**What is a Software?**

software, instructions that tell a computer what to do. The software comprises the entire set of programs, procedures, and routines associated with the operation of a computer system.

**What is Software Engineering?**

Software engineering is a field of engineering, for designing and writing programs for computers or other electronic devices. A software engineer, or programmer, writes software and compiles the software to make instructions the computer can follow. Good quality software is easier to fix and add new features.

**Object-Oriented Principles when designing software:**

* **S.O.L.I.D:**
* Single Responsibility (SRP):

A class that is implemented should have one single job to be performed. Not only the classes but also the modules, functions, APIs etc.

* Open/Closed Principle (OCP):

Object/Entities should be opened for extension but closed for modifications. In practice, this means creating software entities whose behaviour can be changed without the need to edit and recompile the code itself.

* Liskov substitution Principle:

The Liskov Substitution Principle states that any subclass object should be substitutable for the superclass object from which it is derived.

* Interface segregation principle (ISP):

The interface segregation principle states that no code should be forced to depend on methods it does not use.

* Dependency inversion principle (DIP):

The Dependency Inversion Principle states that high-level modules should not depend on low-level modules; both should depend on abstractions. Abstractions should not depend on details. Details should depend upon abstractions.

* **Approaching the solutions:**
* Think throughout the problem:
* Make sure to clear any unclear parts before designing the solution and you understand the problem that you are going to resolve.
* Divide and conquer:
* Divide the problem into smaller problems and make it manageable and easily understandable. Using that find the priority and clarity.
* KISS:
* Do not deliberately make your solution complex, keep it simple and stupid
* Learn, especially from mistakes:
* Always anticipate the changes as much as possible and do not over-engineer them, but keep the provisions to extend
* Always remember why software exists:
* Loss of the bigger picture might cause following a wrong path.
* Remember that you are not the user:
* End-user is not technically capable as same as you are so, do not assume that the user will understand. Always the user-friendliness and user experience matter.
* **Implementing the solution:**
* YAGNI-(You ain’t gonna need it):
* Write the code you need for the moment only that.
* DRY-(Don’t repeat yourself):
* Always reuse the code you wrote.
* Embrace abstraction:
* Make sure your system functions properly without knowing the implementation details of every component part.
* DRITW-(Don’t reinvent the wheel):
* Someone else might have already solved the same problem. Make use of that.
* Write code that does one thing well:
* A single piece of code that does one thing and that one thing well.
* Debugging is harder than writing code:
* Readable code is better than compact code.
* Kaizen:
* Fix not just the bug but the code around it.

**Best practices in Software Engineering:**

* Unit testing:
* The unit could be a class, function, module, API etc. It will verify the class, function Is working as expected and delivers the expected output.
* Code quality:
* Code should be readable and easily understandable.
* Code review:
* Peer reviews, lead reviews and pair programming are some methods of doing code reviews and code review is the best way to improve the code quality
* Version Controlling:
* Code should always be version controlled and it should let multiple developers collaborate on the same code base.
* Continuous integration:
* Developers need to check the code to a shared repository several times a day.

**What is version controlling?**

Version control, also known as source control, is the practice of tracking and managing changes to software code. Version control systems are software tools that help software teams manage changes to source code over time. Version control helps teams solve these kinds of problems, tracking every individual change by each contributor and helping prevent concurrent work from conflicting.

Version control software is an essential part of the everyday modern software team's professional practices. Individual software developers who are accustomed to working with a capable version control system in their teams typically recognize the incredible value version control also gives them even on small solo projects.

**Why is version controlling?**

Version control is important to keep track of changes and keep every team member working on the right version. You should use version control software for all code, files, and assets that multiple team members will collaborate on. In fact, good version control software supports a developer's preferred workflow without imposing one way of working. Ideally, it also works on any platform, rather than dictate what operating system or toolchain developers must use. Great version control systems facilitate a smooth and continuous flow of changes to the code rather than the frustrating and clumsy mechanism of file locking - giving the green light to one developer at the expense of blocking the progress of others.

**Best practices of VCS:**

1. Commit often

When developing a feature, there are several steps needed to complete the whole feature. By committing often, you are ensuring that you are dealing with a small change at a time and it will make it easier to revert if the change introduces a bug or if the change is no longer necessary. It will also help you group your commitment in a consistent way. Each commit will contain a particular change.

1. Add a commit message

Commit is communication - a commit message is a way to convey a message to other team members. A good commit message should be a short text explaining what was done. Even if it is not compulsory, a commitment message is very important.

1. Do not push or commit unfinished work!

Even though it is required to commit often it doesn’t mean that you need to commit or push a work that is not completed.

1. Decide on common branch practices.

Set a rule when working with branches and stick to it. A common practice should be agreed upon prior to development and should be followed by each team member. updating your branches often will make your integration easier, this will result in fewer conflicts to handle. Updating a branch involves checking out the central or remote repository and integrating changes in the current branch. Another best practice could be creating a new branch for every change since branches are generally easy to manipulate and to create and this makes management easier.

1. Push source code to your repository

The repository is not a place where you push everything. Only source code related to the project you are working on should be pushed on the repository, it is not the image of your local workspace. Some files should not be pushed on the repository: generated files, workspace configuration, etc.

**Popular VCS:**

* Helix Core (Perforce)
* Git
* SVN
* ClearCase
* Mercurial
* TFS

**What is Git?**

Git is a free and open-source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. It is a DevOps tool used for source code management. Git is used to track changes in the source code, enabling multiple developers to work together on non-linear development. On the other hand, git can be introduced as a distributed version control as well. Which means the client gets a complete clone of the source code. In a disaster situation, full source along with all history can be restored from a client.

**What is Git vs GitHub?**

The key difference between Git and GitHub is that Git is open-source tool developers install locally to manage source code, while GitHub is an online service to which developers who use Git can connect and upload or download resources.

**Advantages of Git:**

* Unlike centralized version control systems, Git branches are cheap and easy to merge.
* This facilitates the feature branch workflow popular with many Git users.
* Feature branches provide an isolated environment for every change to your codebase.

**Git commands:**

* Git status

Displays the state of the working directory and the staged snapshot. You’ll want to run this in conjunction with git add and git commit to see exactly what’s being included in the next snapshot.

* Git branch

This command is your general-purpose branch administration tool. It lets you create isolated development environments within a single repository.

* Git init

Initializes a new Git repository. If you want to place a project under revision control, this is the first command you need to learn.

* Git Add

Moves changes from the working directory to the staging area. This gives you the opportunity to prepare a snapshot before committing it to the official history.

* Git commit

Passing the --amend flag to git commit lets you amend the most recent commit. This is very useful when you forget to stage a file or omit important information from the commit message. A commit is the Git equivalent of a "save". Traditional saving should be thought of as a file system operation that is used to overwrite an existing file or write a new file. Alternatively, Git committing is an operation that acts upon a collection of files and directories.

* Git push

Pushing is the opposite of fetching (with a few caveats). It lets you move a local branch to another repository, which serves as a convenient way to publish contributions. This is like svn commit, but it sends a series of commits instead of a single changeset.

* Git clone

Creates a copy of an existing Git repository. Cloning is the most common way for developers to obtain a working copy of a central repository.

* Git checkout

In addition to checking out old commits and old file revisions, git checkout is also the means to navigate existing branches. Combined with the basic Git commands, it’s a way to work on a particular line of development.

* Git pull

Pulling is the automated version of git fetch. It downloads a branch from a remote repository, then immediately merges it into the current branch. This is the Git equivalent of svn update.

* Git merge

A powerful way to integrate changes from divergent branches. After forking the project history with git branch, git merge lets you put it back together again.

Graphical user interface, application

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated( dif between public n private)

A screenshot of a computer

Description automatically generated with medium confidence(readme definition)

Graphical user interface, application

Description automatically generated(cloning)

A screenshot of a computer screen

Description automatically generated with medium confidence(->)

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated(output of #)

Graphical user interface, text

Description automatically generated

(Install git 🡪ref link to install)

(create a folder with a relevant name. Add the folders you want to add.)

(right click on the folder)

Graphical user interface, text

Description automatically generated

Graphical user interface, text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated(Git bash interface) (include the details in the git cheat sheet)

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface

Description automatically generated



Graphical user interface, text, application, website

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text

Description automatically generated with medium confidence

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, website

Description automatically generated

Graphical user interface, text, website

Description automatically generated with medium confidence