**Experiment-7**

**Aim:** Define an appropriate metrics for at least 3 quality attributes for any software application of your interest?

Software quality is a vital yet often overlooked aspect of software development; in 2020, the Consortium for Information & Software Quality (CISQ) reported that the impact of poor software quality was around [$2.08 trillion](https://www.it-cisq.org/the-cost-of-poor-software-quality-in-the-us-a-2020-report.htm) to US businesses.

**What is software quality?**

The ultimate goal in software quality is that your final product has the least amount of defects possible while still maintaining the required functionality.

Software quality metrics are the tools that enable us to gauge the overall quality of your product.

There is a wide variety of software quality frameworks in the tech industry that utilize an even broader spectrum of software quality metrics.

Some frameworks emphasize large macro-level metrics, like calculating defects in a code base and other broad quantitative approaches.

A majority of these frameworks share some common characteristics that address the fundamentals of software quality and software quality metrics. Three popular frameworks we will take a look at are:

1. ISO/CISQ software quality metrics
2. AWS [Well-Architected Framework](https://aws.amazon.com/architecture/well-architected/)
3. Google’s testing emphasis for quality

CISQ updated its leaner ISO-adapted framework to outline the most vital characteristics of software quality. [CISQ has defined four primary](https://www.it-cisq.org/standards/code-quality-standards/) characteristics for an industry-standard of software quality as:

1. Reliability
2. Performance Efficiency
3. Security
4. Maintainability

Reliability

Software reliability, in a nutshell, is will the product always function as intended, additionally what is the risk that the product might fail.

Three software quality metrics that can be used to measure reliability are:

Mean Time to Failure (MTTF)

This is the time interval between two failures, which is normally measured in hours. So an MTTF of 240 would indicate that a failure is to be expected every 240 hours.

Mean Time to Repair (MTTR)

Now that you have a failure, how long does it take to fix the problem? MTTR gives you the average time it takes the team to get the feature or product back to its normal functional state.

Mean Time Between Failure (MTBF)

This is one of the most used quality metrics in reliability, MTBF is the calculated average time between failures. By adding MTTF and MTTR together, we get the total time it takes for a failure to occur plus the time it took to resolve.

### Performance Efficiency

Performance Efficiency is how well your product responds, and the time it takes to process functionality. This could mean how fast features, data, or web pages are loaded or how quickly the product responds to user input.

#### Load testing (Soak Testing)

Testing that your product can take sustained traffic and requests is integral to better understanding the performance efficiency of your product. If you were to have a significant increase in concurrent users, would all features maintain their stability?

### Security

Security is obviously a key part of any software product and most certainly for providing a high standard of software quality. Security entails both vulnerabilities in the product but also how does your team responds once a vulnerability is realized.

## Amazon Web Services software quality pillars

The AWS [Well-Architected Framework](https://aws.amazon.com/architecture/well-architected/) was first created by Amazon for running workloads in the cloud and since has been adopted as an industry standard by both Google and Microsoft. The AWS pillars are integral to creating scalable cost-effective systems to support products and resource-intensive features.

AWS defines five pillars for their cloud software quality:

1. operational excellence
2. performance efficiency
3. security
4. cost optimization
5. reliability

Fundamentally architecture software quality will have a significant emphasis on reliability, performance efficiency, and security. Since we have already discussed these characteristics, let’s go deeper into another vital category AWS has identified, cost optimization.

### ROI as a software quality metric for cost optimization

Cost optimization is a characteristic of software quality that focuses on avoiding unnecessary costs and allocating funds to the appropriate functions.

The best and most simple cost optimization quality metric is understanding the return on investment (ROI) for your software quality needs. This ensures that the software quality of a product, or specific features of a product, are getting the best bang for your buck.

When considering building out a new feature or fixing an identified bug always take into account the impact of the defect and the cost to fix the defect. Improving software quality is an investment and should be subjected to a cost-benefit analysis to fully appreciate its impact.

For example, you may have a known bug in your product that affects a small number of users. The bug has a simple workaround and is not a huge pain point for the users. The bug is living somewhere in older legacy code that would need a massive overhaul. It may be worthwhile not fixing the issue at the moment as the ROI of asking an engineer to solve the issue is negligible.

It’s important to remember that providing software quality must balance with what is economically feasible for the business.

## Google reliability testing: code coverage as a quality metric

Google has engrained testing into the heart of the development process so that every team is responsible for the quality of their products, and testers are only an external source used for creating automation.

Additionally, there is a significant emphasis for teams to commit smaller bite-size commits that can be easily rolled back if failures do arise.

One vital software quality metric widely used at Google is code coverage. Code coverage is the percentage of code that is covered by automated tests. At Google, over [90 percent of the projects](https://research.google/pubs/pub48413/) are covered with automated testing tools.

By measuring what percentage of your codebase has code coverage you can better understand the risk that failures might occur. The closer a codebase is to 100% coverage means that all code is being run through testing in one way or another.

It’s also important to note that you want quality testing, not just tests that are passing bare minimum test cases to bump your code coverage percentage.

Google uses a four-step process for almost all testing:

1. Testing by a dedicated internal testing team.
2. Crowd testing, which can be crowdsourced or a Google-based group of testers.
3. Dogfooders, Googlers who will use the product or feature in their daily work.
4. Beta testers, a small group of end-users using a pre-release version of the product or feature.

At each of these steps, the Google team is attempting to de-risk the product and increase their confidence in a lack of major failure of defects. One more note, at every stage each group is testing different parts of the product and not doubling down on the same test flows.