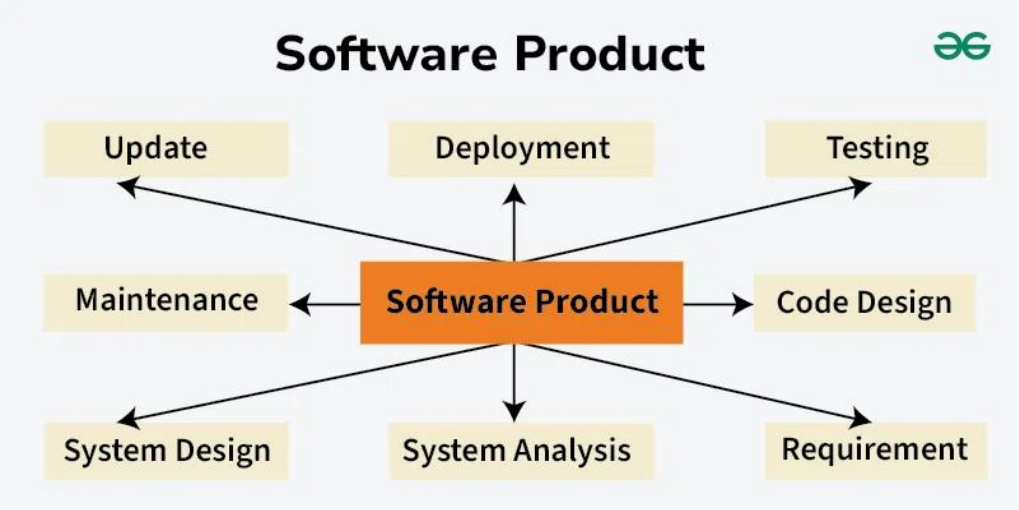
Software Engineering Introduction

Software Engineering is a systematic, disciplined, quantifiable study and approach to the design, development, operation, and maintenance of a software system. These article help you understand the basics of software engineering. This Introduction part covers the topic like Basics of Software and Software engineering, What is the need of Software Engineering etc.

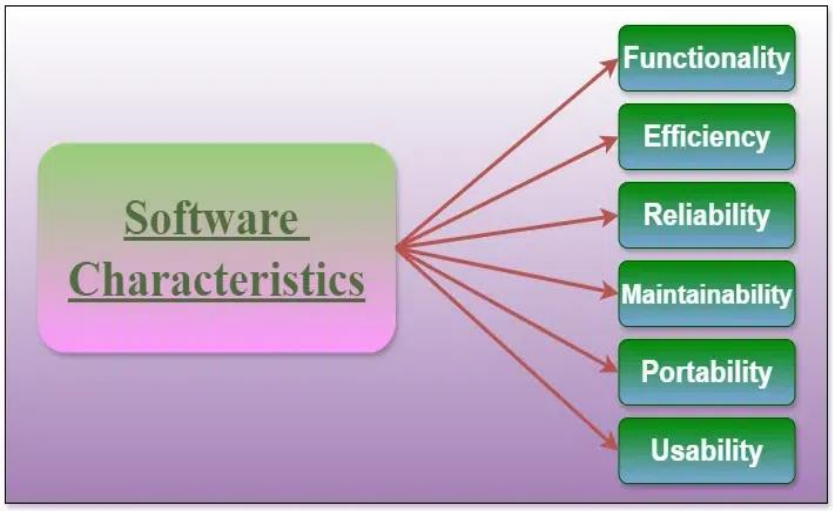
Software Engineering

Software Engineering is the process of designing, developing, testing, and maintaining software. It is a systematic and disciplined approach to software development that aims to create high-quality, reliable, and maintainable software. Software engineering includes a variety of techniques, tools, and methodologies, including requirements analysis, design, testing, and maintenance.



Components of Software Characteristics

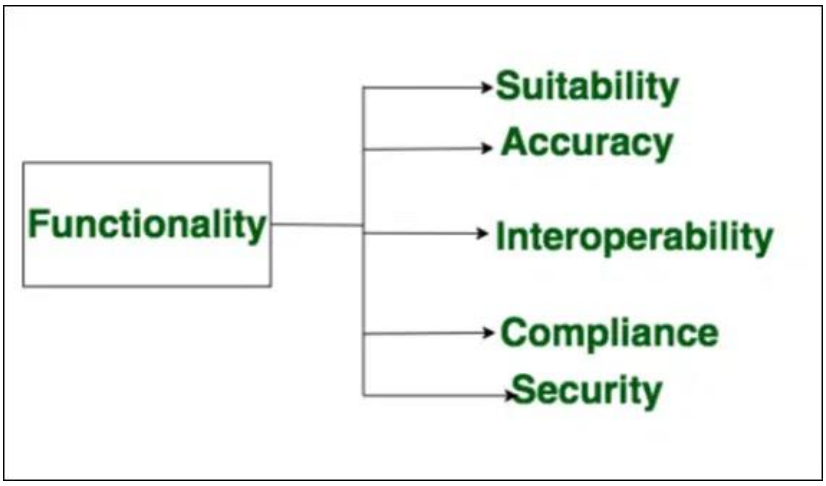
There are 6 components of Software Characteristics are discussed here. We will discuss each one of them in detail.



Functionality

Functionality refers to the set of features and capabilities that a software program or system provides to its users. It is one of the most important characteristics of software, as it determines the usefulness of the software for the intended purpose. Examples of functionality in software include:

- Data storage and retrieval
- Data processing and manipulation
- User interface and navigation
- Communication and networking
- Security and access control
- Reporting and visualization
- Automation and scripting



Reliability:

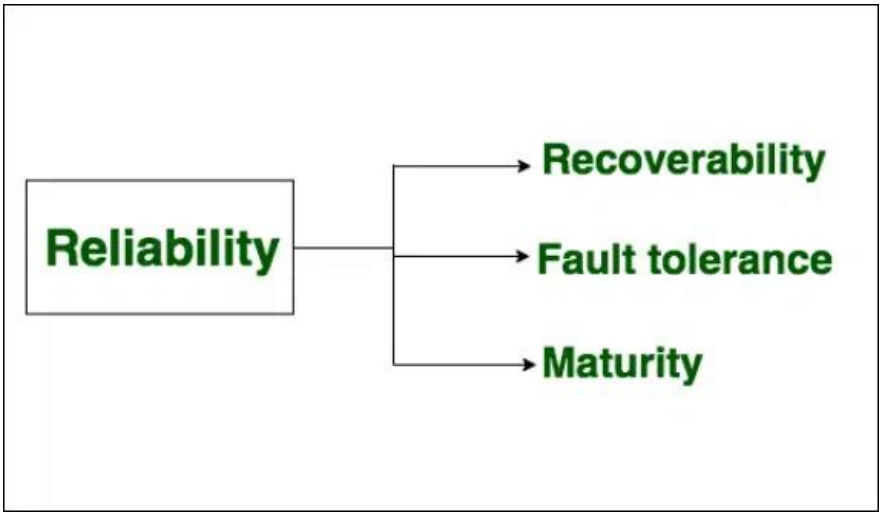
A set of attributes that bears on the capability of software to maintain its level of performance under the given condition for a stated period of time.
Reliability is a characteristic of software that refers to its ability to perform its intended functions correctly and consistently over time. Reliability is an important aspect of software quality, as it helps ensure that the software will work correctly and not fail unexpectedly.

Examples of factors that can affect the reliability of software include:

1. Bugs and errors in the code
2. Lack of testing and validation
3. Poorly designed algorithms and data structures
4. Inadequate error handling and recovery
5. Incompatibilities with other software or hardware

To improve the reliability of software, various techniques, and methodologies can be used, such as testing and validation, formal verification, and fault tolerance.
Software is considered reliable when the probability of it failing is low and it is able to recover from the failure quickly, if any.

Required functions are:



Efficiency:

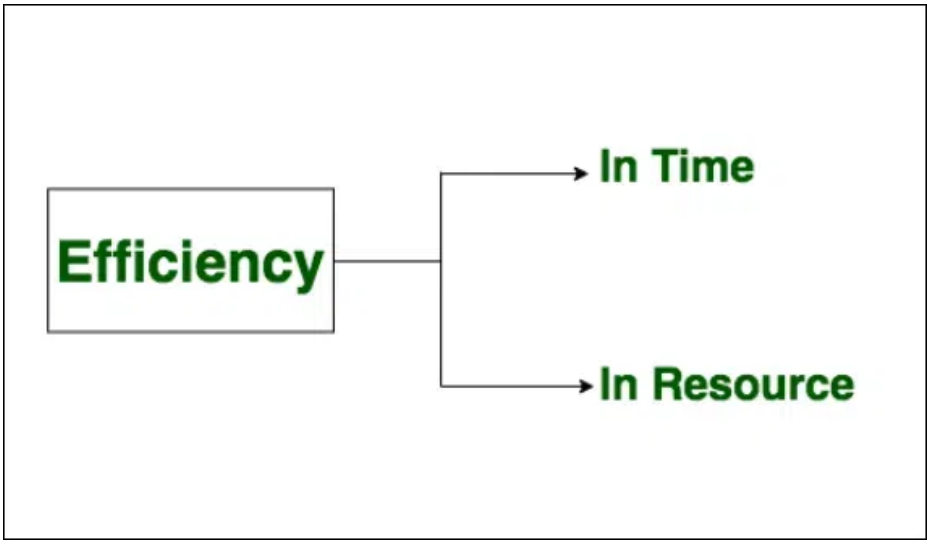
Efficiency is a characteristic of software that refers to its ability to use resources such as memory, processing power, and network bandwidth in an optimal way. High efficiency means that a software program can perform its intended functions quickly and with minimal use of resources, while low efficiency means that a software program may be slow or consume excessive resources.

Examples of factors that can affect the efficiency of the software include:

1. Poorly designed algorithms and data structures
2. Inefficient use of memory and processing power
3. High network latency or bandwidth usage
4. Unnecessary processing or computation
5. Unoptimized code

To improve the efficiency of software, various techniques, and methodologies can be used, such as performance analysis, optimization, and profiling.

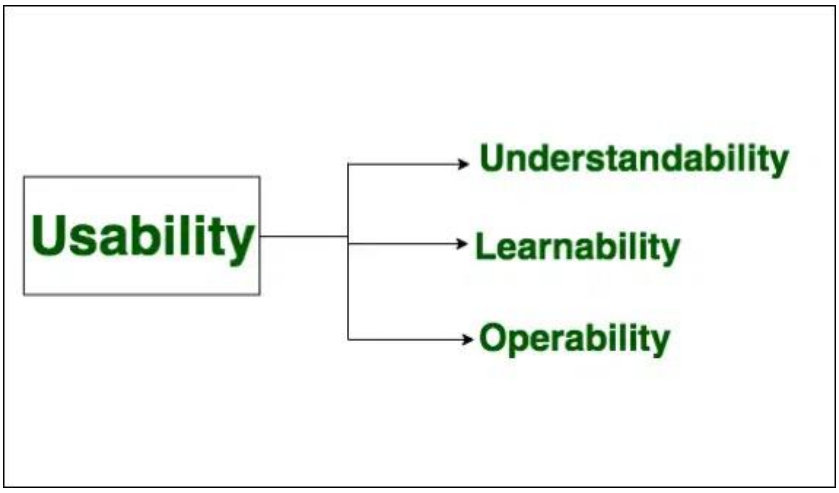
Required functions are:



Usability:

It refers to the extent to which the software can be used with ease. the amount of effort or time required to learn how to use the software.

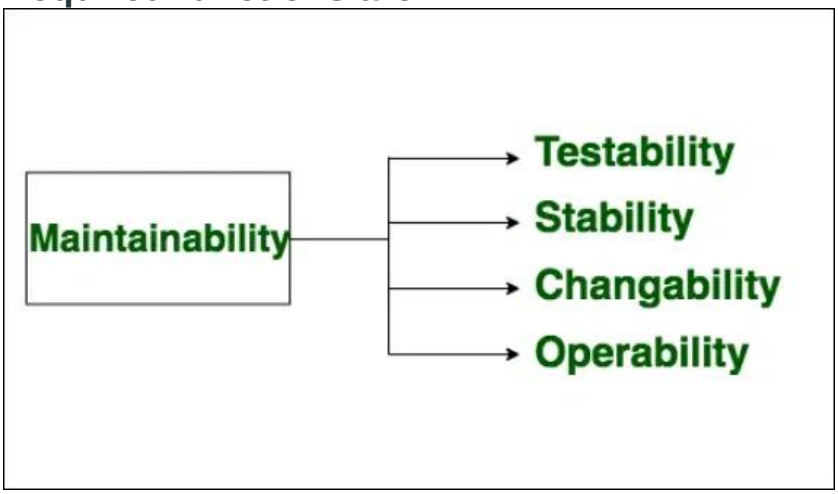
Required functions are:



Maintainability:

It refers to the ease with which modifications can be made in a software system to extend its functionality, improve its performance, or correct errors.

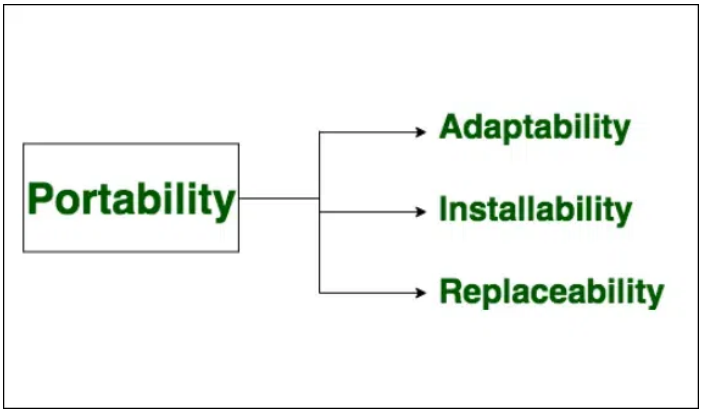
Required functions are:



Portability:

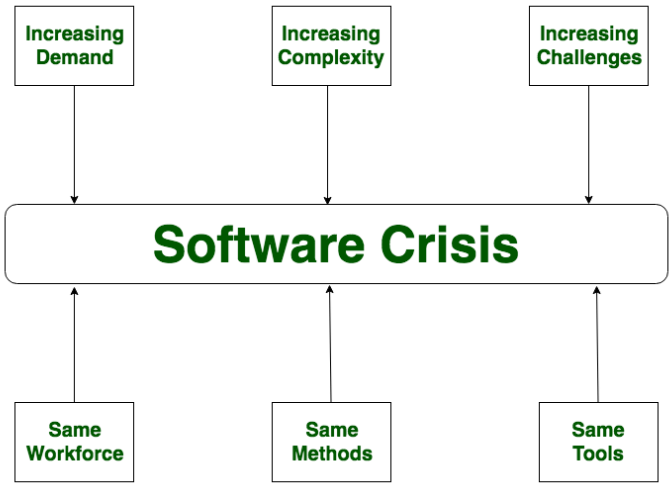
A set of attributes that bears on the ability of software to be transferred from one environment to another, without minimum changes.

Required functions are:



Software Crisis

The term “software crisis” refers to the numerous challenges and difficulties faced by the software industry during the 1960s and 1970s. It became clear that old methods of developing software couldn’t keep up with the growing complexity and demands of new projects. This led to high costs, delays, and poor-quality software.



Causes of Software Crisis

Following are the causes of Software Crisis:

- The cost of owning and maintaining software was as expensive as developing the software.
- At that time Projects were running overtime.
- At that time Software was very inefficient.

- The quality of the software was low quality.
- Software often did not meet user requirements.
- The average software project overshoots its schedule by half.
- At that time Software was never delivered.

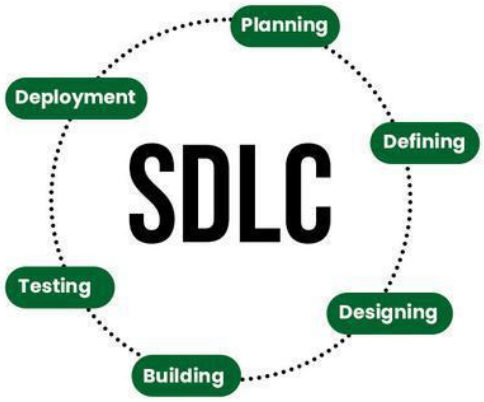
solution of Software Crisis:

There is no single solution to the crisis. One possible solution to a software crisis is software engineering because software engineering is a systematic, disciplined, and quantifiable approach. For preventing software crises, there are some guidelines:

- Reduction in software over budget.
- The quality of the software must be high.
- Less time is needed for a software project.
- Experienced and skilled people working on the software project.
- Software must be delivered.
- Software must meet user requirements.

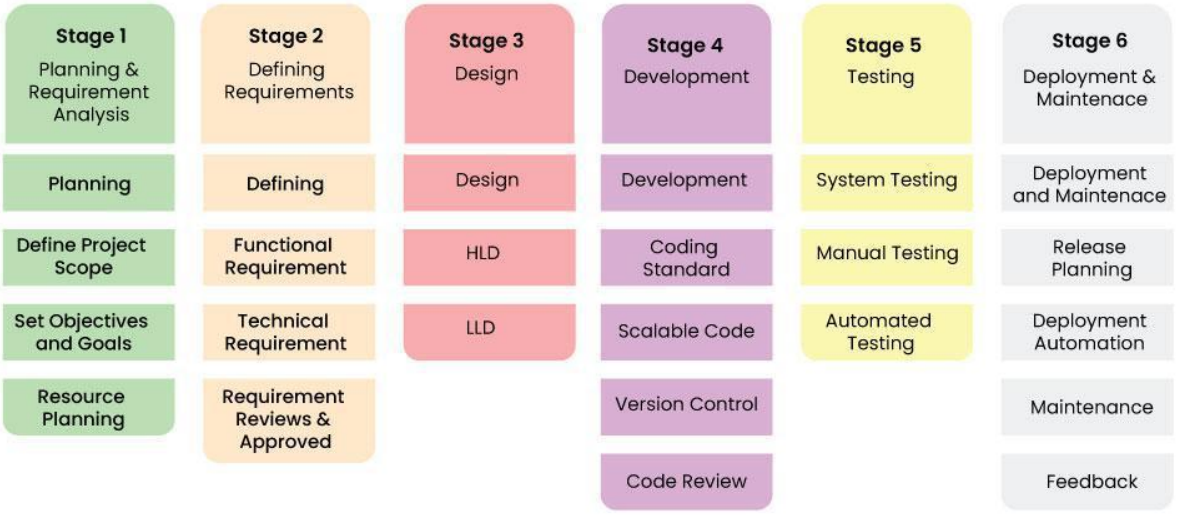
Software Development Life Cycle (SDLC)

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



Stages of the Software Development Life Cycle

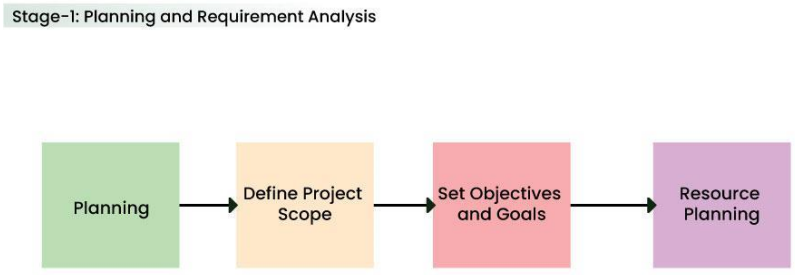
SDLC specifies the task(s) to be performed at various stages by a software engineer or developer.



6 Stages of Software Development Life Cycle

Stage-1: Planning and Requirement Analysis

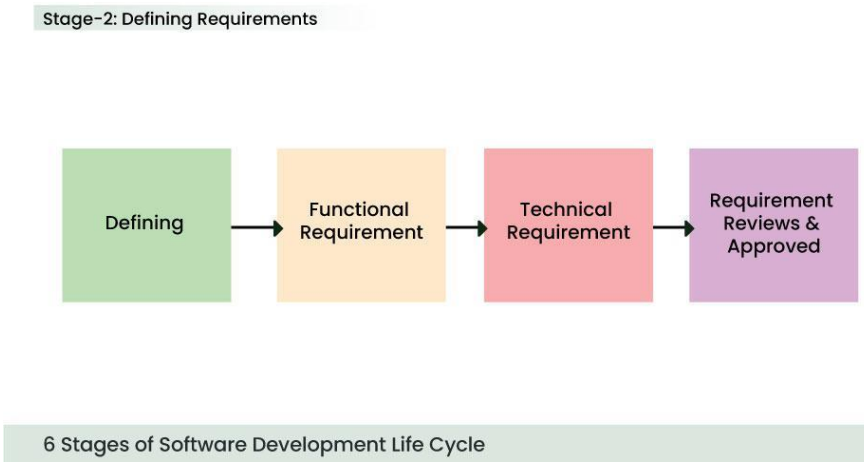
Planning is a crucial step in everything, just as in. In this same stage, is also performed by the developers of the organization. This is attained from customer inputs, and sales department/market surveys.



6 Stages of Software Development Life Cycle

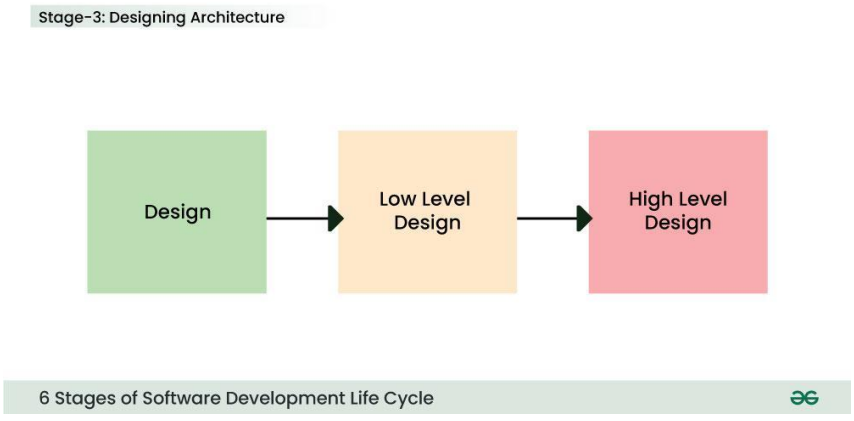
Stage-2: Defining Requirements

In this stage, all the requirements for the target software are specified. These requirements get approval from customers, market analysts, and stakeholders.
This is fulfilled by utilizing SRS (Software Requirement Specification).



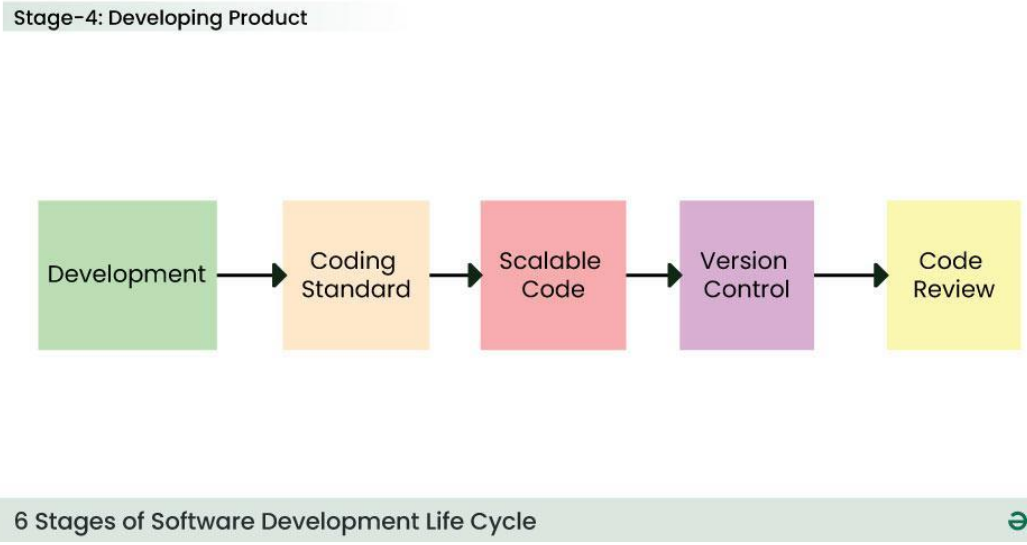
Stage-3: Designing Architecture

SRS is a reference for software designers to come up with the best architecture for the software. Hence, with the requirements defined in SRS, multiple designs for the product architecture are present in the Design Document Specification (DDS).



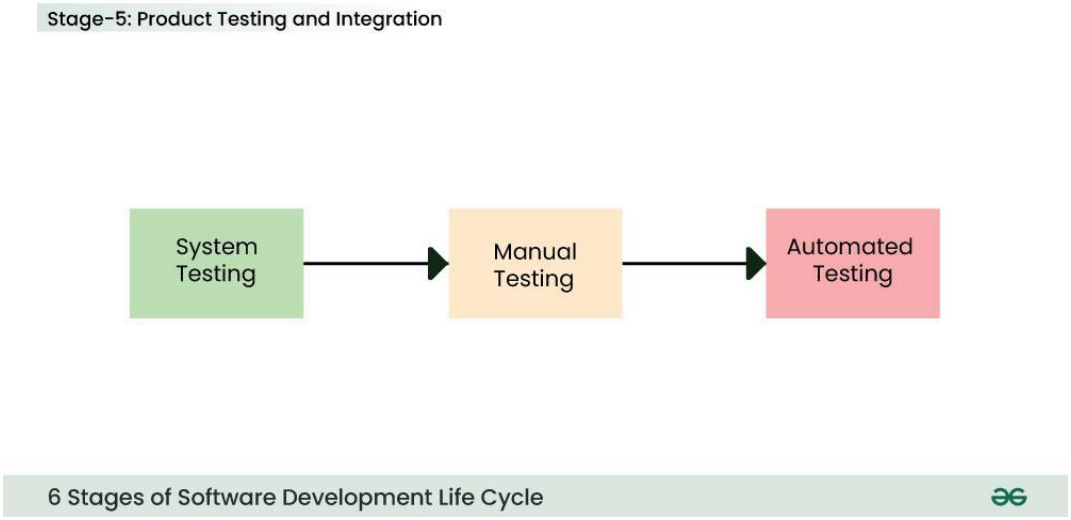
Stage-4: Developing Product

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



Stage-5: Product Testing and Integration

After the development of the product, testing of the software is necessary to ensure its smooth execution. Although, minimal testing is conducted at every stage of SDLC. Therefore, at this stage, all the probable flaws are tracked, fixed, and retested. This ensures that the product confronts the quality requirements of SRS.



Stage-6: Deployment and Maintenance of Products

After detailed testing, the conclusive product is released in phases as per the organization's strategy. Then it is tested in a real industrial environment.