Quality of the Code

Code quality is a loose approximation of how long-term useful and long-term maintainable the code is. Code that is thrown away tomorrow: Low quality. Code that is being carried over from product to product, developed further, maybe even open sourced after establishing its value: High quality.

Here are the main attributes that can be used to determine code quality:

“

- Clarity:

  Easy to read and oversee for anyone who isn’t the creator of the code. If it’s easy to understand, it’s much easier to maintain and extend the code. Not just computers, but also humans need to understand it.

- Maintainable:

  A high-quality code isn’t over complicated. Anyone working with the code has to understand the whole context of the code if they want to make any changes.

- Documented:

  The best thing is when the code is self-explaining, but it’s always recommended to add comments to the code to explain its role and functions. It makes it much easier for anyone who didn’t take part in writing the code to understand and maintain it.

- Re-factored:

  Code formatting needs to be consistent and follow the language’s coding conventions.

- Well-tested:

  The less bug the code has the higher its quality is. Thorough testing filters out critical bugs ensuring that the software works the way it’s intended.

- Extensible:

  The code you receive has to be extendible. It's not really great when you have to throw it away after a few weeks.

- Efficiency:

  High-quality code doesn't use unnecessary resources to perform a desired action. "

So, shortly the qualities of a good code are:

- Follows a coding standard, uses linting.
- Lots of documentation.
- Low complexity.
- Unit tested.
- Easy to maintain.
- Executes quickly.
- Stress tested.
- Security scanned.
- Fuzzed

It is very rare to find code that meets all of those criteria. However when it does, the level of technical debt it ridiculously low and makes it very easy to change and maintain.

Measurements used to measure the Quality of Code

- The code works correctly in the normal case.
- The performance is acceptable, even for large data.
- The code is clear, using descriptive variable names and method names.
- Comments should be in place for things that are bit hard to understand.
- The code is broken down so that when things change, you have to change a fairly small amount of code.
- The code handles unexpected cases correctly.
- Code should not be repeated.

- The UI should feel natural: you should be able to click the things it feels like you should click, and type where it feels like you should type.
- The code should do what it says without side effects.

These are some measurements that you can use to measure the quality of a code.

Tools for maintaining the Code Quality

- Collabolator :

According to many rankings it is clearly a leader among all the tools. Collaborator is the most comprehensive peer code review tool, built for teams working on projects where code quality is critical.

- Codebrag :

It is an free and open-source tool. It is famous for its simplicity . Codebrag is used to solve issues like non-blocking code review, inline comments & likes, smart email notifications etc. What's more it helps in delivering enhanced software using its agile code review.

- Gerrit :

Gerrit is a web based code review system, facilitating online code reviews for projects using the Git version control system. Gerrit makes reviews easier by showing changes in a side-by-side display, and allowing inline comments to be added by any reviewer.

- Linters :

Used for static analysis of the source code, linters serve as primary indicators of potential issues with the code. PyLint is a popular choice for Python, while ESLint is used for JavaScript.

- Coverage.py :

This tool measures code coverage, showing the parts of the source code tested for errors. Ideally, 100% of the code is checked, but 80-90% is a healthy percentage.

- SonarQube :

  A more sophisticated analysis tool. SonarQube digs deeper into the code and examines several metrics of code complexity. This allows the developers to understand your software better.

Package Management Tools in Software Development

“A package manager or package management system is a collection of software tools that automates the process of installing, upgrading, configuring, and removing computer programs for a computer's operating system in a consistent manner. Package managers are designed to eliminate the need for manual installs and updates. “Package managers are pretty important in day-to-day software development. Package managers prevent coupling your code base with library code bases.

- It lets people get started on your project easily.

  If one of your fellow engineers at work needs to pull down a project from, say, Github, and it requires a bunch of libraries (say, a promise library like "Bluebird", a utils library like "Lodash", a string parsing library like "I") and they all should be at a specific version to make sure nothing breaks, then getting those libraries on your machine is as simple as running npm install.

- It minimizes the amount of code you track in version control.

  You shouldn't be tracking libraries of other code bases in your repository. Your source code and the source code of any libraries you're using should be maintained separately. It also unnecessarily bloats your repositories.

- It allows easier updating and usage of any libraries in your project.

  There are some methods that were added in the later versions of this utils library that aren't available in previous versions. What if I wanna use one of the new methods in a project that other engineers are also working on? Well, it's as simple as updating the version in

the ”package.json”. That, combined with good practices like making sure your local dependencies are up-to-date, makes it easier for developers to all work on the same project at the same time.

Package Management Tools

- DPKG - Debian Package Management System

  DPKG is a base package management system for the Debian Linux family, it is used to install, remove, store and provide information about ”.deb” packages.

  It is a low-level tool and there are front-end tools that help users to obtain packages from remote repositories and/or handle complex package relations .

- RPM - Red Hat Package Manager

  This is the Linux Standard Base packing format and a base package management system created by RedHat. Being the underlying system.

- Pacman Package Manager - Arch Linux

  It is a popular and powerful yet simple package manager for Arch Linux and some little known Linux distributions, it provides some of the fundamental functionalities that other common package managers provide including installing, automatic dependency resolution, upgrading, uninstalling and also downgrading software.

- Zypper Package Manager - openSUSE

  It is a command line package manager on OpenSUSE Linux and makes use of the “libzypp” library, its common functionalities include repository access, package installation, resolution of dependencies issues and many more.

- Portage Package Manager - Gentoo

  It is a package manager for Gentoo, a less popular Linux distribution as of now, but this won't limit it as one of the best package managers in Linux.

  The main aim of the Portage project is to make a simple and trouble free package management system to include functionalities such as backwards compatibility, automation plus many more.

Build Tools

Build tools are tools to manage and organize your builds, and are very important in environments where there are many projects (large scale software development), especially if they are inter-connected. They serve to make sure that where various people are working on various projects, they don't break anything. And to make sure that when you make your changes, they don't break anything either.

Build tools are usually run on the command line, either inside an IDE or completely separate from it. The idea is to separate the work of compiling and packaging your code from creation, debugging, etc. A build tool can be run on the command or inside an IDE, both triggered by you. They can also be used by continuous integration tools after checking your code out of a repository and onto a clean build machine. "make" was an early command tool used in *nix environments for building C/C++.

Typically build automation is completed with a scripting language that enables the developer to link modules and processes within the compilation process. This scripting encompasses several tasks including documentation, testing, compilation, and distribution of the binary software code.

Build automation is a crucial step in moving towards a continuous delivery model and is an important part of DevOps, or best practices to establish a more agile relationship between Development and IT Operations.

Ex:

- Jenkins :

  “It is a great tool for companies who are trying to minimize manual effort and are looking for more automatic release processes. It works great to start our regression tests, code coverage builds, or any ind of automatic ob under the roof. It is easily configurable and jobs can be easily copied and linked to GitHub repo.”

- CircleCL :

“CircleCI is perfect for a CI/CD pipeline for an app using a standard build process. It'll take more work for a complex build process, but should still be up to the task unless you need a lot of integrations with other tools. If you have a big team and can spare someone to focus full time on just the CI/CD tools, maybe something like Jenkins is better, but if you're just looking to get your app built, tested, and delivered without a huge amount of effort, CircleCI is probably your preferred tool.”

- Bamboo :

“If you value integration over cost, Bamboo is clearly the way to go. It offers tight integration to the rest of the Atlassian suite, and when you need traceability from issue to build, Atlassian is the right way to go.”

- TeamCity :

“TeamCity is well suited for an organization using continuous integration, meaning you release code to production often, and an agile project management system. There are free versions available for small teams and enterprise versions available for large teams with many different builds.”

- Travis Cl :

“If you are developing software using test driven development and want to leverage the use of your cloud platform by deploying quickly

and easily with continuous integration and continuous deployment, Travis CI is a great tool.”

Build Life Cycle & Maven

Maven is a tool to comprehend the complete state of a development effort in the shortest period of time. So what Maven does is that Making the build process easy ,Providing a uniform build system, Providing quality project information ,Providing guidelines for best practices development ,Allowing transparent migration to new features. So Maven is a project management and comprehension tool that provides developers a complete build life cycle framework.

Build life cycle Phases:

- Validate : validate the project is correct and all necessary information is available.
- Compile : compile the source code of the project.
- Test : test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed.
- Package : take the compiled code and package it in its distributable format, such as a JAR.
- Verify : run any checks on results of integration tests to ensure quality criteria are met.
- Install : install the package into the local repository, for use as a dependency in other projects locally.
- Deploy : done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.

these lifecycle's phases are some general ideas about what steps every software project needs to perform to be built, as we know, software artifacts can have many different natures and so need quite specific stuff to be done in every phase. That's what lifecycle mapping is. It is binding between lifecycle's phases and concrete

plugins' goals that is specific for artifact's nature (which means packaging in Maven world).

✧ Convention Over Configuration .

"Maven uses Convention over Configuration, which means developers are not required to create build process themselves. Developers do not have to mention each and every configuration detail. Maven provides sensible default behavior for projects. When a Maven project is created, Maven creates default project structure. Developer is only required to place files accordingly and he/she need not to define any configuration in "pom.xml." "

✧ Build Profiles.

A Build profile is a set of configuration values, which can be used to set or override default values of Maven build. Using a build profile, you can customize build for different environments such as Production v/s Development environments.

There are tree type:

- Per Project,
- Per User,
- Global.

✧ Dependency Management.

"Dependency management is a core feature of Maven. Managing dependencies for a single project is easy. Managing dependencies for multi-module projects and applications that consist of hundreds of modules is possible. Maven helps a great deal in defining, creating, and maintaining reproducible builds with well-defined classpaths and library versions."

Dependency scope is used to limit the transitivity of a dependency, and also to affect the classpath used for various build tasks.

There are 6 scopes :

- Compile ,
- Provided ,
- Runtime ,
- Test ,
- System ,
- Import .