**Goal-Question-Metric (GQM) Approach for Setting Objectives and Questions and Measuring of them**

**Objectives**

1. Evaluate Code Quality: Decide on the architectural feasibility and code quality of the each project.

2. Analyze Developer Contribution: Determine the degree of contribution of community members, and developers to each of these projects.

3. Measure Project Impact: Understand the role of each project in its domain of functionality.

**Research Questions**

1. Out of all the projects, what is the size of code base and how does it relate to the rates of defects?

2. How many active participants are there in each of these projects, and what influence does this mode have on project advancement?

3. What are the main technologies employed in each project and their role in determining project functionality?

**Metrics used in this analysis include:**

WMC: Estimates how complex a particular class is by adding up the complexities of its methods; lower values indicating better class maintainability.

LCOM: Measures how deeply methods in a class are connected; the higher LCOM, the lower maintainability.

**Subject Programs (Data Set):**

The subject programs considered in this research include Java projects downloaded from GitHub, with certain considerations made to qualify Java projects suitable for analysis. The criteria include:

1. Minimum Size: Every project has to be at least 10000 lines of code.

2. Age: A project must be of minimum age of three years so as to prove that a project undergoes changes and is sustained in the long run.

3. Development Team: Every project should have at least three developers, so that they would maintain code quality and pay attention to maintainability.

Below is a table summarizing key attributes of the five selected projects:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Project Name | Size (LoC) | Age (Years) | Number of Developers | Description |
| Spring Framework | 175871 | 5 | 387 | A multi-subject, powerful, open-source Java platform for developing enterprise applications, the Spring Framework offers options like DI, AOP, and more and compatibility with numerous technologies. |
| Spring-boot | 151778 | 5 | 368 | Spring Boot is one of the simplest frameworks which add a great amount of performance to the Java application development because it minimizes the configuration code and provide the easy way for deploying the application. |
| Signal Android | 609700 | 10 | 256 | Signal is an encrypted messaging app designed with privacy in mind; users can experience fully encrypted Voice and video calls, messages, and file sharing. |
| Kafka | 142143 | 11 | 1215 | Geographically distributed event streaming to support large scale applications. |
| Skywalking | 163731 | 7 | 475 | Skywalking Stop worrying about Observability, tracing, and monitoring when it is this complex.. |

**Spring Framework-** The Spring Framework is a very large, publicly distributed application which is written in 175, 871 lines of code by 387 developers. For such a large-scale effort, specific issues to maintainability are rather apparent owing to the size of the effort. As the number of dependencies between the components rises, and when working with the enhanced number of modules a revision or debugging becomes more challenging as the system develops.

However, despite the above mentioned challenges, the framework is well able to maintain its ‘modular architecture’ which makes it possible to have different teams tackle different parts of the system. This modularity means that as the whole project advances each module can be updated, implemented, calibrated and otherwise modified with fairly low interference on the system. Furthermore, practice in employing design principles such as the Single Responsibility Principle (SRP) means that the classes remain reasonably clear and simple.

Other activities of importance include the utilization of automated CK tools such as the CK metrics analyzer. These tools are quite useful for getting an overall health status of the code base, helps to find potential problematic areas which could be the high coupling between objects or highly complex systems, which may need to be addressed by some refactoring.

In conclusion, the size of the Spring Framework is still something that poses certain maintainability issues as a outcome of which these issues are solved through proper design patterns and modularity and diligent supervision. This shows that the size of the program does not have any influence on maintainability as per se; this is helpful so long as the code is properly organized and managed.

**Spring Boot-** Spring Boot, with 151,778 lines of code and developers 368, is a micro framework of building Java applications since it eliminates much of the configuration and makes deployment easy. Because of its slightly smaller size as compared to the Spring Framework, it has a similar problem associated with maintainability that is typical of complex projects.

That is why, despite the fact that the Spring Boot is a small framework, it has low severity of many problems related to the large code base, due to the principle of working with defaults and conventions. What it also does is that since there is minimal requirements for elaborate setup and configuration it helps in eliminating possible errors in the application and ease management. This proposed architecture of Spring Boot is actually a more modular one where the developer gets to deal with only a sub-set of sub-systems at one given time which also reduces the cognitive load considerably and there by makes the framework much more user friendly.

However, as with just about any large project, Spring Boot could gradually become more complicated as features are built in to address the original shortcomings; problems like higher coupling and greater class complexity. The automated tools such as CK metrics are used regularly in the monitoring of maintainability of the project so that even as the size increases it will not become unmanageable.

Therefore, the size of Spring Boot impacts maintainability but the overall aim of the framework is to reduce the amount of code generated and increase readability. The correct architectural decisions that are quickly supplemented with architectural improvements and planned code quality checks guarantee that Spring Boot has remained very maintainable and highly scalable.

**Signal Android-** Signal Android, with 609700 lines of code by 256 developers provides instant messenger with emphasis on security and is an encrypted application for voice-video calls, messages, and files. Because of this, there are large maintainability issues, particularly since the app is bound to become more complex as more features are added or new security patches introduced.

The large code size hinders the identification of which other components a given component depends on, or with which other components it interacts; this makes it easy to propagate or introduce errors when modifying it. Additionally, complexity increases and when writing large sets of code as with any other large system, it usually becomes challenging to ensure that the code remains simple to understand. Still, it is really complex, and here Signal decided to implement modularization and security-driven architecture approaches because one module can be changed independently of the others without affecting the whole system.

Still, it takes code reviews and refactoring, as well as such factors in large project maintainers, as automated code health monitoring. All in all, the main maintainability issues associated with Signal’s scale rank with conventional problems in software applications of a similar staff size; however, Signal offsets these challenges with its emphasis on modular constructivism and safety measures that reduce the tensions between extra capability and keepability.

**Kafka-** with 142,143 lines of code, developed by 1215 contributors, it is a high performance, distributed event streaming plat form for large scale application. The large size and distributed scale of Kafka also presents several maintenance costs mainly pertaining to dealing with its deeply layered structure and numerous cooperating parts.

Due to our extensive use of Kafka and the big codebase it’s associated with, there is more potential for coupling which makes it harder to modify, say, a class in some way without adversely affecting other classes. There are even more contributors, and this is an important factor that needs good governance to be able to keep consistency in coding and make maintainability. Nevertheless, these problems are somewhat averted by Kafka because of its modular design and an importance given to scalability– typically, nothing precludes changing one component while leaving the others intact.

However, Kafka’s size and it scaling are classic symptoms of things that gets worse with growth, however, Kafka’s design decisions and its focus on decentralized modularity keeps Kafka sane when it grows large. Concretely, the codebase health will require constant supervision and a series of tools to be applied autonomously.

**Skywalking-** Skywalking is a cutting–edge, extensible, and distributed tracing and monitoring system with 163,731 lines of code from 475 developers. As a large system, Skywalking has traditional maintainability issues particularly in dealing with complex inter-service dependencies and managing sources that can come from a growth of tens service instances.

The large size of Skywalking also leads to problems of consistency maintenance and coupling reduction within different modules, and its application scenarios require a relatively large number of users, including the user terminal. When using a codebase, the interactions become massive, and dependencies become harder to manage since the time taken to change or debug the interactions increases as the size of the codebase increases. However, some of these problems are solved in Skywalking due to its modular architecture; this means that updates can be made to and maintenance initiated on solely individual sections of the system.

Surprisingly, the project received many community contributions and testing bots, which made it easily maintainable for its massive size. However, the scale of Skywalking introduces new challenges, these are offset by modularity and strong engagement of the community. The fact that this project demonstrates how you could be scalable and maintain the necessary level of flexibility at the same time earns this concept high points as good design and governance.

**Tool Description:**

CK-Code Metrics Tool: For obtaining C&K metric values, we utilized the CK-Code metrics tool developed for Java applications. This static analysis tool allows us to efficiently gather metrics like LCOM and WMC for the selected Java projects. The tool is available on GitHub at [CK-Code GitHub Repository] (<https://github.com/mauricioaniche/ck>).

**Results of the Analysis:**

When the CK-Code tool was applied on the selected projects the following observations were made with respect of WMC, and LCOM are expressed in form of charts or tables.

1.Bar Chart of WMC Values: Shows generally how complex classes within a projects are, and which classes are above or beneath the average complexity level.

2. Line Chart of LCOM Values: Visualizes cohesion trends, and identifies classes for which methods cooperate in contrast to those with high LCOM.

**Example observations include:**

The value of WMC at Bug Tracking System is somewhat high (equals to 15), what signifies higher complexity of the methods on the other hand, the value of LCOM equates to 0.4, meaning of this methods are well coherent.

On the other hand, an undesirable aspect is identified in the Library Management System in terms of maintainability with the LCOM value of 0.7 and WMC of 18.

**Conclusions:**

From the result analysis, it is evident that there is a significant correlation with regards to the class size and its WMC and LCOM maintainability characteristics. Classes with sizes greater than 10,000 LoC seemed to have relatively high WMC which may imply that they could be more complex and harder to maintain.” However, classes with lower LCOM values had smaller maintainability woes and this was true for all sizes of classes.

This trend raises the notion that even though size is a factor in maintainability, the internal compartmentalization of methods in classes strongly determines the level of maintainability. Future work could entail carrying out additional research on extra C&K metrics and more projects so as to reinforce these findings in other programming platforms.

**References:**

- Aniche, M. (2015). CK-Code metrics tool. Retrieved from GitHub [CK Metrics(https:>(http://github.com/mauricioaniche/ck)

- C&K Metrics Overview. General records of Carbon and Kayak metrics and their interpretations in software engineering.

- Measurement in Metrics literature of Software quality as postulated before this empirical research.

- All these projects show considerable functionality and have been kept going thus allowing us study the maintainability in relation to size.

**Description of the Tool Used:**

The tool used in Collected & Known metrics collection is called CK-Code whose source code is shared by Mauricio Aniche in GitHub. This tool analyses Java code for static analysis and computes many parameters including the WMC and the LCOM to enable the developers to quickly gauge the quality of the code and how easy or hard it would be to maintain. It is used for analysis of the software code without the presence of compiled versions for simplification of the metric collection throughout the various Java projects.