**What is code Quality**

The Desire to create the code that is easy to understand and work with

**Why Code Quality** - our code should speak even if we are not available

1. The code we write should be understood by computers first. If they don’t, they can probably throw compile and run time errors
2. Humans (Other developers) – If they don’t understand. Its waste of time asking us about the code that we write
3. Future You – We might tend to forget after some time because we wrote some part of the code

**Problems with poor code Quality**

High maintenance cost – It’s difficult to make a change as it might break some other part of the code

Large number of defects

Frustrated end users.

According to [State of Code Review](https://smartbear.com/resources/ebooks/the-state-of-code-review-2020-report/), 40% of developers feel that workload and time constraints impact their ability to focus on the quality of the code

**Importance/Advantages of good Code Quality**

By focusing on code quality from the start:

* team can reduce the amount of time spent understanding code
* decrease the number of defects on your projects, and
* increase the likelihood of success.

**Code Quality Types**

**Functional Code Quality:**

Functional code quality means following or meeting functional requirements. It’s about what the code does. Things that ensure functional code quality include unit testing and functional testing

**Structural code Quality:**

Structural code quality means adhering to project-specific guidelines, minimizing unnecessary details, and maintaining clean code. It’s about how the code looks.

**Code Quality Metrics and Tools**

1. Complexity: We can use any of these to measure complexity
   1. IDE
   2. Command line tools
   3. Clouse based application suites – Code climate, SonarQube, Codescene
2. Code Coverage: How good is our test suite
3. Code Duplication: If a change is needed in the code. In how many places does it needs to be changed/modified
   1. Example tools are Flay CPD and simian
4. Coding style: How clean is our code and adhere to our coding standards. Hound is one such tool

**Complexity**

One way to reduce code complexity is to make sure that each method has only one responsibility. When we use the word “and” when answering the question “what does this function do” then it says the method has more than one responsibility. If an existing method is complex, then refactor it to reduce the complexity.

Following are two types of complexities.

For both:

0-5 Easy to understand

6-10 complex

10+ too complex

**Cognitive Complexity**:

Counts statements that can make code hard to understand

Examples: Complex Boolean logics, deeply nested loops

Tools: Code climate

**Cyclometric Complexity**:

It is a static analysis measure of how difficult code is to test.

Alternatively, this measure is a hint of how many distinct test cases you should write to have 100% code coverage.

**Code Coverage**

How much of code base is executed when test suite runs. Its measures the percentage. Teams should be proud if the code coverage is 85%. When we talk about tests it can be

**Unit test** – Its test that run on very small part of the program like function. Its bit easy to achieve 100% code coverage here because if the program must throw an error for input, we can create that error manually by giving the input

**Acceptance test** – Interact with the system the same way the user does. It’s difficult to reach 100% here because we might not know all the cases that user might do and sometimes the software might behave differently in different months of the year then it’s difficult to predict and implement such testing data.

**Integration test** – Used to evaluate a set of components at once.

**Code churn** - Code churn (also interchangeably known as rework) is a metric that indicates how often a given piece of code—e.g., a file, a class, a function—gets edited.

**Code Duplication:**

Many a times when a new requirement comes in. Few developers tend to write the requirement completely from scratch instead of using a part of existing functionality to finish the task as they are scared that existing functionality might break. So, in this case some part of the code is duplicated in two places. When a change comes again or two fix a bug, the developer must change in two places

Developers are recommended to be DRY – Don’t repeat yourself

**Class coupling**

It measures how many classes use other classes in it. The lower this number, the better it is. Good software design requires, low coupling and high cohesion

**Coding style:**

It’s better to have guidelines so that code can be consistent everywhere including indentation, spaces, braces, naming the variables etc.

Example tools: Flake 8

**Automated Code Analysis**

It’s a tool used to analyze structural code quality. It analyzes the program based on set of pre-defined rules and best practices via fully automated process. This can find all issues including

* Security issues
* Code duplications
* Style variations
* Other issues

This can be scalable and can be used for large number of projects

* We can conduct the automated code analysis statically without executing an application. Static analysis collects info based on the source code
* Sonar Qube is an open-source platform tool that delivers best static code analysis for languages like c++, c#, Java and many other languages
* Sonar Qube is recommended for automated analysis on every project at EPAM
* Sonar Qube is also helpful in calculating the technical debt on the project
* Most modern IDEs like Visual studio, IntelliJ IDEA can also do the static code analysis.
* Fortify is a static code analyzer that performs automated security testing
* These tools can’t understand the complex algorithms and developer’s intentions and hence can’t detect architectural or software design-related issues