

Fundamentals of Computer Engineering

Module II - Unit 5 Software.

Teachers: Moisés Martínez (1ºA English)

Year: 2022 - 2023

What is Software?

What is Software?

Software is a set of instructions called **programs** used to operate computers executing specific tasks.

Software is a sequence of instructions that tell a computer what to do.

Software is a sequence of computer **instructions**, which also includes data, documentation and other intangible components used to execute specific tasks in a computer.

What is Software?

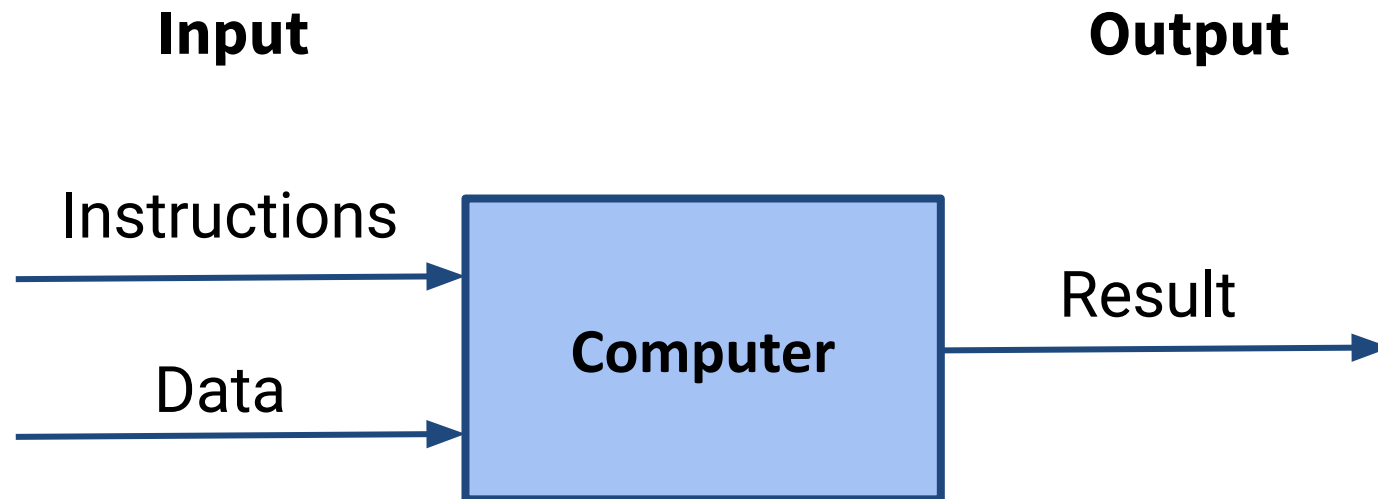
Software is a set of instructions called **programs** used to operate computers executing specific tasks.

Software is a sequence of instructions that tell a computer what to do.

Software is a sequence of computer **instructions**, which also includes data, documentation and other intangible components used to execute specific tasks in a computer.

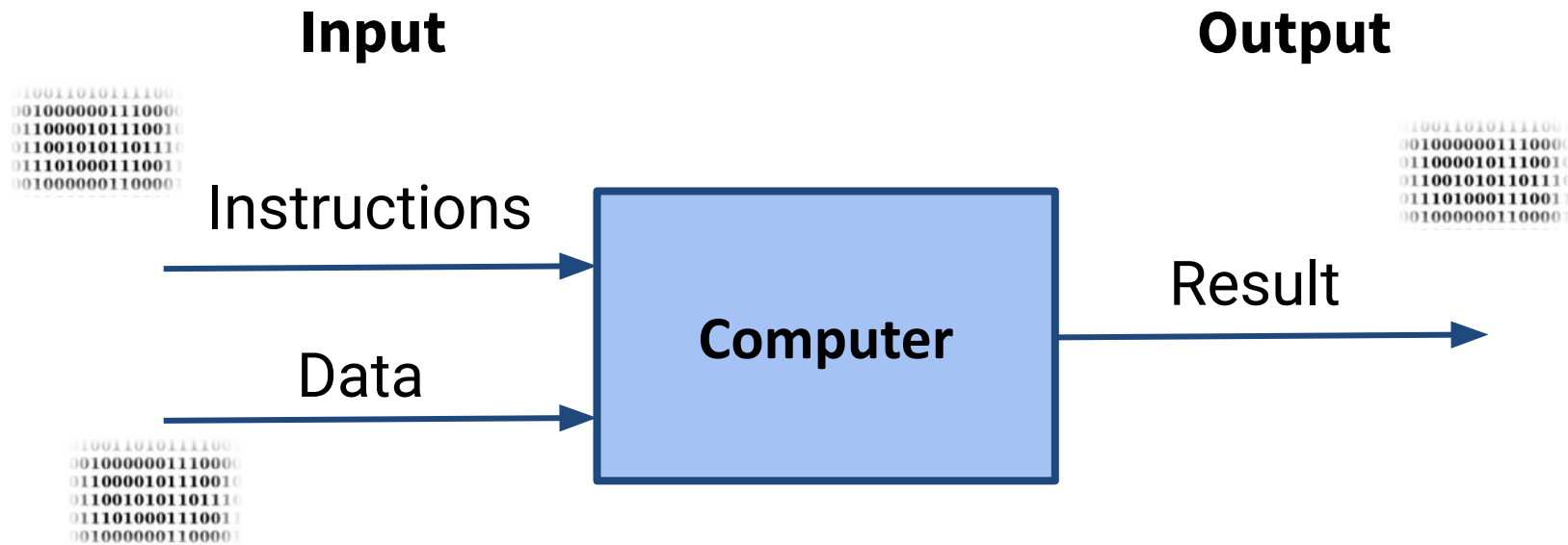
Basic Concepts

Software is a sequence of computer instructions, which also includes data, documentation and other intangible components used to execute specific tasks in a computer.



Basic Concepts

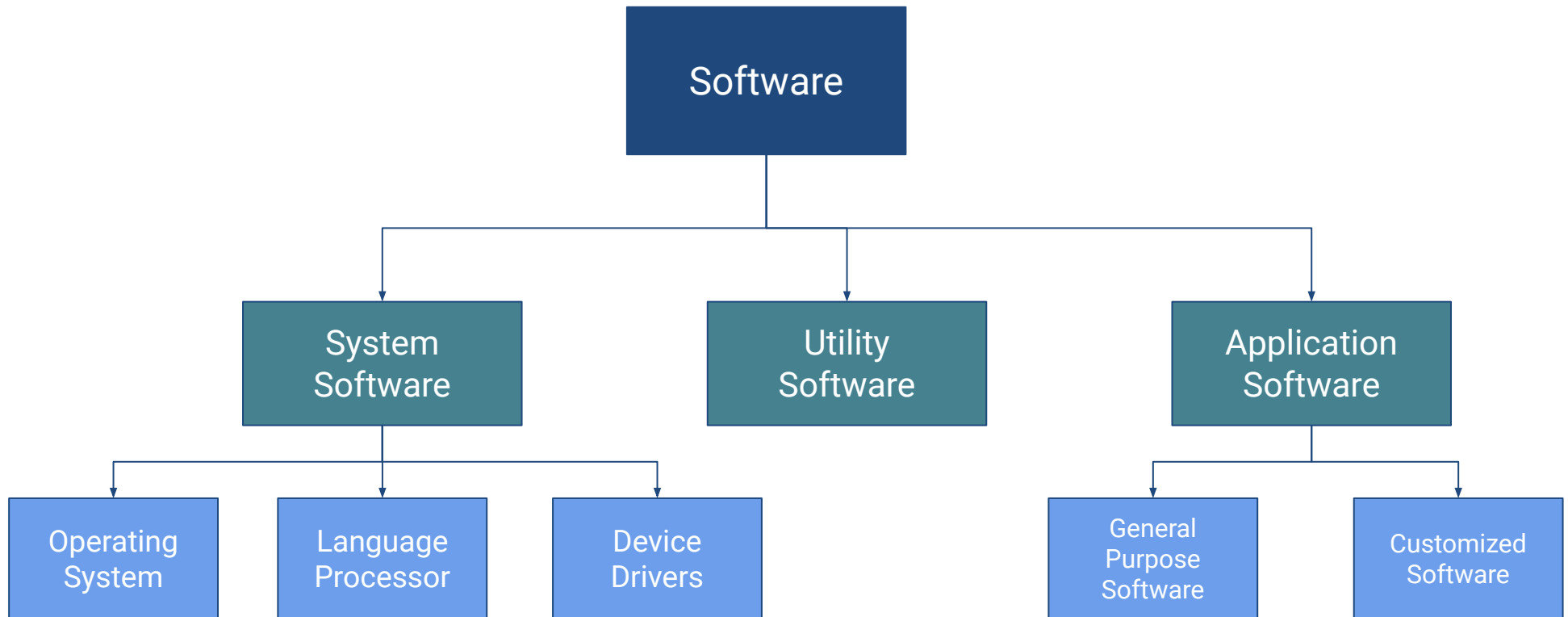
Software is a sequence of computer instructions, which also includes data, documentation and other intangible components used to execute specific tasks in a computer.



Software is made usually of computer programs which are a sequence of instructions in a specific programming language for a computer.

Basic Concepts

There are different types of software that can run on a computer: system software and application software.



Basic Concepts

System Software

System software is the software in a computer that directly operates the computer hardware and provides the basic functionality to the users as well as to the other software to operate smoothly.

- It controls the computer's internal functioning.
- it connects applications to Hardware devices.
- It is written in a low-level language in general.
- It is difficult to design and understand.
- It is fast in speed (execution speed) than other kind of softwares.
- It is written in a low-level language like assembly or machine code.

Basic Concepts

System Software

System Software is divided in three subtypes which are:

- Operating System (OS) Software is the main program of a computer. This software manages all the resources such as memory, CPU, hard disk, etc; and it provides an interface to the user which is used to interact with the computer system.
- Language Processor Software is an special software that converts the human-readable language into a machine language and vice versa. It converts programs written in high-level programming languages like Java, C, C++, Python; into a machine code or object code.
- Device Driver Software is a software that controls a device and helps that device to perform its functions. Every device like a keyboard, mouse, router, etc; needs a driver to connect with the computer system eternally.

Basic Concepts

Utility Software

Utility software is a special type of system software and performs specific tasks to keep the computer running in a good state. This kind of software is always running in the background.

- It performs specialized tasks to keep OS working safe.
- It is usually running in the background.
- It needs some OS software to execute.
- It is written in a high-level language in general, like python, C or C#.
- it can be controlled or configure by users.

Security programs like anti-virus scans and removes viruses from the OS. Most computers will include some sort of anti-virus software, but you can add your own.

Optimisation programs can include tools for system clean-up, disk defragmentation, and file compression.

Basic Concepts

Application Software

Application software is the software that performs special functions or provides functions that are much more than the basic operations of the computer, like the end-users applications. This type includes word processors, spreadsheets, database management, inventory, payroll programs, browsers, videogames, media apps, etc.

- It performs specialized tasks like word processing, spreadsheets, email, videogames etc.
- It needs more storage space.
- It needs some OS software to execute.
- It is more interactive and friendly for the users (user friendly), this means it is easy to use and design.
- It is written in a high-level language in general, like python, C or C#.

Basic Concepts

Application Software

Application Software is divided in two subtypes which are:

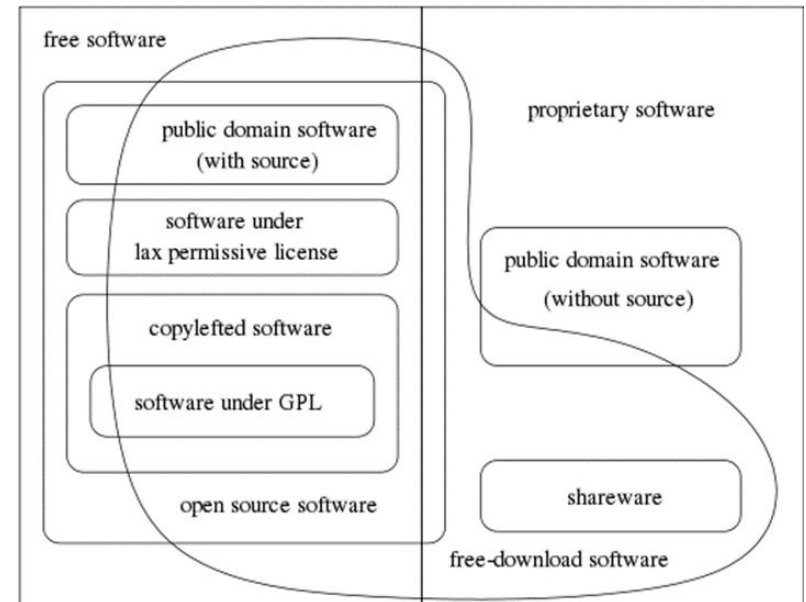
- General Purpose Software is created for general tasks which involved different kind of interaction with the . For example, MS-Word, MS-Excel, PowerPoint, VideoLan, Firefox etc.
- Customized Software is used or designed to perform specific tasks or functions or designed for specific organizations. For example, railway reservation system, airline reservation system, invoice management system, etc.

Basic Concepts - Licenses

Licenses

A software license is a legal instrument (usually by way of contract law, with or without printed material) governing the use or redistribution of software. According to the [GNU Project](#):

- Freeware. Free, copyrighted software.
- Shareware. Use with limitations.
- Free software. May be copied, modified and distributed.
- Open source software. Shared intellectual property.
- Proprietary software. It is not free, it belongs to a company.
- Commercial software. Its purpose is to generate economic profit.



Software development Life Cycle

Software development Life Cycle

The software development life cycle (SDLC) consists of a group of standardised phases followed by the software development team during the developing process. The number of phases in a software development life cycle can vary depending on the software **development methodology** and **framework** used to work.

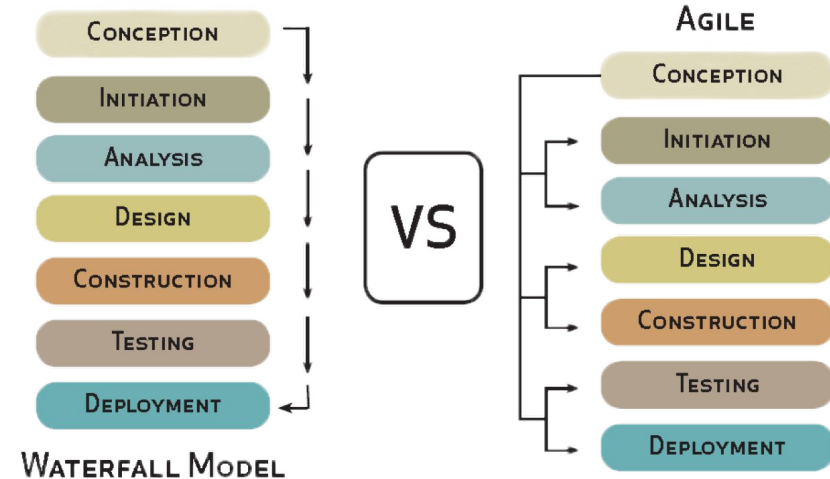


Software development Life Cycle

Software development methodology

Software development methodology is a process or a serie of processes used in software development.

- Waterfall (Big projects).
- Feature-Driven Development.
- Incremental.
- Rapid Application Design (RAD).
- Extreme Programming (XP).
- Scrum.
- Kanvan.
- Lean.

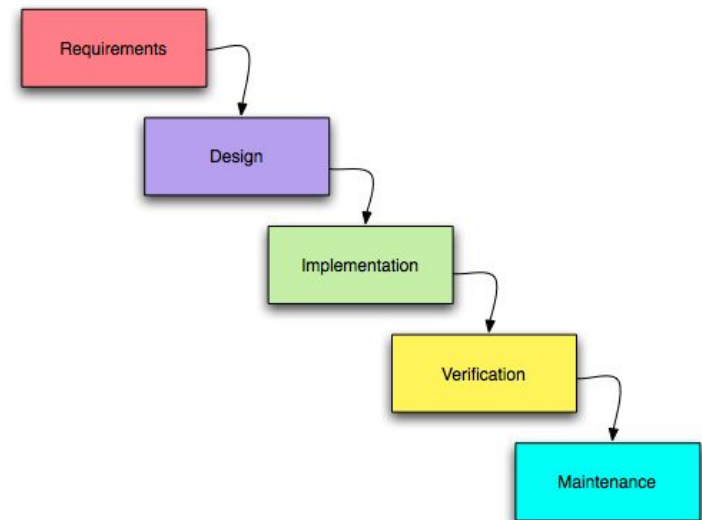


Software development Life Cycle

Software development methodology

The **waterfall methodology** is a linear project management approach, where stakeholder and customer requirements are gathered at the beginning of the project, and then a sequential project plan is created to accommodate those requirements.

- It has a static product backlog.
- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model.
- Phases are processed and completed one at a time.
- Process and results are well documented.
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Cannot accommodate changing requirements.

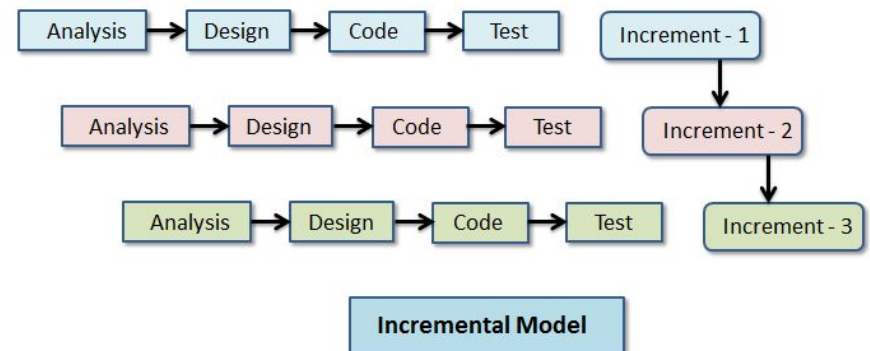


Software development Life Cycle

Software development methodology

The **Incremental methodology** is a linear project management approach, where the project requirements are divided into multiple standalone modules of the software development cycle. In this model, each module goes through the requirements, design, implementation and testing phases. Every subsequent release of the module adds function to the previous release.

- It requires a good planning designing.
- It is flexible and less expensive to change requirements and scope.
- Problems might cause due to system architecture as such not all requirements collected up front for the entire software lifecycle.
- Each iteration phase is rigid and does not overlap each other.
- Rectifying a problem in one unit requires correction in all the units and consumes a lot of time.
- Each increment can get feedback from customers and stakeholders.

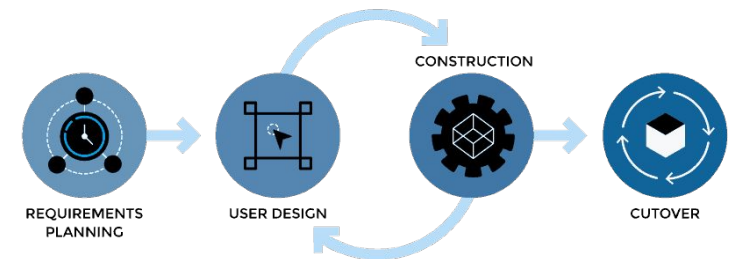


Software development Life Cycle

Software development methodology

The **Rapid Application Design (RAD) methodology** is a linear project management approach, where the project is focusing on designing and prototyping stage for the purpose of getting instant user feedback. Constant iterations of user feedback and quick incremental updates help to achieve better result at the end of the project.

- Project requirements can be changed at any time.
- It needs strong team collaboration.
- Encourages and priorities customer feedback.
- It cannot work with large teams
- it needs highly skilled developers (senior level).
- The time between prototypes and iterations is usually short.
- Systems which can be modularised can only be developed using it.
- It is more complex to manage when compared to other linear models.

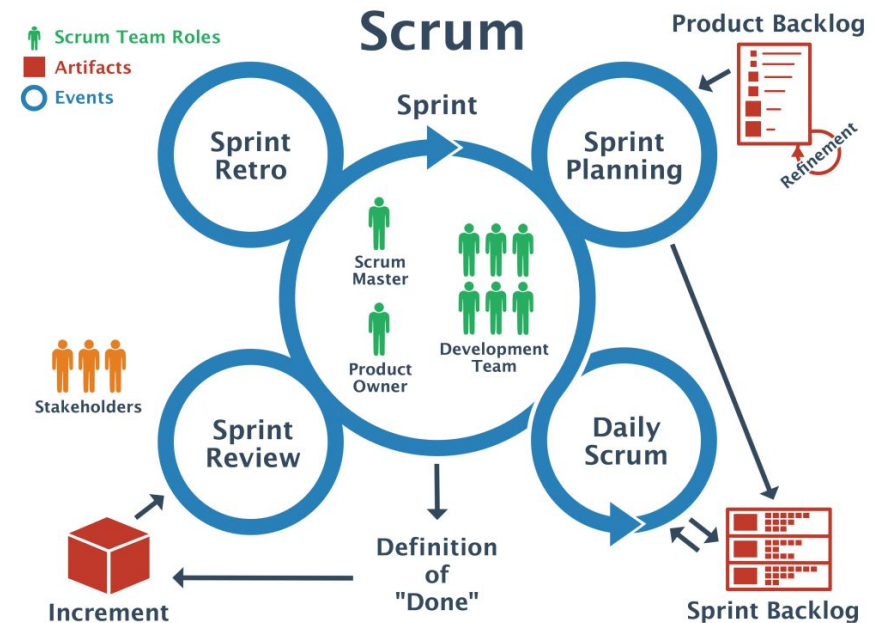


Software development Life Cycle

Software development methodology

The **Scrum methodology** is an agile project management approach, where the project is evolving during a sequence of **Sprints** (short and periodic blocks for development composed of meetings and development) which provides a complete result that is a variation of the final product that must be able to be delivered to the client.

- It focus in team interactions to ensure a good project evolution.
- It has a dynamic product backlog.
- It help teams complete project deliverables quickly and efficiently.
- It works well for fast-moving development projects.
- It is easy to get and adopt feedback from customers and stakeholders.
- The chances of project failure are high if individuals aren't very committed or cooperative.
- The daily meetings sometimes frustrate team members.



Software development Life Cycle

Software development methodology

The **Extreme Programming (XP) methodology** is an agile project management approach similar to scrum but focus on the software code, where the project is evolving during a sequence of **Sprints** which provides a complete result that is a variation of the final product that must be able to be delivered to the client.

- It focus in software development to ensure a good project evolution.
- It has a dynamic product backlog.
- The team is focused on the most important tasks at hand.
- It has stable software through continuous testing.
- It has relatively large time investment and high costs.
- The software code is usually clear and comprehensible at all times.
- It is easy to get and adopt feedback from customers and stakeholders.
- it needs highly skilled developers (senior level).

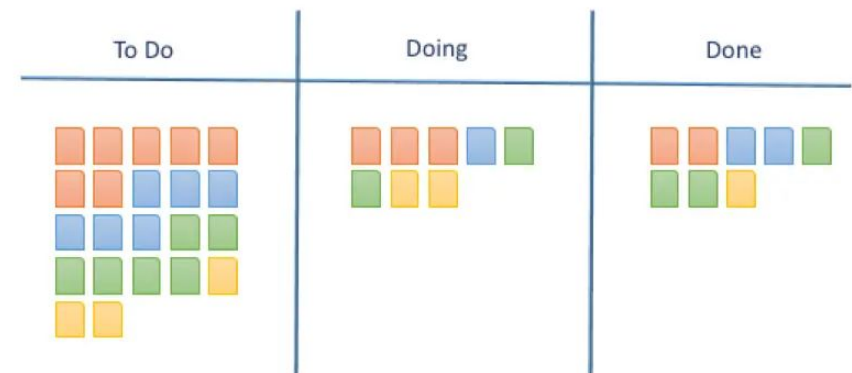


Software development Life Cycle

Software development methodology

The **Kanvan methodology** is an agile project management approach, where the project is evolving balancing demands with the available capacity, and by improving the handling of system-level bottlenecks. This means that tasks from backlog are solved according to the team capacity.

- It is ease and simple to understand and use.
- It offers a maximum adaptability, which is incredible for more extensive ventures that require ongoing changes.
- It encourages collaboration making the whole team work together to convey the ideal outcomes.
- It reduces costs and wastage.
- It does not fit into really well in dynamic projects.
- There are no real phases defined during the life cycle.
- There are no timeframes associated with each phase.



Software development Life Cycle

Software development methodology

The **Lean methodology** is an agile project management approach, where the project is evolving by optimizing the people, resources, effort, and energy toward creating value for the customer. It is based on two guiding tenets, software continuous improvement and respect for people.

- It is ease and simple to understand and use.
- It is focus on profits maximization. Lean practices will help in cost reduction as well as savings that can be added to profit.
- It is easy to get and adopt feedback from customers and stakeholders.
- There are high chances of missing the deadline and sometimes even a second chance is not given due to the lack of flexibility.
- The project is highly dependent on cohesiveness of the team and the individual commitments of the team members.
- it needs highly skilled developers (senior level).

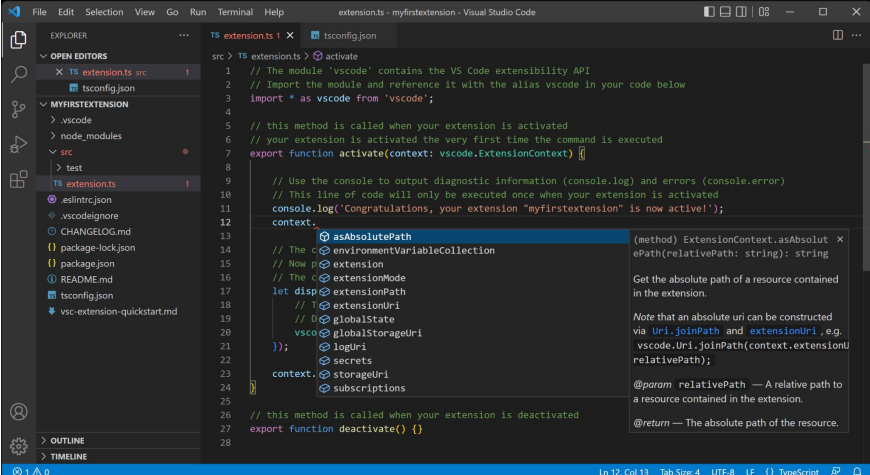


Software development Life Cycle

Software development framework

Software development framework is an abstraction in which software can be development. Frameworks provides a standard way to build and deploy software applications and is a universal, reusable software environment that provides particular functionality as part of a larger software platform to facilitate the development software applications.

An Integrated development environment (IDE) is a software application that provides some tools to software development like source code editor, build automation tools and a debugger. IDEs allow to deploy different frameworks.



```
src > ts extensions > activate
1 // The module 'vscode' contains the VS Code extensibility API
2 // Import the module and reference it with the alias vscode in your code below
3 import * as vscode from 'vscode';
4
5 // this method is called when your extension is activated
6 // your extension is activated the very first time the command is executed
7 export function activate(context: vscode.ExtensionContext) {
8
9
10 // Use the console to output diagnostic information (console.log) and errors (console.error)
11 // This line of code will only be executed once when your extension is activated
12 console.log("Congratulations, your extension 'myfirstextension' is now active!");
13 context
14
15 // The c@environmentVariableCollection
16 // Now p@extension
17 // The c@extensionMode
18 let disp@extensionPath
19 // T@extensionUri
20 // D@globalState
21 vsco@globalStorageUri
22 }); logUri
23 // secrets
24 context. storageUri
25 // subscriptions
26
27 // this method is called when your extension is deactivated
28 export function deactivate() {}
```

(method) ExtensionContext.asAbsolute ×
ePath(relativePath: string): string
Get the absolute path of a resource contained in the extension.
Note that an absolute uri can be constructed via Uri.joinPath and extensionUri, e.g. vscode.Uri.joinPath(context.extensionUri, relativePath);
@param relativePath — A relative path to a resource contained in the extension.
@return — The absolute path of the resource.

Software development Life Cycle

Why is the software development life cycle important?

Follow a software development life cycle results usually improve the quality of resulting software product and reduce the time needed to build and deploy decreasing the overall costs. Besides ...

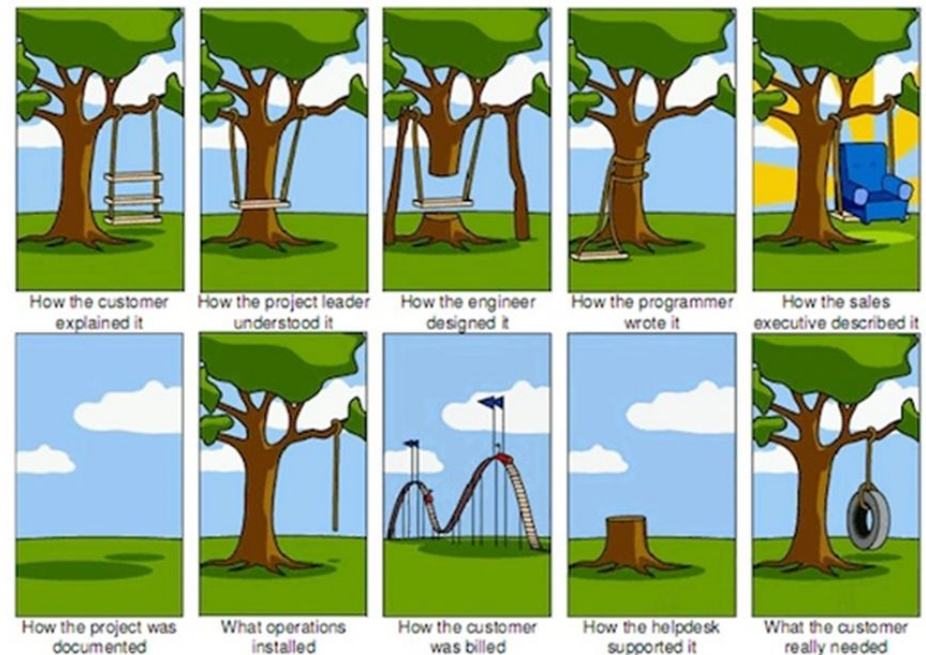
- It provides management with a higher degree of oversight and control.
- It generates documentation about the software and the development process.
- The software goals are defined by all stakeholders at the beginning allowing to define a roadmap to reach them.
- All stakeholders can provide their input at the most appropriate and effective phase of the software development process.

Software development Life Cycle

Analysis phase

During this first phase, the needs of the customer and the end-users are defined and analysed.

- What problem will the software solve? What is solving that problem potentially worth?
- How big a potential user base does the software have?
- Does other software exist that solves the same problem?
- How will this software improve on that?
- How much might it cost to develop software to effectively solve the problem?
- Is solving the problem valuable enough to offer an acceptable return on investment on that cost?
- Do we have the resources to support that cost?



Software development Life Cycle

Design phase

During this phase, the architecture/structure of the software is defined based on the analysis definition done in the previous phase.

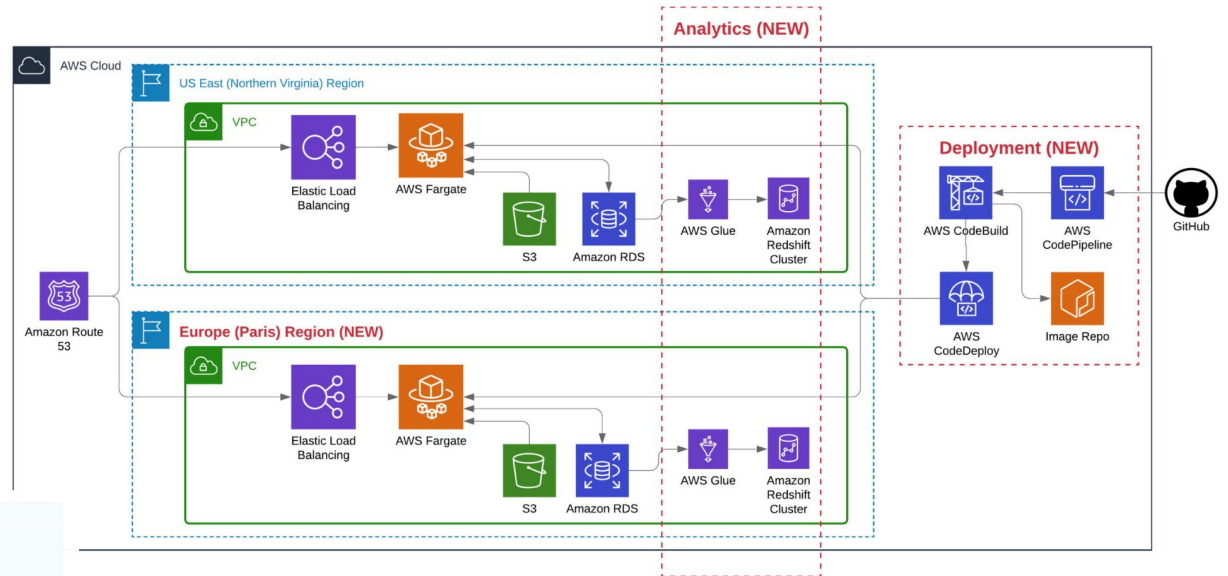
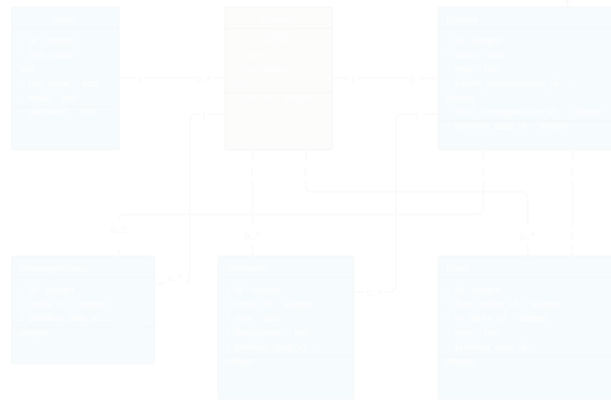
The result of this phase is the Software Design Document (SDD) which contains the detailed description of the system where each part/component must be defined. This documents is usually divided into two documents:

- High-level or Architectural Design Document (ADD) defines the structure of the solution (once the analysis phase has described the problem) by identifying major modules (sets of functions that will be associated) and their relationships.
- Detailed Design Document (DDD) defines the components and/or algorithms defined and the structure of the code.



Software development Life Cycle

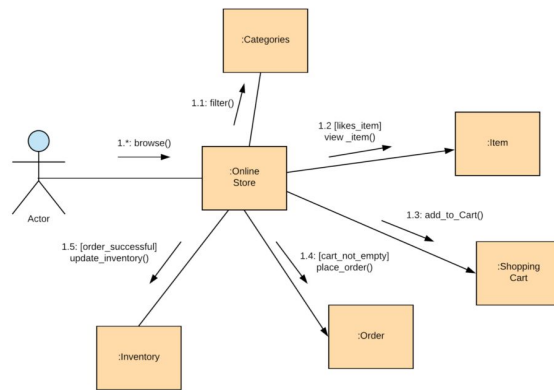
Design phase



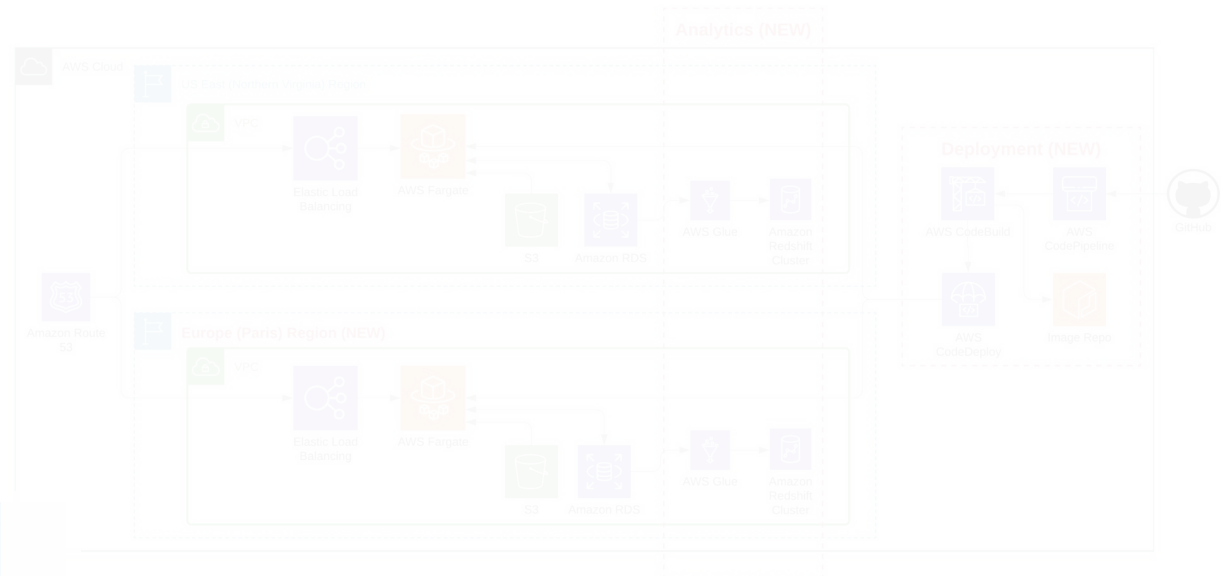
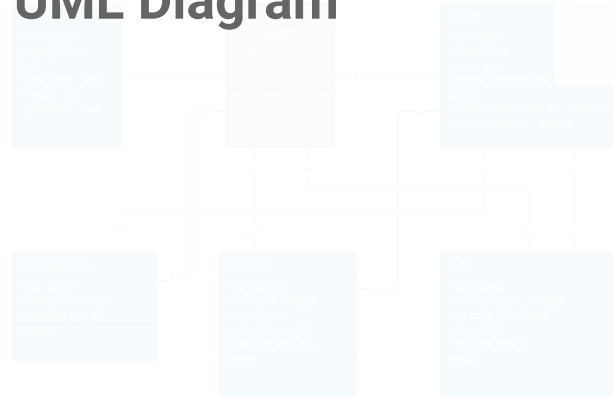
Cloud Architecture Diagram

Software development Life Cycle

Design phase

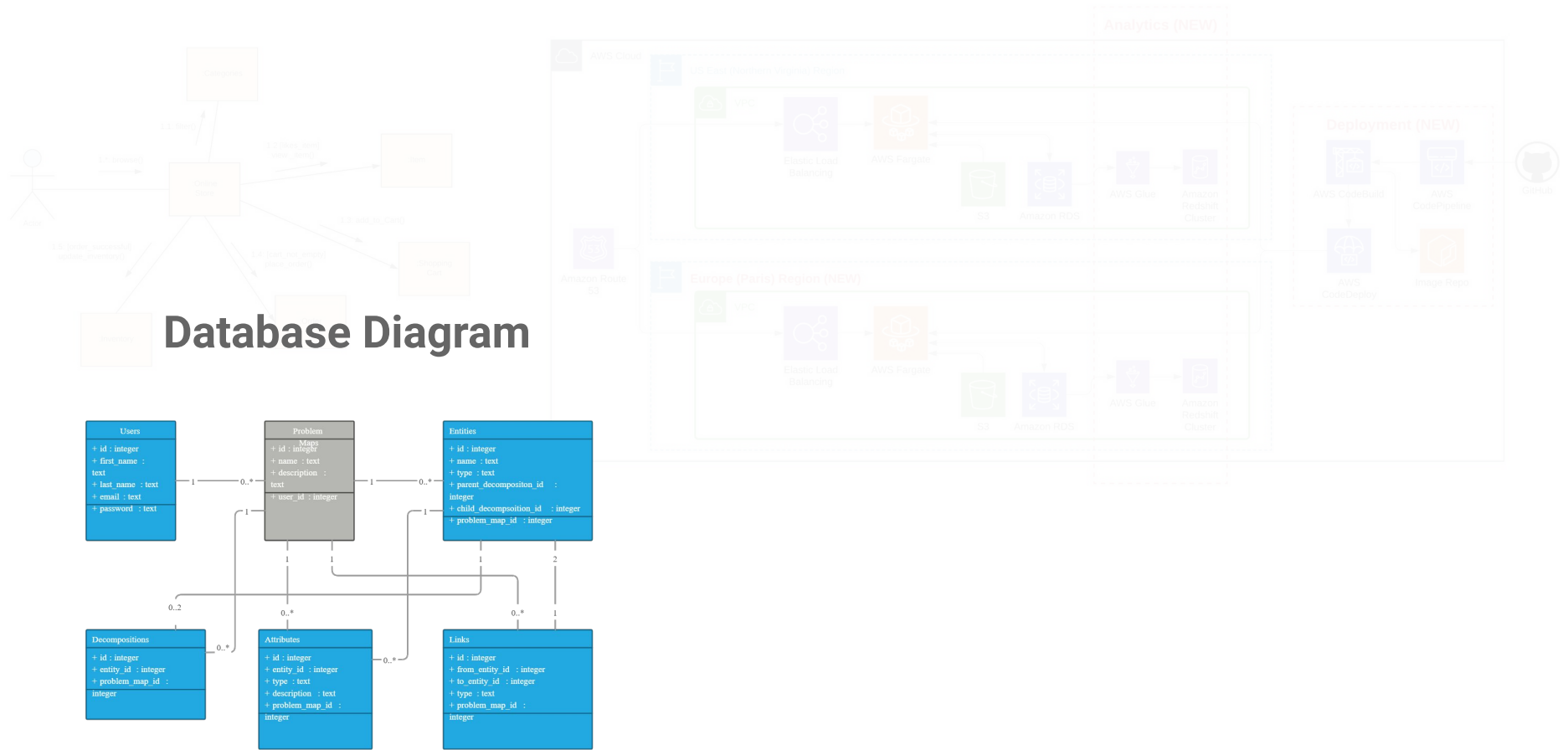


UML Diagram



Software development Life Cycle

Design phase

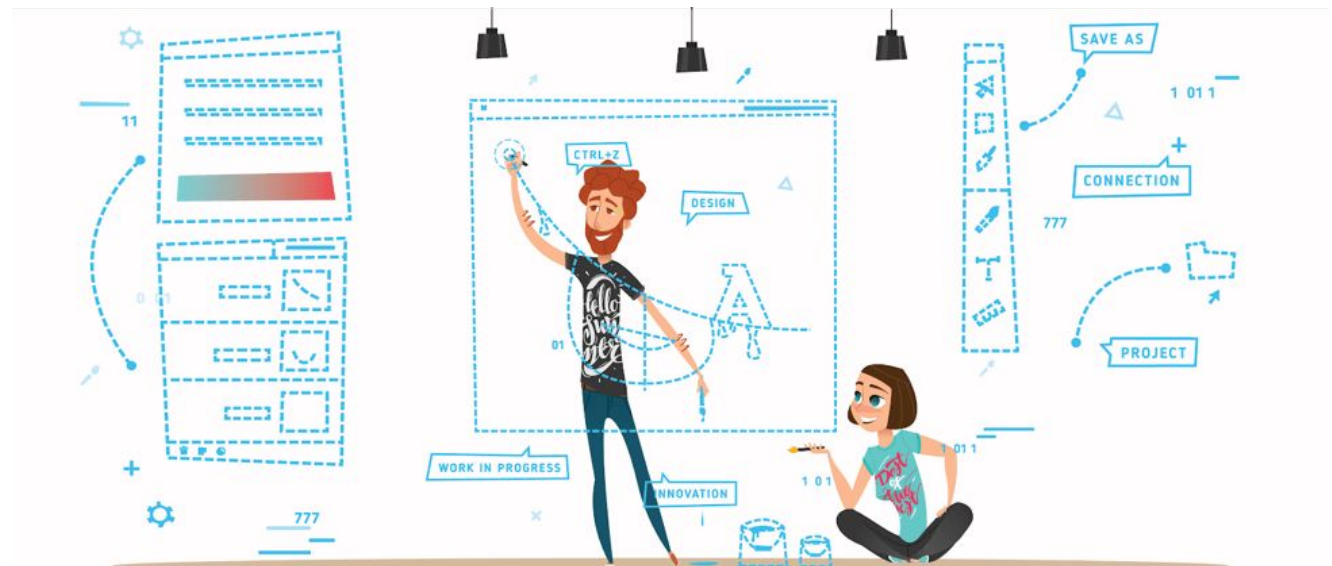


Software development Life Cycle

Design phase

In this phase is really important to identify mostly of the elements of the software even the interactions with other softwares.

- GUI (Graphical User Interfaces).
- Storage softwares.
- Algorithms.
- Components.

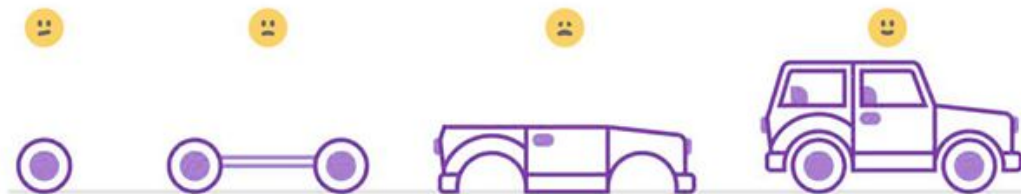


Software development Life Cycle

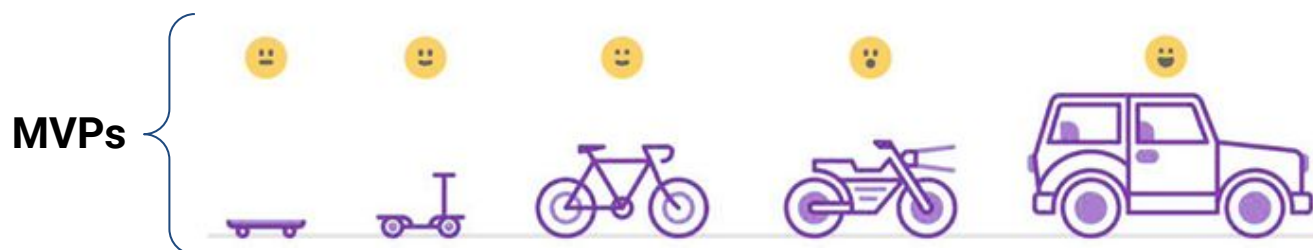
Development phase

The development phase is where the software is coded and tested.

- In a Waterfall methodology, the development is defined and documented in technical detail from beginning to end during and then it build from beginning to end.



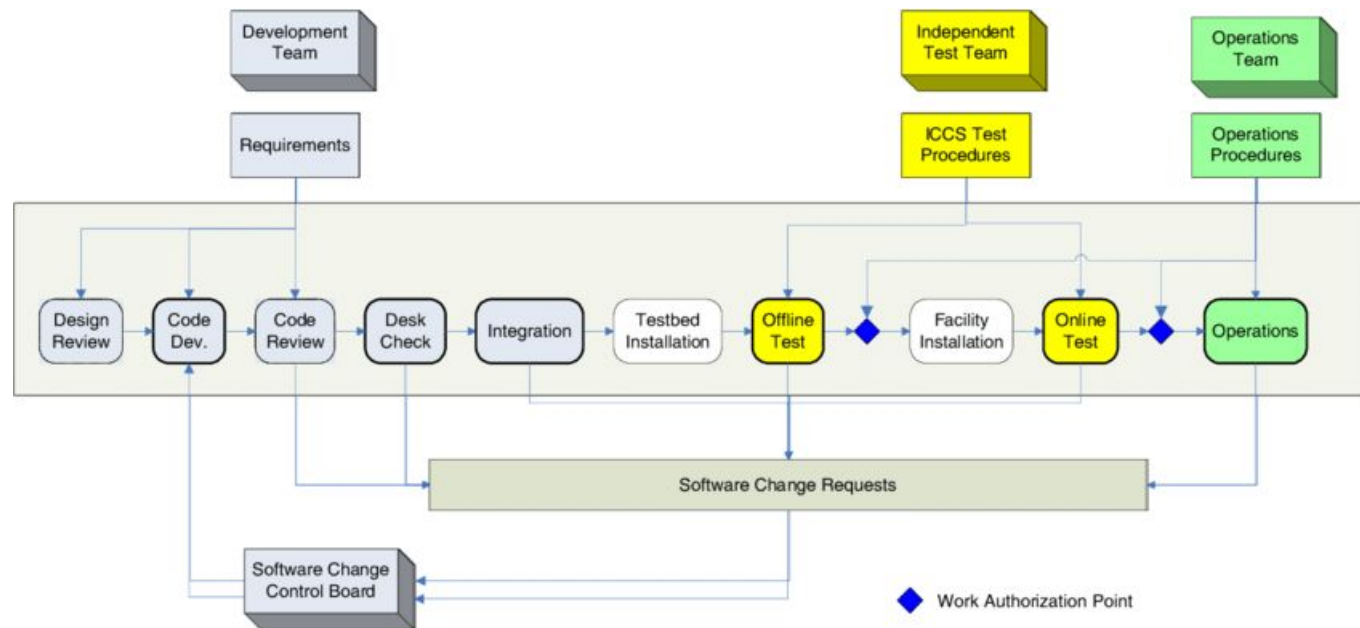
- In an Agile methodology, the development is divided in different iterative phases of an iterative SDLF, where in each phase is built a Most Valuable Product (MVP).



Software development Life Cycle

Deployment phase

The deployment phase is when the software application is moved from the development and testing environments into a production environment where it is available to users. Deployment can be a relatively simple process or an extremely complex one, depending on the nature of the software.



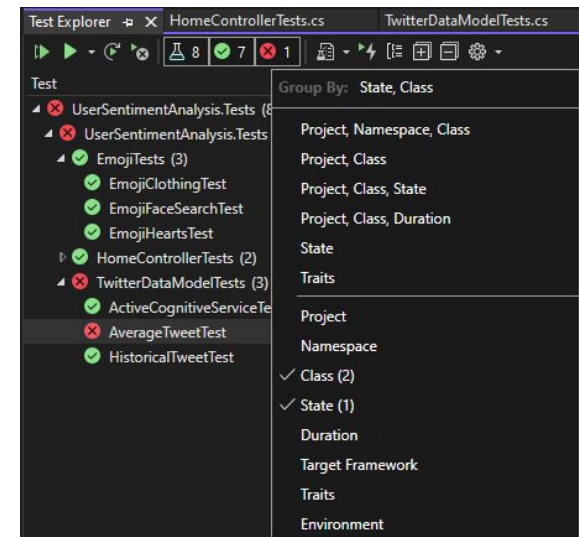
Software development Life Cycle

Testing phase

The testing phase is probably the most important phase, the software developed in the previous phase is tested against requirements defined in the planning and analysis phase as well as for performance and bugs.

The **goal** of the phase is to **ensure the software product has been developed as defined**, both in terms of **user functionality** and the **resources** needed for it to run smoothly and meet availability and security requirements. Software testing usually involves:

- Manual testing of user journeys.
- Automated testing based on tools and scripts.



Software development Life Cycle

Maintenance phase

The maintenance phase involves making sure the software application continues to run as it is intended to in terms of functionalities, performance and availability.

This phase involves some tasks, during the whole life of the product, which it does not finish when the development phase is finish. These tasks are:

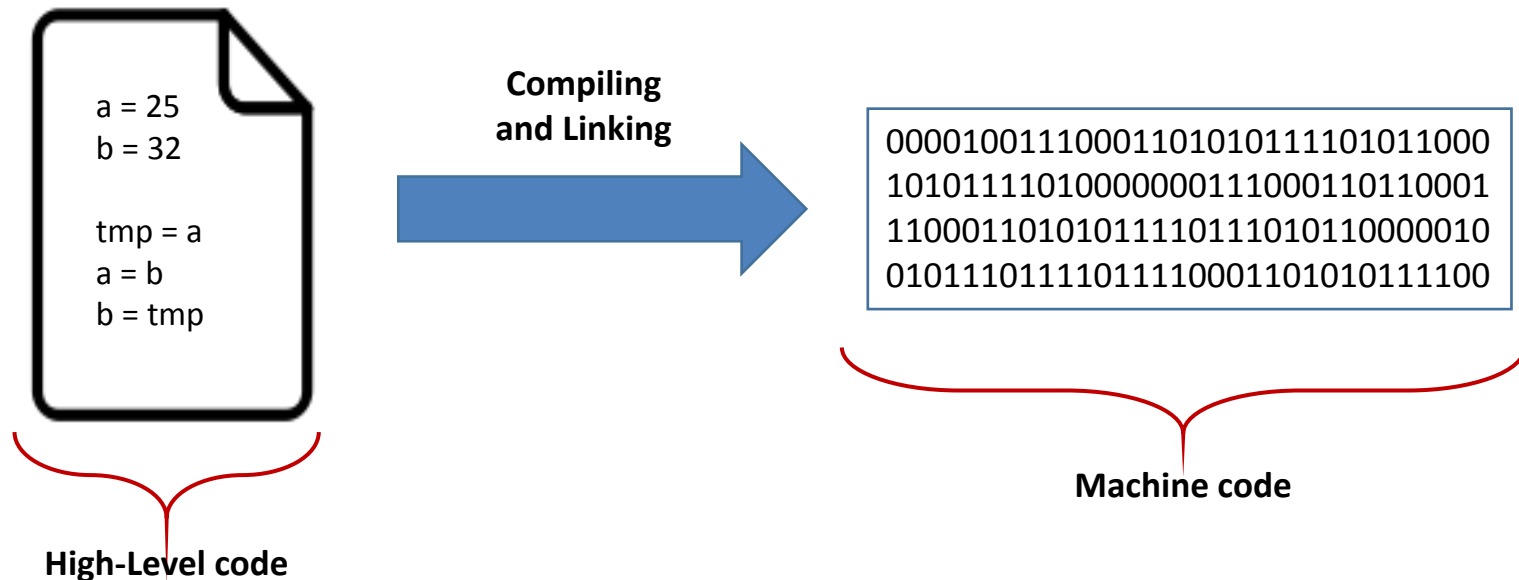
- Monitoring software performance.
- Evaluation.
- Error and bugs repair.
- System improvement or evolution.



From code to CPU

From code to CPU

Software is commonly written in High-Level and/or Low-Level languages. Software must be compiled to generate the machine code to execute in the CPU. Machine code is made up of machine instructions.



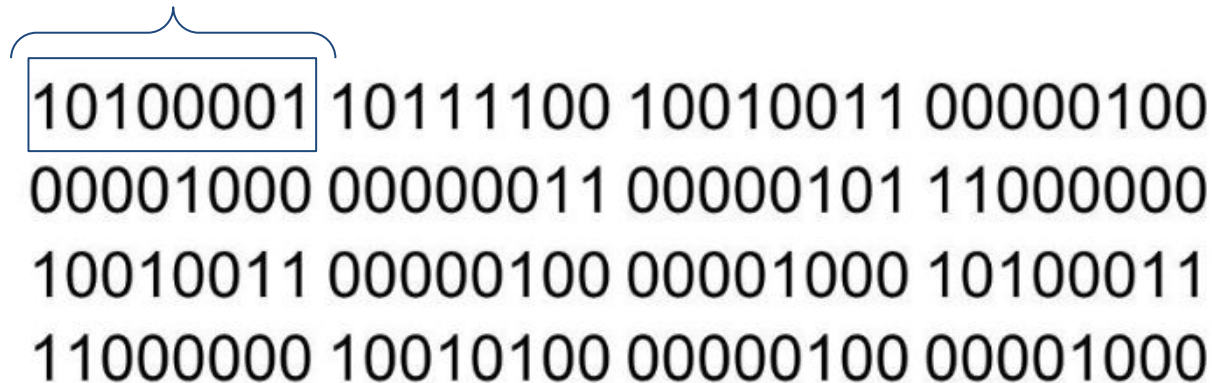
A machine instruction is an elementary operation that can execute in the CPU.

From code to CPU

There are three different types of computer languages:

- The **machine language**, also called machine code or object code, is a set of binary digits 0 and 1. These binary digits are understood and read by a computer system and interpreted easily. It is considered a **native language** as it can be directly understood by a central processing unit (CPU).

Byte



```
10100001 10111100 10010011 00000100
00001000 00000011 00000101 11000000
10010011 00000100 00001000 10100011
11000000 10010100 00000100 00001000
```

This code means: $z = x + y$;

From code to CPU

There are three different types of computer languages:

- The **assembly language** or low-level language refers to the low level of abstraction from machine language. It is used to program for microprocessors and many other programmable devices. This language uses registers, memory addresses, and call stacks.

<pre>if (op1 === op2) { x = 1; } else { x = 2; }</pre>	<pre>mov ax, op1 mov bx, op2 cmp ax, bx jne L1 mov x, 1 jmp L2 L1: mov x, 2 L2:</pre>	<p>Jump if not equal</p> <p>Jump</p>
JavaScript	Assembly Language	

These languages are different depending of the processor architecture (Z80, X86, x86-64, IA64, or AMD64).

From code to CPU

There are three different types of computer languages:

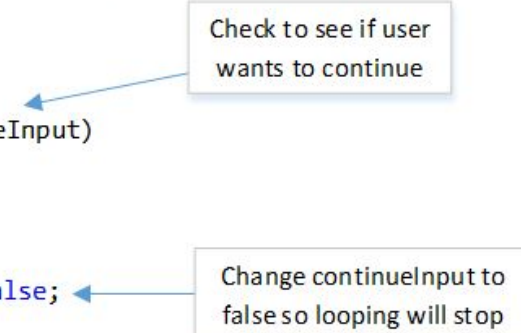
- The **high-level languages** refers to the higher level of abstraction from machine language. These languages allow programmers to write and test code more efficiently. Besides, they have more safeguards to keep coders from issues commands that could potentially damage computers, although they do not give programmers as much control as low-level ones do.

```
string inputString;

cout << "Enter student names (max of 50). Enter q to quit" << endl;

int count = 0;
bool continueInput = true;

while (count ++ < 50 && continueInput)
{
    cout << "Name: ";
    cin >> inputString;
    if (inputString == "q")
        continueInput = false;
}
```



- C
- C++
- C#
- Pascal
- Java
- Python

From code to CPU

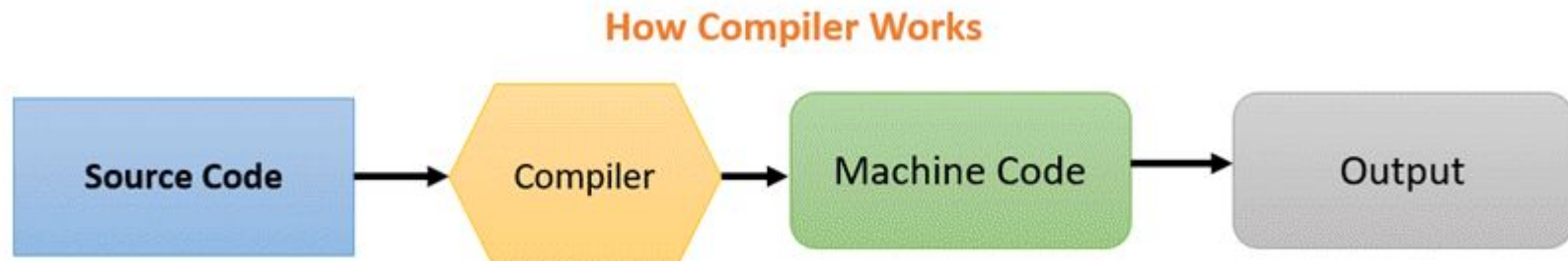
Main differences between high-level and low-level languages.

High Level Language	Low Level Language
Programmer friendly	Machine friendly
Less memory efficient	Highly memory efficient
Easy to understand for programmers	Difficult to understand for programmers
Simple to debug	Complex to debug comparatively
Simple to maintain	Complex to maintain comparatively
Portable	Non-portable
Machine-independent. They can run on any platform	Machine-dependent
Needs compiler or interpreter for translation	Needs assembler for translation
Widely used for programming	Not commonly used in programming

From code to CPU

Compilers and Interpreters

A **compiler** is a computer program that transforms code written in a high-level programming language into the machine code. This means that a compiler is a program which translates the human-readable code to a language a computer processor understands (binary 1 and 0 bits).



Compilers convert the code into machine code (creating an exe) before program runs,

From code to CPU

Compilers and Interpreters

An **interpreter** is a computer program, which converts each high-level program (source code, pre-compiled code, and scripts) statement into the machine code. This means that a compiler is a program which translates the human-readable code to a language a computer processor understands (binary 1 and 0 bits).



Interpreters convert code into machine code when the program is running.

From code to CPU

Compilers and Interpreters

Dimension	Compilers	Interpreters
Advantage	The program code is already translated into machine code. Thus, its code execution time is less.	Interpreters are easier to use, especially for beginners.
Disadvantage	You can't change the program without going back to the source code.	Interpreted programs can run on computers that have the corresponding interpreter.
Machine code	Store machine language as machine code on the disk	Not saving machine code at all.
Running time	Compiled code runs faster	Interpreted code runs slower
Model	It is based on language translation linking-loading model.	It is based on Interpretation Method.

From code to CPU

Compilers and Interpreters

Dimension	Compilers	Interpreters
Program generation	Generates output program (in the form of exe) which can be run independently from the original program.	Do not generate output program. So they evaluate the source program at every time during execution.
Execution	Program execution is separate from the compilation. It performed only after the entire output program is compiled.	Program Execution is a part of Interpretation process, so it is performed line by line.
Memory requirement	Target program execute independently and do not require the compiler in the memory.	The interpreter exists in the memory during interpretation.
Input	It takes an entire program	It takes a single line of code.

From code to CPU

Compilers and Interpreters

Dimension	Compilers	Interpreters
Output	Compilers generate intermediate machine code.	Interpreter never generate any intermediate machine code.
Errors	Display all errors after, compilation, all at the same time.	Displays all errors of each line one by one.
Pertaining Programming languages	C, C++, C#, Scala, Java all use compiler.	PHP, Perl, Ruby uses an interpreter.
Output	Compilers generate intermediate machine code.	Interpreter never generate any intermediate machine code.