. since it converts the code line by line, execution speed is slow.
3. It stops translation as soon as it found error.
4. Performance is poor compare to compiled program.
5. It can be stop and resume during the interpretation

Program

What is program

- Program is an organized list of instructions that, when executed, it perform any operation, and do the job it is also known a code.
- Without programs, computers are useless. A program is like a recipe. It contains a list of variables and a list of directions (Statements) that tell the computer what to do with the variables.

What is instruction

- The term instruction is often used to describe the programming commands.
- It force the computer to do a specific task.

What are Keywords

- Keyword is a word that is reserved by a program because the word has a special meaning an purpose in any computer language.
- keywords that cannot be used as variable names.

Features of Good Program

- **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.
- **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.
- **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.
- **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.

Features of Good Program

- **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.
- **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.
- **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.

Error in program

- In computer programming, an **error or bug** is an action which is inaccurate or incorrect. In some usages, an error is synonymous with a mistake. An error in a program that causes it to operate incorrectly, it may terminate abnormally (or crash) or normally.
- **Syntax errors**: errors due to the fact that the syntax of the language is not respected.
  Example int a=5: // missing semi column at the end of statement
  - Misspelled variable and function names
  - Missing semicolons
  - Unmatched parentheses, square brackets, and curly braces
  - Using a variable that has not been declared
  - Incorrect format in selection and loop statements
- **Semantic errors**: errors due to an improper use of program statements.
  Example x=(3+5; b+c=a; //missing closing parenthesis
- **Logical errors**: Logic errors occur when a programmer implements the algorithm for solving a problem incorrectly.
  Example y= x/c //where the value of c=0 cause math related error.
- The compiler does not detect these errors.
  - Multiplying when you should be dividing
  - Adding when you should be subtracting
  - Opening and using data from the wrong file
  - Displaying the wrong message

Error in program

Runtime errors

- A type of error that occurs during the execution of a program is known as run-time error. Runtime errors may crash your program when you run it.
- Example: Trying to open a file which is not created or not exist.

Linker errors :

- Linker errors are generated when the linker encounters what looks like a function call; but it cannot find a function with that name.
- This is usually caused by misspelling a C standard function (like main) or not including the header file for a function.
  - Misspelling a standard C function
  - Not including the header file for a function

Compile time:

- Compile time errors are error occurred due to typing mistake, if we do not follow the proper syntax and semantics of any programming language then compile time errors are thrown by the compiler.

- They wont let your program to execute a single line until you remove all the syntax errors or until you debug the compile time errors. The following are usual compile time errors:
  - Syntax errors
  - Typechecking errors
  - Compiler crashes (Rarely)

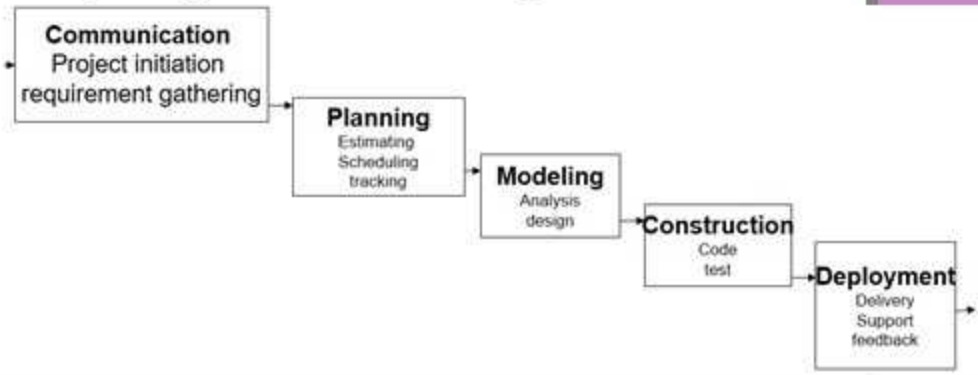
Software development Model

- Software development model is the process of constructing the software depending upon the situation.
- A software development process is the process of dividing software development work into different phases to improve design, product management, and project management.

1.Waterfall model:

This Model suggests a systematic, sequential approach to SW development that begins at the system level and progresses through analysis, design, code and testing

- Used when requirements are well understood in the beginning
- Also called classic life cycle
- A systematic, sequential approach to Software development
- Begins with customer specification of Requirements and progresses through planning, modeling, construction and deployment



Software development Model

Problems of Waterfall model:

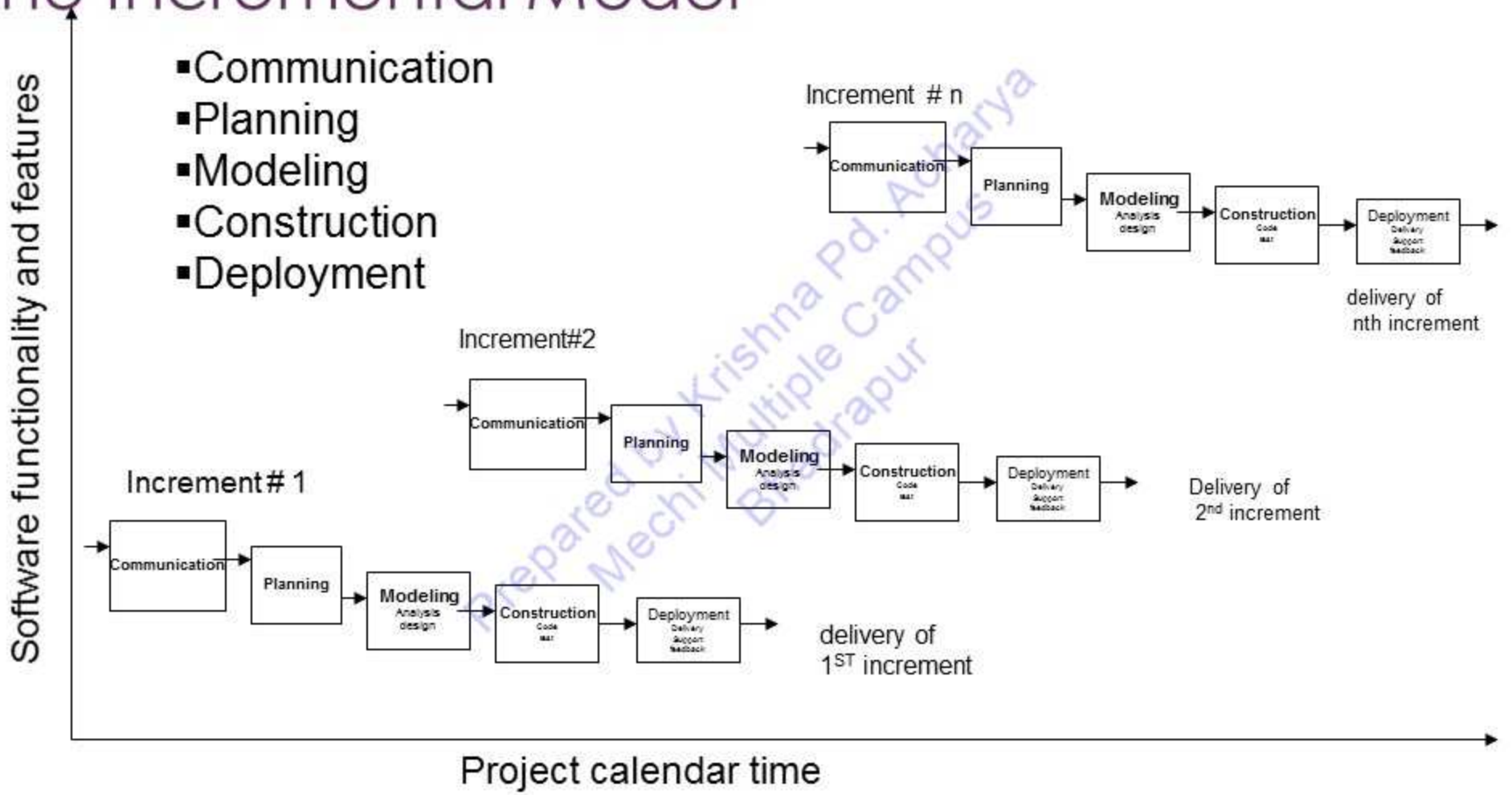
- Real projects rarely follow the sequential flow since they are always iterative
- The model requires requirements to be explicitly spelled out in the beginning, which is often difficult
- A working model is not available until late in the project time plan.
- Blocking stage in every stage.

Incremental Model

- Linear sequential model is not suited for projects which are iterative in nature
- Incremental model suits such projects
- Used when initial requirements are reasonably well-defined and compelling need to provide limited functionality quickly
- Functionality expanded further in later releases
- Software is developed in increments
- Software releases in each increments especially in 1st increment constitutes Core product which supposed to address basic requirements.
- Core product undergoes detailed evaluation by the customer as a result, plan is developed for the next increment
- Plan addresses the modification of core product to better meet the needs of customer
- Process is repeated until the complete product is produced

The Incremental Model

- Communication
- Planning
- Modeling
- Construction
- Deployment



Software development Model

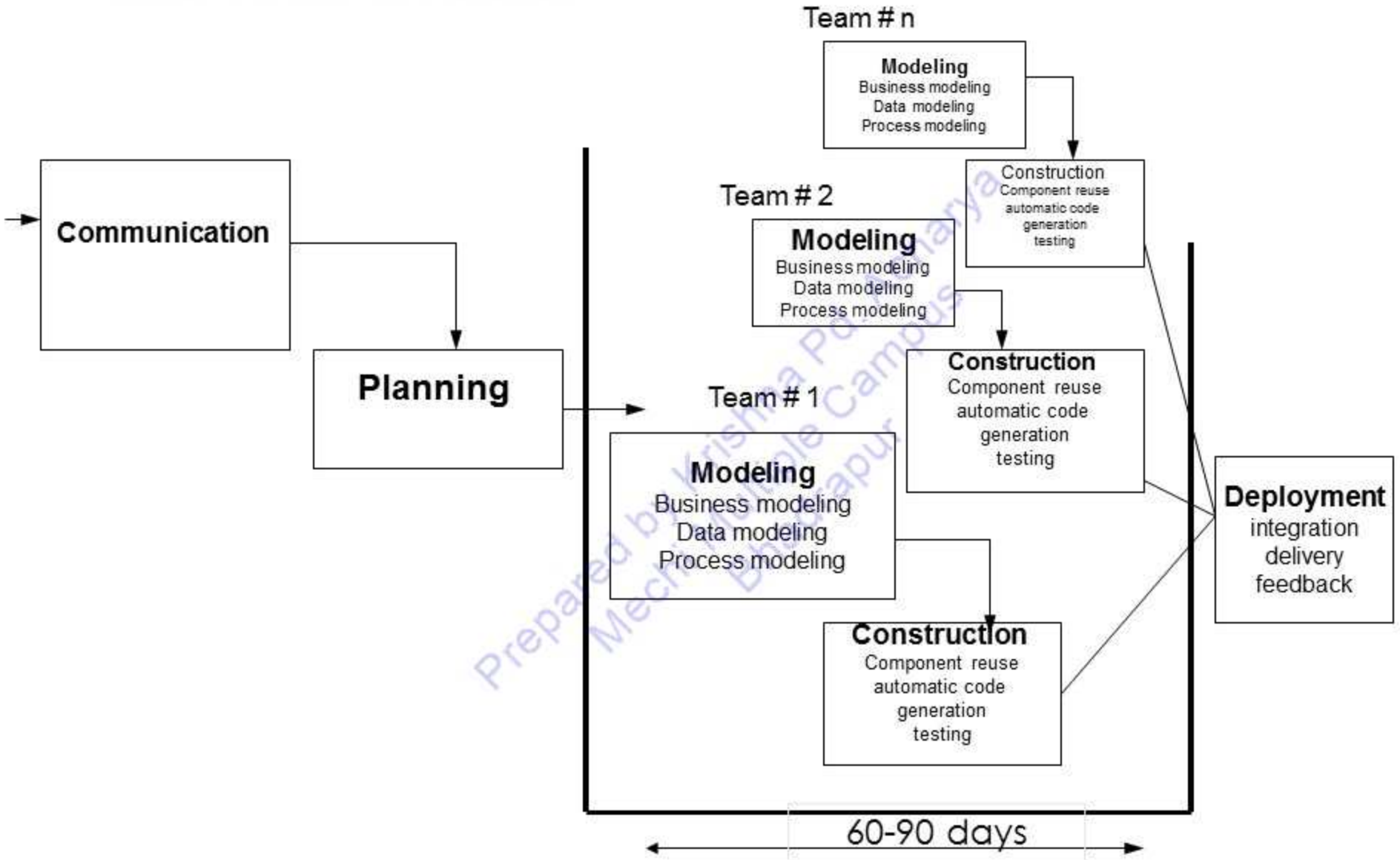
RAD Model (Rapid Application Development)

- An incremental software process model
- Having a short development cycle
- High-speed adoption of the waterfall model using a component based construction approach
- Creates a fully functional system within a very short span time of 60 to 90 days
- Multiple software teams work in parallel on different functions
- Modeling encompasses three major phases: Business modeling, Data modeling and process modeling
- Construction uses reusable components, automatic code generation and testing

Problems in RAD

- Requires a number of RAD teams
- Requires commitment from both developer and customer for rapid-fire completion of activities
- Requires modularity
- Not suited when technical risks are high

The RAD Model

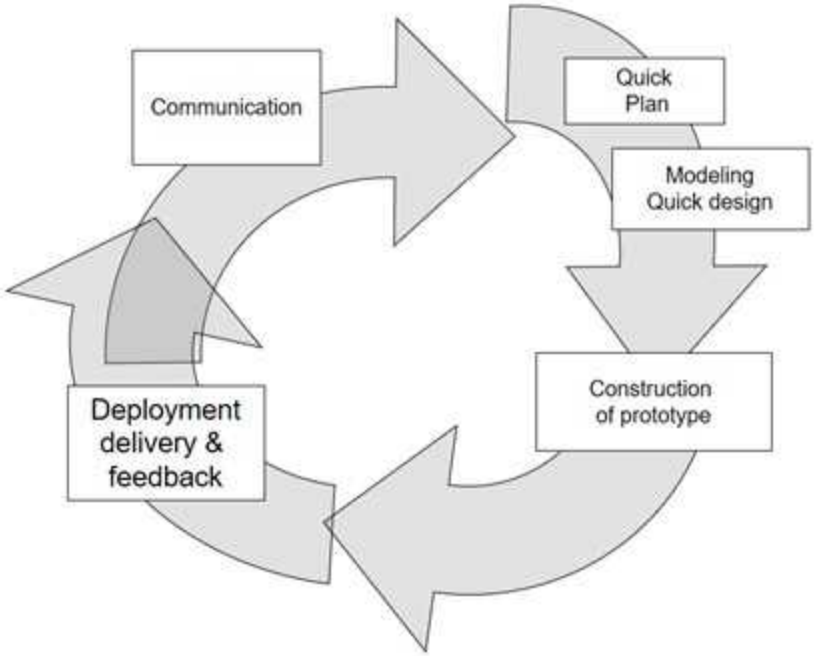


Software development Model

Prototyping Model

- Mock up or model(throw away version) of a software product
- Used when customer defines a set of objective but does not identify input, output, or processing requirements
- Developer is not sure of:
  - efficiency of an algorithm
  - adaptability of an operating system
  - human/machine interaction
- Begins with requirement gathering
- Identify whatever requirements are known
- Outline areas where further definition is mandatory
- A quick design occur
- Quick design leads to the construction of prototype
- Prototype is evaluated by the customer
- Requirements are refined
- Prototype is turned to satisfy the needs of customer

Evolutionary Models: Prototype



Software development Model

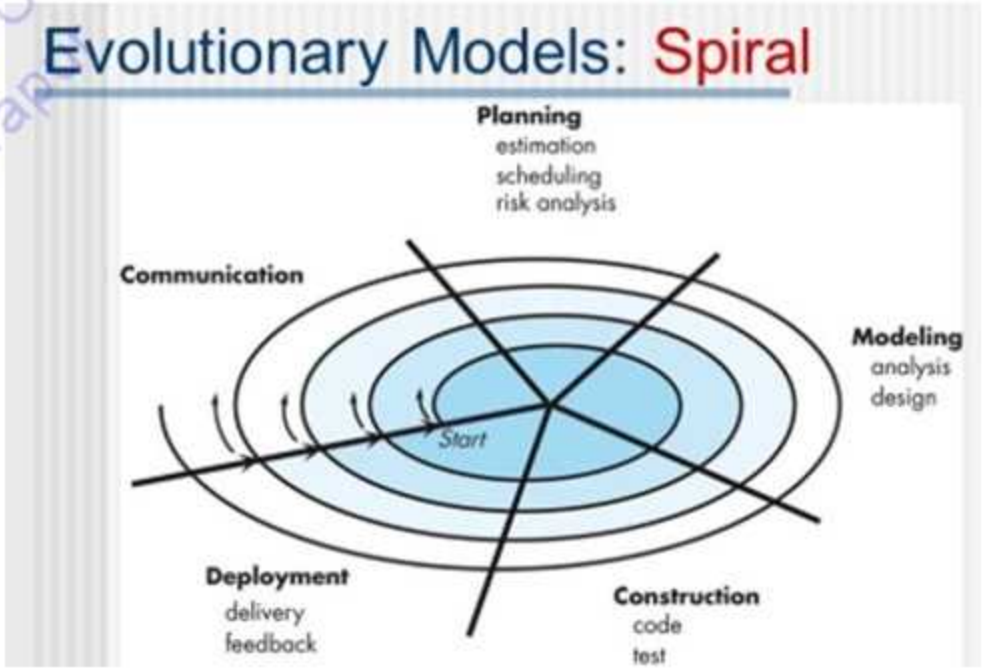
The spiral model

- An evolutionary model which combines the best feature of the classical life cycle and the iterative nature of prototype model
- Include new element : Risk element
- Starts in middle and continually visits the basic tasks of communication, planning , modeling, construction and deployment
- Realistic approach to the development of large scale system and software
- Software evolves as process progresses
- Better understanding between developer and customer
- The first circuit might result in the development of a product specification
- Subsequent circuits develop a prototype and sophisticated version of software

Software development Model

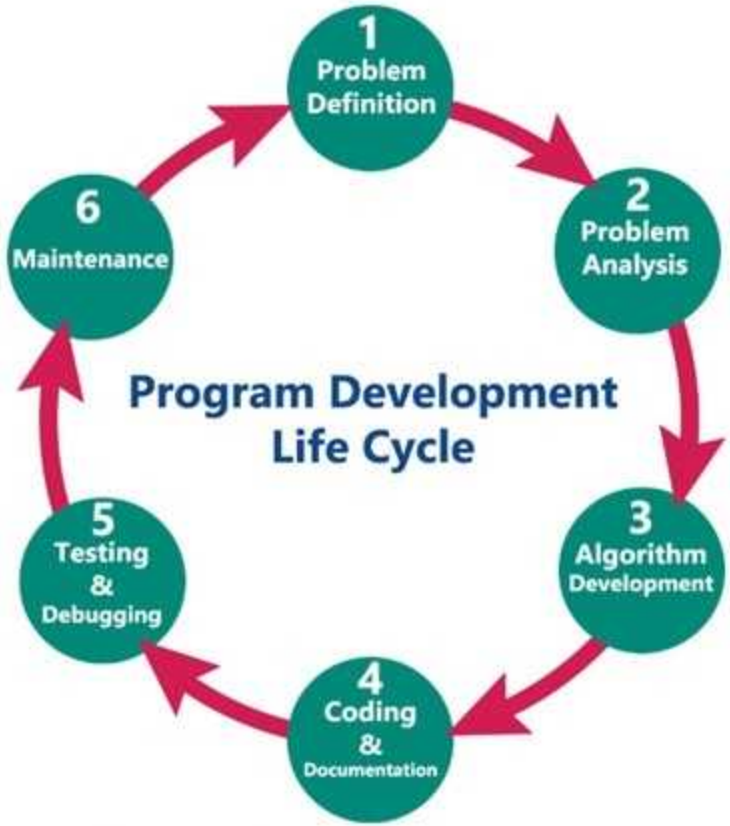
The spiral model

- COMMUNICATION
  - Tasks required are establish effective communication between developer
- PLANNING
  - Estimation
  - Scheduling
  - Risk analysis
- MODELING
  - Analysis
  - Design
- CONSTRUCTION
  - Code
  - Test
- DEPLOYMENT
  - Delivery
  - Feedback



Program development Life cycle

- When we want to develop a program using any programming language, we follow a sequence of steps.
- These steps are called phases in program development. The program development life cycle is a set of steps or phases that are used to develop a program in any programming language.



Program development Life cycle

1. Problem Definition

- In this phase, we define the problem statement and we decide the boundaries of the problem. In this phase we need to understand the problem statement, what is our requirement, what should be the output of the problem solution. These are defined in this first phase of the program development life cycle.

2. Problem Analysis

- In phase 2, we determine the requirements like variables, functions, etc. to solve the problem. That means we gather the required resources to solve the problem defined in the problem definition phase. We also determine the bounds of the solution.

3. Algorithm Development

- During this phase, we develop a step by step procedure to solve the problem using the specification given in the previous phase. This phase is very important for program development. That means we write the solution in step by step statements.

4. Coding & Documentation

- This phase uses a programming language to write or implement actual programming instructions for the steps defined in the previous phase. In this phase, we construct actual program. That means we write the program to solve the given problem using programming languages like C, C++, Java etc.,

Program development Life cycle

5. Testing & Debugging

- During this phase, we check whether the code written in previous step is solving the specified problem or not. That means we test the program whether it is solving the problem for various input data values or not. We also test that whether it is providing the desired output or not.

6. Maintenance

- During this phase, the program is actively used by the users. If any enhancements found in this phase, all the phases are to be repeated again to make the enhancements. That means in this phase, the solution (program) is used by the end user. If the user encounters any problem or wants any enhancement, then we need to repeat all the phases from the starting, so that the encountered problem is solved or enhancement is added.

Different programming paradigm

- In science and philosophy, a paradigm is a distinct set of concepts or thought patterns, including theories, research methods, postulates, and standards for what constitutes legitimate contributions to a field.
- A programming paradigm is a fundamental style of computer programming. Paradigms differ in the concepts and methods used to represent the elements of a program (such as objects, functions, variables, constraints). And also steps that comprise a computation (such as assignations, evaluation, continuations, data flows).
- Different approaches to programming have developed over time.
- The concept of a "programming paradigm" as such dates at least to 1978.
- The lowest-level programming paradigms are machine code.
- In the 1960s, assembly languages were developed followed by development of procedural languages.
- Following the widespread use of procedural languages, object-oriented programming (OOP) languages were created and followed by many more.

Imperative programming paradigm

- Imperative programming is a programming paradigm that uses statements that change a program's state.
- In much the same way that the imperative mood in natural languages expresses commands.
- An imperative program consists of commands for the computer to perform.
- Imperative programs describe the details of HOW the results are to be obtained.
- HOW means describing the Inputs and describing how the Outputs are produced.
- Examples are: C, C++, Java, PHP, Python, Ruby etc.

DECLARATIVE PROGRAMMING PARADIGM

- Declarative programming is a programming paradigm—a style of building the structure and elements of computer programs—that expresses the logic of a computation without describing its control flow.
- Declarative programming focuses on what the program should accomplish.
- Declarative programming often considers programs as theories of a formal logic, and computations as deductions in that logic space.
- Examples are: SQL, XSQL (XMLSQL) etc.

FUNCTIONAL PROGRAMMING PARADIGM

- Functional programming is a subset of declarative programming.
- Programs written using this paradigm use functions, blocks of code intended to behave like mathematical functions.
- Functional languages discourage changes in the value of variables through assignment, making a great deal of use of recursion instead. Examples are : F#, Haskell, Lisp, Python, Ruby, JavaScript etc.

OBJECT ORIENTED PROGRAMMING PARADIGM

- Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which may contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods.
- There is significant diversity of OOP languages, but the most popular ones are class-based, meaning that objects are instances of classes, which typically also determine their type.
- In OOP, computer programs are designed by making them out of objects.
- Examples are: C++, C#, Java, PHP, Python.

Multi-paradigm

- A multi-paradigm programming language is a programming language that supports more than one programming paradigm.
- The design goal of such languages is to allow programmers to use the most suitable programming style and associated language constructs for a given job.
- Languages such as C++, Java, Python are multiparadigm programming languages that support object-oriented programming to a greater or lesser degree, typically in combination with imperative, procedural programming.

Paradigm Summary

Paradigm Description Examples

Imperative

- Programs as statements that directly change computed state (datafields).
- C, C++, Java, PHP, Python, Ruby.

Functional

- Treats computation as the evaluation of mathematical functions avoiding state.
  C++, Lisp, Python, JavaS cript

Object-oriented

- Treats datafields as object s manipulated through predefined methods only
- C++, C#., Java, PHP, Python .

Declarative

- Defines program logic, but not detailed control flow
- SQL, CSS

System Design tools

- System design tools are shapes, symbols, tables of human readable and understandable documents which represent flow of system, components of system, conditions and actions to be taken while implementing into the code.
- it is the intermediate stage, which helps human-readable requirements to be transformed into actual code and can be used to various types computer software to design.

1. Data Flow Diagram

Data flow diagram is graphical representation of flow of data in an information system. It can depict incoming data flow, outgoing data flow and stored data. The DFD does not mention anything about how data flows through the system.

- Entities - Entities are source and destination of information data. Entities are represented by a rectangle 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.

System Design tools

2.Decision Table:

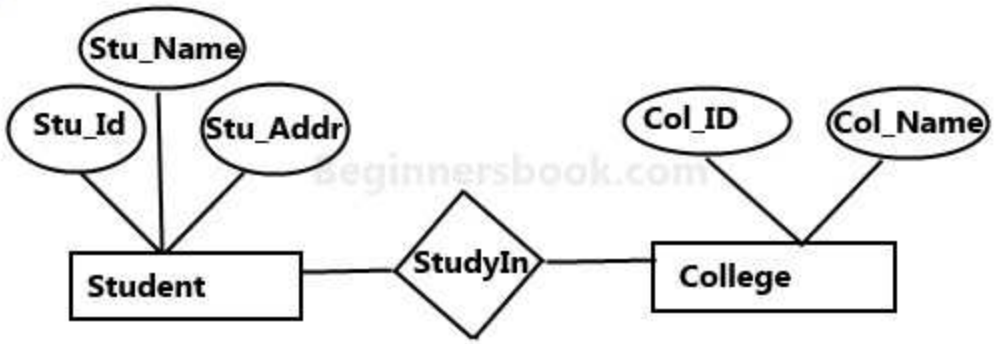
- A Decision table represents conditions and the respective actions to be taken to address them, in a structured tabular format.
- It is a powerful tool to debug and prevent errors. It helps group similar information into a single table and then by combining tables it delivers easy and convenient decision-making.

Printer troubleshooter

		Rules							
Conditions	Printer prints	No	No	No	No	Yes	Yes	Yes	Yes
	A red light is flashing	Yes	Yes	No	No	Yes	Yes	No	No
	Printer is recognized by computer	No	Yes	No	Yes	No	Yes	No	Yes
Actions	Check the power cable			✓					—
	Check the printer-computer cable	✓		✓					—
	Ensure printer software is installed	✓		✓		✓		✓	—
	Check/replace ink	✓	✓				✓		—
	Check for paper jam		✓		✓				—

System Design tools

- 2. ER Diagram: It is a visual representation of data that describes how data is related to each other. In ER Model, we disintegrate data into entities, attributes and setup relationships between entities, all this can be represented visually using the ER diagram.
- An **entity** is an object, things, or event that exists and is distinguishable from other objects.
  - Example: specific person, company, event, plant
- Entities have *attributes (it is property of entity which means it define entity)*
  - Example: people have *names* and *addresses*
- A relationship is an association among several entities



Sample E-R Diagram

System Design tools

3. Pseudocode: It is an artificial and informal language that helps programmers develop algorithms. Pseudocode is a "text-based" detail (algorithmic) design tool.

Examples:

If student's grade is greater than or equal to 60
    Print "passed"
      else
   Print "failed"

4. algorithm

A programming algorithm is a computer procedure (finite set)and tells your computer precisely what steps solve a problem or reach a goal.

Step 1: Start

Step 2: Create a variable to receive the user's email address

Step 3: Clear the variable in case it's not empty

Step 4: Ask the user for an email address

Step 5: Store the response in the variable

Step 6: Check the stored response to see if it is a valid email address

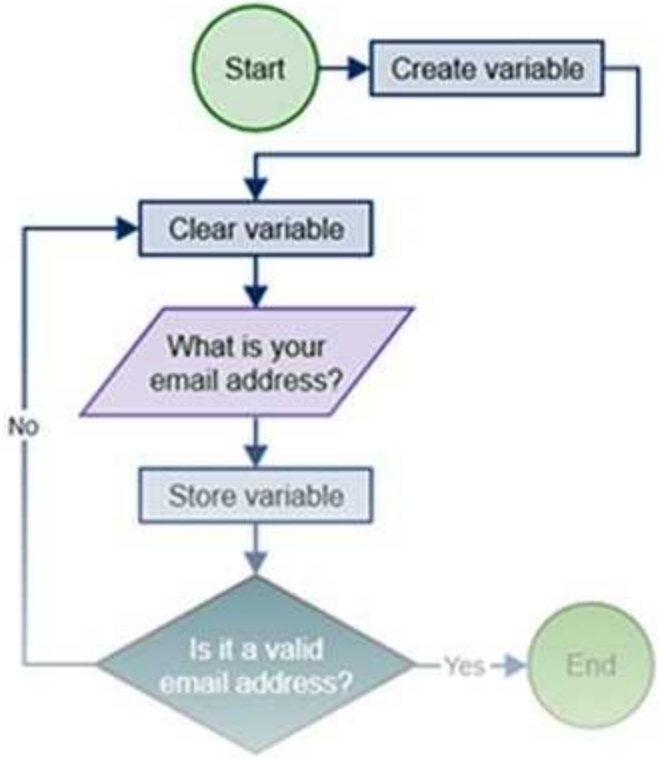
Step 7: Not valid? Go back to Step 3.

Step 8: End

System Design tools

5. Flow chart:

A flowchart is a type of diagram that represents an algorithm, workflow or process. The flowchart shows the steps as boxes of various kinds, and their order by connecting the boxes with arrows. This diagrammatic representation illustrates a solution model to a given problem.



Questions

1. What is programming language? Explain different types of programming language?
2. Discuss the different advantages and disadvantage of different programming language.
3. What do you mean by language processor? Explain different types of language processor.
4. Explain the features of good program.
5. What do you mean by an error? Explain different types of error.
6. What do you understand by software process model? Explain three software process model with its strength and weakness.
7. What is SDLC? Explain each steps in detail.
8. What is programming paradigm? Explain different types of paradigm.
9. Why system design tools are necessary in programming? Explain three different system design tools.