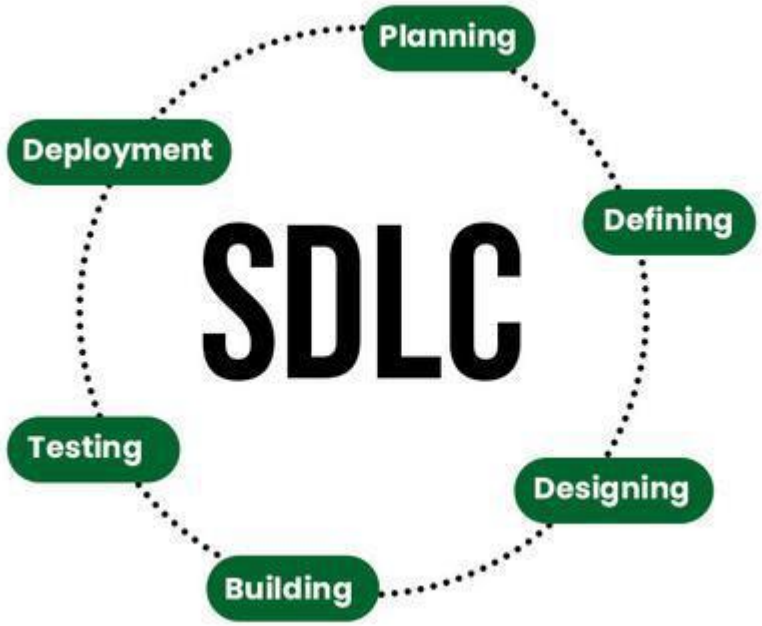
Developing Product

Lecture-8

Software Development Life Cycle (SDLC)

Software development life cycle (SDLC) is a structured process that is used to design, develop, and test good-quality software. SDLC, or software development life cycle, is a methodology that defines the entire procedure of software development step-by-step.



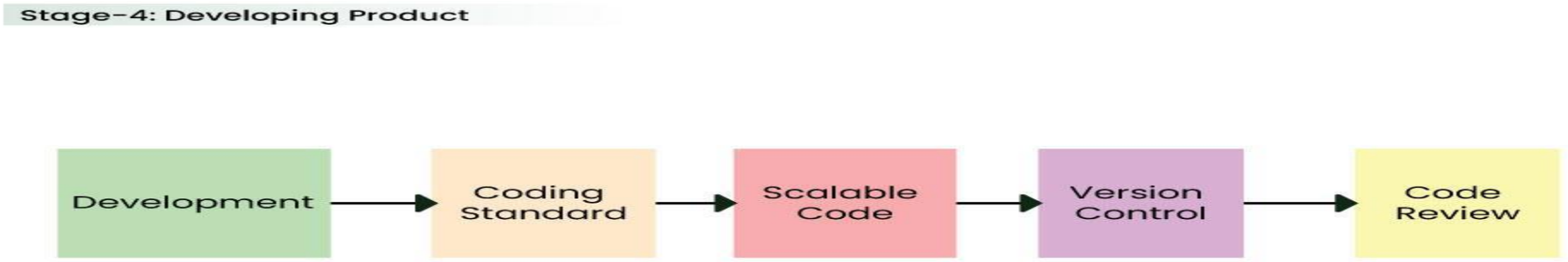
Software Development Life Cycle (SDLC)

- Stage-1: Planning and Requirement Analysis
- Stage-2: Defining Requirements
- Stage-3: Designing Architecture
- Stage-4: Developing Product
- Stage-5: Product Testing and Integration
- Stage-6: Deployment and Maintenance of Products

https://media.geeksforgeeks.org/wp-content/uploads/20231220112830/6-Stages-of-Software-Development-Life-Cycle.jpg

Stage-4: Developing Product

- At this stage, the fundamental development of the product starts. For this, developers use a specific programming code as per the design in the DDS. Hence, it is important for the coders to follow the protocols set by the association. Conventional programming tools like compilers, interpreters, debuggers, etc. are also put into use at this stage. Some popular languages like C/C++, Python, Java, etc. are put into use as per the software regulations.



Coding

- More precisely, code is a rule-based set of instructions written in a language that your computer understands. And coding is the act of writing, testing, and debugging code.

Goals of Coding

- **To translate the design of system into a computer language format:** The coding is the process of transforming the design of a system into a computer language format, which can be executed by a computer and that perform tasks as specified by the design of operation during the design phase.
- **To reduce the cost of later phases:** The cost of testing and maintenance can be significantly reduced with efficient coding.
- **Making the program more readable:** Program should be easy to read and understand. It increases code understanding having readability and understandability as a clear objective of the coding activity can itself help in producing more maintainable software.

Characteristics of Programming Language

- **Readability:** A good high-level language will allow programs to be written in some methods that resemble a quite-English description of the underlying functions. The coding may be done in an essentially self-documenting way.
- **Portability:** High-level languages, being virtually machine-independent, should be easy to develop portable software.
- **Generality:** Most high-level languages allow the writing of a vast collection of programs, thus relieving the programmer of the need to develop into an expert in many diverse languages.
- **Brevity:** Language should have the ability to implement the algorithm with less amount of code. Programs mean in high-level languages are often significantly shorter than their low-level equivalents.

Characteristics of Programming Language

- **Error checking:** A programmer is likely to make many errors in the development of a computer program. Many high-level languages invoke a lot of bugs checking both at compile-time and run-time.
- **Cost:** The ultimate cost of a programming language is a task of many of its characteristics.
- **Quick translation:** It should permit quick translation.
- **Efficiency:** It should authorize the creation of an efficient object code.

Parameter	High-Level Language	Low-Level Language
Basic	These are programmer-friendly languages that are manageable, easy to understand, debug, and widely used in today's times.	These are machine-friendly languages that are very difficult to understand by human beings but easy to interpret by machines.
Ease of Execution	These are very easy to execute.	These are very difficult to execute.
Process of Translation	High-level languages require the use of a compiler or an interpreter for their translation into the machine code.	Low-level language requires an assembler for directly translating the instructions of the machine language.
Efficiency of Memory	These languages have a very low memory efficiency. It means that they consume more memory than any low-level language.	These languages have a very high memory efficiency. It means that they consume less energy as compared to any high-level language.
Portability	These are portable from any one device to another.	A user cannot port these from one device to another.
Comprehensibility	High-level languages are human-friendly. They are, thus, very easy to understand and learn by any programmer.	Low-level languages are machine-friendly. They are, thus, very difficult to understand and learn by any human.
Dependency on Machines	High-level languages do not depend on machines.	Low-level languages are machine-dependent and thus very difficult to understand by a normal user.
Debugging	It is very easy to debug these languages.	A programmer cannot easily debug these languages.
Maintenance	High-level languages have a simple and comprehensive maintenance technique.	It is quite complex to maintain any low-level language.
Usage	High-level languages are very common and widely used for programming in today's times.	Low-level languages are not very common nowadays for programming.
Speed of Execution	High-level languages take more time for execution as compared to low-level languages because these require a translation program.	The translation speed of low-level languages is very high.

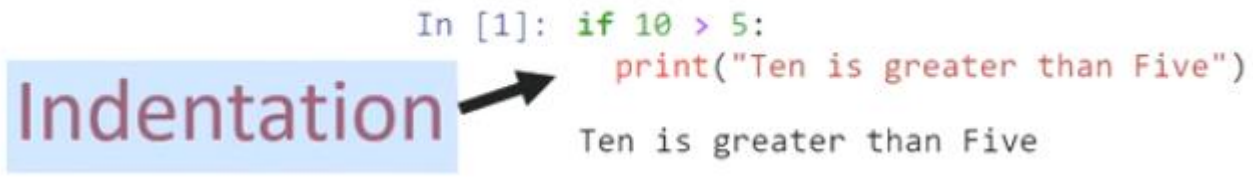
Characteristics of Programming Language

- **Modularity:** It is desirable that programs can be developed in the language as several separately compiled modules, with the appropriate structure for ensuring self-consistency among these modules.
- **Widely available:** Language should be widely available, and it should be feasible to provide translators for all the major machines and all the primary operating systems.

Coding Standards

1.Indentation: Proper and consistent indentation is essential in producing easy to read and maintainable programs.Indentation should be used to:

1. Emphasize the body of a control structure such as a loop or a select statement.
2. Emphasize the body of a conditional statement
3. Emphasize a new scope block



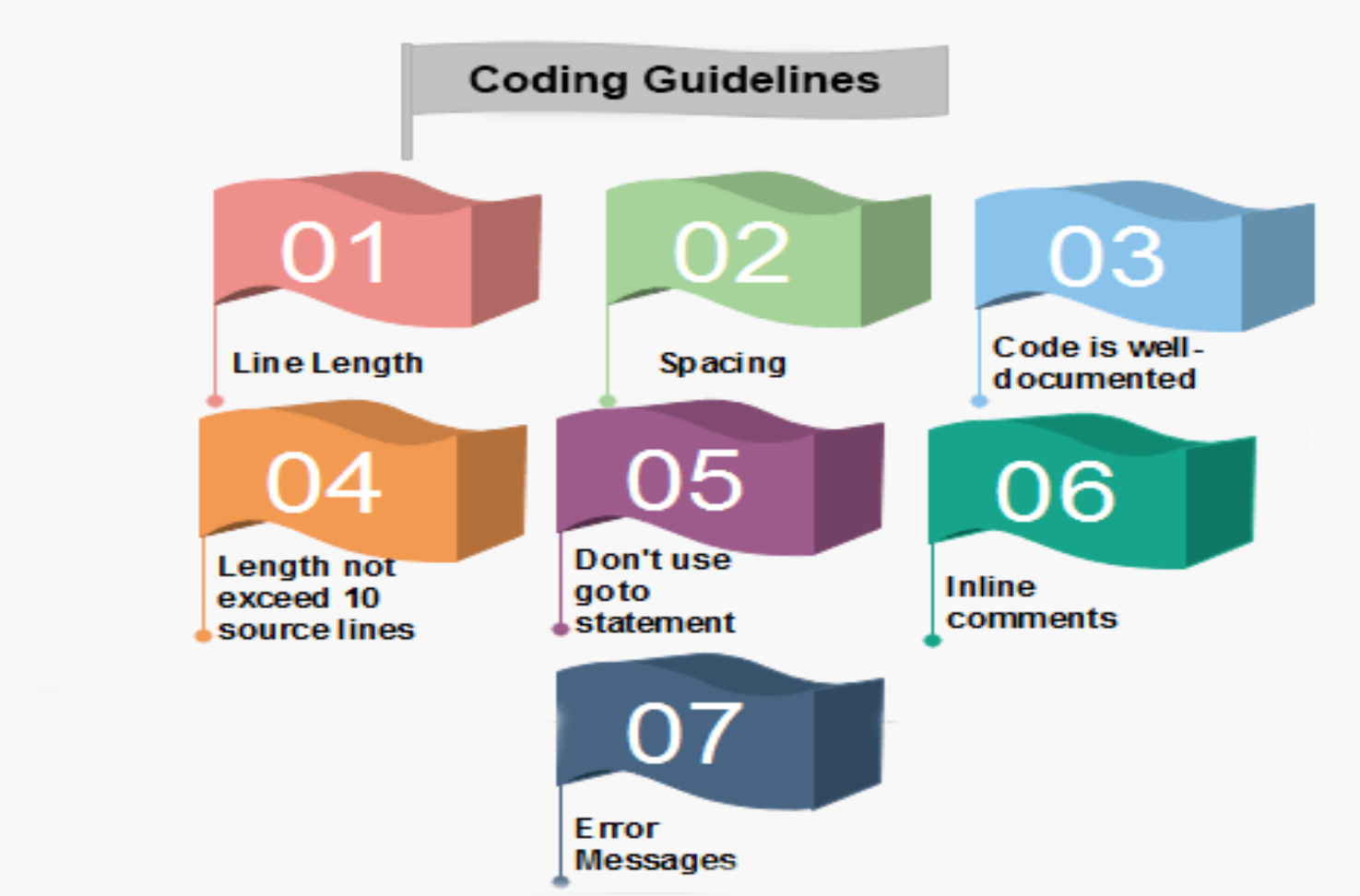
2.Inline comments: Inline comments analyze the functioning of the subroutine, or key aspects of the algorithm shall be frequently used.

3.Rules for limiting the use of global: These rules file what types of data can be declared global and what cannot.

Coding Standards

- **Structured Programming:** Structured (or Modular) Programming methods shall be used. "GOTO" statements shall not be used as they lead to "spaghetti" code, which is hard to read and maintain, except as outlined line in the FORTRAN Standards and Guidelines.
- **Naming conventions for global variables, local variables, and constant identifiers:** A possible naming convention can be that global variable names always begin with a capital letter, local variable names are made of small letters, and constant names are always capital letters.
- **Error return conventions and exception handling system:** Different functions in a program report the way error conditions are handled should be standard within an organization. For example, different tasks while encountering an error condition should either return a 0 or 1 consistently.

Coding Guidelines



Coding Guidelines

- **1. Line Length:** It is considered a good practice to keep the length of source code lines at or below 80 characters. Lines longer than this may not be visible properly on some terminals and tools. Some printers will truncate lines longer than 80 columns.
- **2. Spacing:** The appropriate use of spaces within a line of code can improve readability.
- **Example:**
- **Bad:**
```
cost=price+(price*sales_tax)
fprintf(stdout ,"The total cost is %5.2f\n",cost);
```
- **Better:**
```
cost = price + ( price * sales_tax )
fprintf (stdout,"The total cost is %5.2f\n",cost);
```

Coding Guidelines

- **3. The code should be well-documented:** As a rule of thumb, there must be at least one comment line on the average for every three-source line.
- **4. The length of any function should not exceed 10 source lines:** A very lengthy function is generally very difficult to understand as it possibly carries out many various functions. For the same reason, lengthy functions are possible to have a disproportionately larger number of bugs.
- **5. Do not use goto statements:** Use of goto statements makes a program unstructured and very tough to understand.
- **6. Inline Comments:** Inline comments promote readability.
- **7. Error Messages:** Error handling is an essential aspect of computer programming. This does not only include adding the necessary logic to test for and handle errors but also involves making error messages meaningful.

Good Code

```
// Select the button element and add a click event listener
const button = document.querySelector("#my-button");
button.addEventListener("click", handleClick);

// Define the click event handler function
function handleClick(event) {
  event.preventDefault();
  const form = document.querySelector("#my-form");
  form.submit();
}
```

The above code is an example of good, clean, readable code focused on web development in JavaScript. It selects a button element and adds a click event listener to it. The click event handler function prevents the default behavior of the button, selects a form element, and submits it. The code uses descriptive variable names, is well-structured, and easy to read and understand. The use of comments in the code helps to explain what the code is doing and makes it more maintainable and easier to modify.

Good Code

```
// Gets the sum of all even numbers in an array
function sumOfEvens(arr) {
  let sum = 0;
  for (let i = 0; i < arr.length; i++) {
    if (arr[i] % 2 === 0) {
      sum += arr[i];
    }
  }
  return sum;
}

// Example usage
const myArr = [1, 2, 3, 4, 5, 6];
const result = sumOfEvens(myArr);
console.log(result);
```

If you're building software to handle a specific problem, but as other details such as your data set change, your code slows significantly, then you have bad code. But if your software is able to solve the problem at a similar speed, no matter how much your data changes, then you have written some good code.
Scalable code is code that has been written in such a way that it doesn't get easily overwhelmed. It's efficient and solves problems or performs tasks as quickly and as frequently as they pop up.

The above code defines a function sumOfEvens that takes an array of numbers and returns the sum of all even numbers in the array. The function is well-structured with a clear name and variable names that indicate their purpose. It uses a for loop to iterate through the array and add even numbers to a sum variable. The function is scalable as it can handle an array of any length, and the code is commented to explain what each line is doing.

Poor code

```
let btn = document.getElementById("my-button");
btn.onclick=function(event) {
  let frm = document.getElementById("my-form");
  frm.submit();
}
```

This code is an example of poorly structured JavaScript code. It uses an inline event handler to submit a form when a button is clicked. The variable names are not descriptive, and the code is not well-structured, which can make it harder to read and understand. There are no comments to explain what the code is doing, which can make it more difficult to modify and maintain. Additionally, the use of an inline event handler can make the code more difficult to debug.