# **What is Software?**

Software: s/w is the language of computer.

* Is a collection of computer programs and related data that provide the instructions for telling a computer what to do and how to do it.
* Just like human language.

**What is Software Engineering?**

Software: a program or set programs containing instructions which provide desired functionality.

Engineering: Process of designing and building something that ensure particular purpose.

Software Engineering: “ Software engineering is the art of developing quality software on time and within budget”.

**SOFTWARE ENGINEERING** is a systematic approach to the design, development, operation, and maintenance of a software system.

A naive view: **Problem Specification Final Program**

But….

* Where did the specification come from?
* How do you know the specification corresponds to the user’s needs?
* How did you decide how to structure your program?
* How do you know the program actually meets the specification?
* How do you know your program will always work correctly?
* What do you do if the users’ needs change?
* How do you divide tasks up if you have more than a one-person team?

“Software engineering is the multi-person construction of multi-version software”. -PARNAS

* TEAM-WORK
* Scale issue (“program well” is not enough) + Communication Issue
* Successful software systems must evolve or perish
* Change is the norm, not the exception

“ Software engineering is different from other engineering disciplines “ - SUMMERVILLE

* Not constrained by physical laws
* Limit = Human mind
* It is constrained by political forces
* Balancing stake-holders

# **EXPLAIN TYPES OF SOFTWARE**

There is 3 types of software.

1. System software
2. Application software
3. Programming language
4. SYSTEM SOFTWARE

* Provides the basic functions for computer usage and helps to run the computer hardware and system.
* Is the s/w used by the computer to translate inputs from various sources into a language which a machine can understand.
* Basically OS coordinates the different hardware components of a computer.
* Ex. Linux, window, macOS, android, iOS

1. APPLICATION SOFTWARE

* Is the general designation of computer programs for performing user tasks.
* Types of application s/w

1. Mobile app:

* Application that run on mobile
* Ex. Instagram, Facebook , etc..

1. Desktop app:

* That run stand-alone in a desktop or laptop computer.
* Ex. Microsoft office suite which includes Word , Excel ,and PowerPoint.
* Ex. Outlook for email, and Firefox, google chrome, Mozilla are the web server.
* Anti-virus is an application and so is the media player.

1. PROGRAMMING SOFTWARE:

* Is the process designing, writing , testing , debugging , and maintaining the source code of computer programs.
* This s/w pawritten in a programming language.
* The purpose of programming is to create a program that exhibits a certain desired behaviour.
* EX. C++, java, simlab , php, python and visual basic.

# **WHAT IS SDLC?**

* SDLC : SOFTWARE DEVELOPMENT LIFE CYCLE
* SDLC is a structure imposed on the development of a software product that defines the process for planning , implementation, testing, documentation, deployment, and ongoing maintenance and support. There are a number if different development models.
* A Software Development Life Cycle is essentially a series or phases, that provide a model for the development and lifecycle management of an application or piece of software.
* The methodology within the SDLC process can vary across industries and organizations, but standards such as ISO/IEC 12207 represent processes that establish a lifecycle for software, and provide a mode for the development, acquisition ,and configurations of software systems.

# **PHASES OF SDLC**

REQUIREMENTS GATHERING

* Establish customer needs

ANALYSIS

* Model and specify the requirement –“what”

DESIGN

* Model and specify a solution –“why”

IMPLEMANTATION

* Construct a solution in software

TESTING

* Validate the solution against the requirements

MAINTENANCE

* Repair defects and adapt the solution to the new requirements

REQUIREMENTS GATHERING

* Features
* Usage scenarios
* Although requirements may be documented in written form, they may be incomplete, unambiguous, or even incorrect.
* Requirements will changes!
* Inadequately captures or expressed in the first place
* User and business needs change during the project
* User and business needs change during the project
* Validation is needed throughout the software lifecycle, not only when the “final system” is delivered.
* Build constant feedback into the project plan
* Plan for change
* Early prototyping [ e.g., UI] can help clarify the requirements
* Functional and non-functional
* Requirements definitions usually consist of natural language, supplemented ( e,g., UML ) diagrams and tables.
* Three types of problems can arise:
* **LACK OF CLARITY**: It is hard to write documents that are both precise and easy-to-read.
* **REQUIREMENTS CONFUSION:** Functional and Non-functional requirements tend to be intertwined.
* **REQUIREMENTS AMALGAMATION**: Several different requirements may be expressed together.
* TYPES OF REQUIREMENTS:
* Functional Requirements : describe system SERVICES FUNTIONS.

| : Compute sales tax on a purchase

| : Update the database on the server

* NON-FUNCTIONAL REQUIREMENTS: are constraints on the system or the development process.

| : Non-functional requirements may be more critical than functional requirements.

|: If these are not met, the system is useless!

**ANALYSIS PHASE**

* The analysis phase defines the requirements of the system, independent of how these requirements will be accomplished.
* This phase defines the problems that the customer is trying to solve.
* The deliverable result at the end of this phase is a requirements document.
* ideally, this document states in a clear and precise fashion what is to be built.
* This analysis represents the “ **WHAT**” phase.
* The requirement documentaries to capture the requirements from the customer’s perspective by defining goals.
* This phase starts with the requirements document delivered by the requirement phase and mapsthe requirements into architecture.
* The architecture defines the components , their interfaces and behaviour.
* The deliverable design document is the architecture .
* This phase represents the “**HOW**” phase.
* Details on computer programming languages and environment , machines, packages, application architecture , distributed architecture layering, memory size, platform, algorithms, data structures ,global type definitions, interfaces, and many other engineering details are established.
* The design may include the usage of existing components.

DESIGN PHASE

* Design architecture document
* Implementation plan
* Critical priority analysis
* Performance analysis
* Test plan
* The design team can now expand upon the information established in the requirement document.
* The requirement document must guide this decision process.
* Analyzing the trade-offs of necessary complexity allows for many things to remain simple which , in turn, will eventually lead to a higher quality product. The architecture team also converts the typical scenarios into a test plan.

IMPLEMENTATION PHASE

* In the implementation phase , the team builds the components either from scratch or by composition.
* Given the architecture document from the design phase and the requirement document from the analysis phase, the team should build exactly what has been requested, though there is still room for innovation and flexibility.
* For example , a component may be narrowly designed for this particular system, or the component may be made more general to satisfy a reusability guideline.
* IMPLEMENTATION – Code
* CRITICAL ERROR REMOVAL
* The implementation phase deals with issues of quality, performance, baselines, libraries, and debugging.

The end deliverable is the product itself. There are already many established techniques associated with implementation.

TESTING PHASE

* Simply stated, quality is very important, Many companies have not learned that quality is important and deliver more claimed functionality but at a lower quality level.
* It is much easier to explain to a customer why there is a missing feature than to explain to a customer why the product lacks quality.
* A customer satisfied with the quality of a product will remain loyal and wait for new functionality in the next version.
* Quality is a distinguishing attribute of a system indicating the degree of excellence.
* Regression testing
* Internal testing
* Unit testing
* Application testing
* Stress testing
* The testing phase is a separate phase which is performed by a different team after the implementation is completed.
* There is merit in this approach; it is hard to see one’s own mistakes, and a fresh eye can discover obvious errors much faster than the person who has read and re-read the material many times.
* Unfortunately , delegating testing to another team leads to as lack attitude regarding quality by the implementation team.
* If the teams are to be known as craftsmen, then the teams should be responsible for establishing high quality across all phases.
* An attitude change must take place to guarantee quality. Regardless if testing is done after the -fact or continuously, testing is usually based on a regression technique split into several major focuses, namely internal, unit ,application, and stress.

**Maintenance Phase**

• Software maintenance is one of the activities in software engineering, and is the process of enhancing and optimizing deployed software (software release), as well as fixing defects.

• Software maintenance is also one of the phases in the System Development Life Cycle (SDLC), as it applies to software development. The maintenance phase is the phase which comes after deployment of the software into the field.

• The developing organization or team will have some mechanism to document and track Defects and deficiencies.

• configuration and version management

• reengineering (redesigning and refactoring)

• updating all analysis, design and user documentation

• Repeatable, automated tests enable evolution and refactoring

Maintenance is the process of changing a system after it has been deployed. Corrective maintenance: identifying and repairing defects Adaptive maintenance: adapting the existing solution to the new platforms. Perfective Maintenance: implementing the new requirements In a spiral lifecycle, everything after the delivery and deployment of the first prototype can be considered ―maintenance‖!

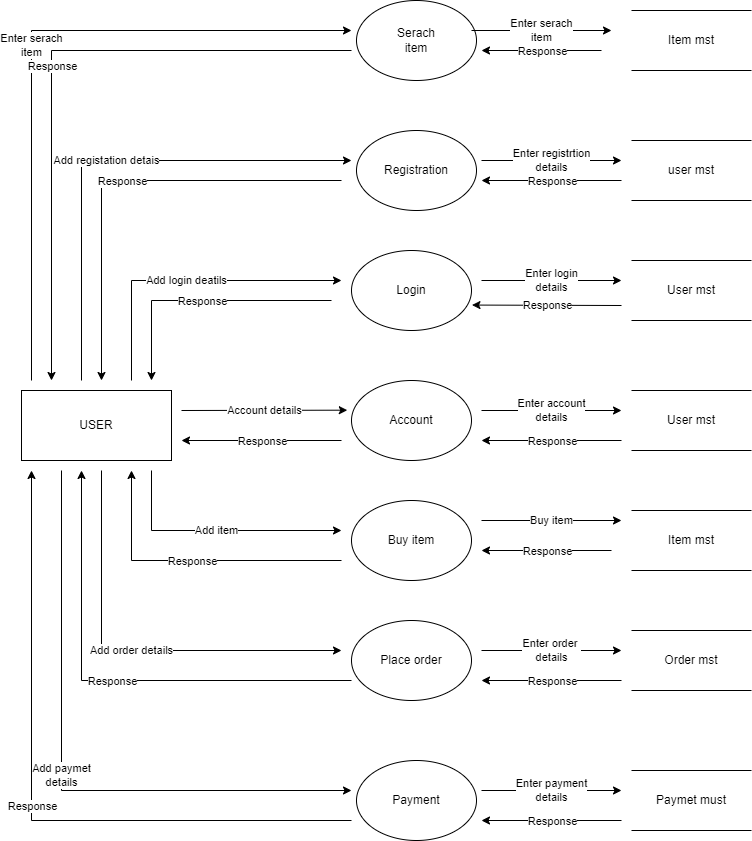
• Software just like most other products is typically released with a known set of defects and deficiencies.

The software is released with the issues because the development organization decides the utility and value of the software at a particular level of quality outweighs the impact of the known Defects and deficiencies.

**What is a data flow diagram (DFD)?**

A Data Flow Diagram (DFD) is a traditional way to visualize the information flows within a system. A neat and clear DFD can depict a good amount of the system requirements graphically. It can be manual, automated, or a combination of both. It shows how information enters and leaves the system, what changes the information and where information is stored. The purpose of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communications tool between a systems analyst and any person who plays a part in the system that acts as the starting point for redesigning a system.

DFD DIAGRAM ON FLIPCART



What is Flow chart? Create a flowchart to make addition of two numbers

A flow chart is a graphical or symbolic representation of a process. Each step in the process is represented by a different symbol and contains a short description of the process step. The flow chart symbols are linked together with arrows showing the process flow direction.

What is a flowchart?

A flowchart is a graphical representation of the operations involved in a data processing system.

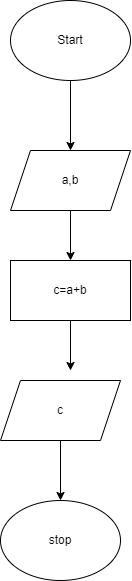


Fig 2 [ flowchart of addition of two number]

What is Use case Diagram? Create a use-case on

bill payment on Paytm.

Ds

A use case diagram is a graphical depiction of a user’s possible interactions with a system. A use case diagram shows various use cases and different types of users the system has and will often be accompanied by other types of diagrams as well. The use cases are represented by either circle or ellipses. The actor are often shown as stick figures

USE – CASE DIAGRAM [bill payment on paytm]

