Metrics Development:
On-Premises Utilities Management Dashboard

Nicole Hessner
IP04, CS504 – Software Engineering
Masters of Science in Computer Science
School of Computing and Technology, City University of Seattle
hessnernicole@cityuniversity.edu

Executive Summary

The development, deployment, and successful adoption of an on-premises energy management system is reliant on most of the same metrics that are required of any other piece of modern software. It needs to be readily available and responsive, and simple to use and maintain, with as few bugs as possible. Maintainability is an especially important aspect of this software project, because the end user will also be taking over software maintenance activities after deployment, despite not having software specialists in their department. The software consists of several backend APIs, a frontend API, and the server-side software. The metrics that will be measured throughout the development and testing of the program and dashboard are API Availability, API Error Codes, API Response Time, API Latency, the code Maintainability Index, Cyclomatic Complexity, Depth of Inheritance, Class Coupling, Lines of Source Code, and Lines of Executable Code. All these metrics drive simplicity, elegance, and most importantly, ease of maintenance and understandability.

Keywords: software development metrics, API metrics, energy management, metrics development, maintainability, API monitoring

1. INTRODUCTION

Metrics are important drivers of software project quality, because they provide objective goals that the software needs to reach before being considered ready for release. Much like Test-Driven Design, having metrics prepared before software development begins ensures that the team has a cohesive definition of success.

The metrics for this project are split into two categories: API Metrics and Software Metrics. This is because the software consists of a backend that utilizes data from different database servers, server-side software that interprets the data, and finally an intranet web portal available to users.

For simplicity, despite several APIs being required to interact with the various backend server connections, all the server APIs will be treated to the same metrics. In practical application, these metrics should be differentiated for each database server interaction, because of the very different requirements for interfacing with a maintenance database, which updates sporadically upon user input, and the compressed air monitoring system, with live data samples every six seconds, for example (Mulesoft, n.d.).

The API metrics for this project include API Availability, API Error Code, API Response Time, and API Latency. The code metrics include the Maintainability Index, Cyclomatic Complexity, Depth of Inheritance, Class Coupling, Lines of Code, and Executable Lines of Code.

2. API AVAILABILITY

Collection Method
This metric will be collected using the self-hosted server uptime monitor Uptime Kuma, which is

available under Open-Source license on GitHub (Better Stack Team, 2022).

Purpose
The purpose of this metric is tracking the server uptime, to ensure a certain level of availability for end users.

Effect on Project Quality
The energy management dashboard is only as useful as its ability to be reliably accessed by the end users. However, since this dashboard is mission-critical to a relatively small number of users, some downtime for maintenance is acceptable on an annual basis.

3. API ERROR CODE

Collection Method
Since Errors per Minute does not seem to be a metric that is measured by Uptime Kuma, the errors will be logged and timestamped, and then analyzed and output to a custom developer dashboard for continuous monitoring. Since Uptime Kuma is open source and locally hosted, it should be possible to add a module within the Kuma dashboard itself.

Purpose
API Error Code metrics measure the number of non-200 family HTTP errors per minute. This measurement is an indication of the number of bugs and errors in the API (Gilling, 2022).

Effect on Project Quality
Tracking API error codes allows the development team to identify bugs early in the process and narrow down where in the API code to look for the bugs based on the most frequent error types.

4. API RESPONSE TIME

Collection Method
API Response Time will be collected using Uptime Kuma, as mentioned in Section 3.

Purpose
Response Time measures Time to First Byte, starting from the time the user makes an initial request. This does not include any time it takes to process or render data client-side, but does include server-side data processing time. It consists of DNS lookup time, authentication and connection time, redirect time, time to first byte, and time to last byte (Sematext Group, 2022).

Effect on Project Quality
Focusing on a lower response time will improve the user experience, and ensure that servers are load balanced correctly. The response time from the server should be less than 200ms to feel instantaneous (Sematext Group, 2022).

5. API LATENCY

Collection Method
API Latency will be collected using Uptime Kuma, as mentioned in Section 3.

Purpose
API Latency measures the lag between a call to the server and its response. A lower latency is desired, because it means that the user can access the content they are expecting more quickly, since the initial communication to the server occurred more quickly (Gilling, 2022).

Effect on Project Quality
Since implementation of the energy management dashboard is expected to have some pushback within certain user classes, a more responsive API will aid in ease of adoption; in my experience, users are more likely to shy away from and complain about an application that is too slow, adding to an existing cultural bias against technological solutions within that user class. It is important to measure both latency and response time, because that allows differentiation between a server communication issue and a code processing time issue. To be considered real-time, the latency of the API must be less than 30ms (Mauro, 2020).

6. MAINTAINABILITY INDEX

The Maintainability Index of this project should be at least 20, since that is considered "green" by the Microsoft Visual Studio Code team. However, the team should aim for as high a value as possible once that threshold is met (Jones & Hogan, 2022).

Collection Method
Since this project will be developed using Microsoft Visual Studio Code, the development team will utilize the built-in Maintainability Index calculation feature. The formula used to calculate the Maintainability Index is shown in Equation 1. Since the Maintainability Index is depending on the Halstead Volume, Cyclomatic Complexity, and Lines of Code, those three metrics are obvious additional choices to measure and will be discussed in later sections (Jones & Hogan, 2022).

Equation 1 – Maintainability Index formula (Jones & Hogan, 2022)

$$Maintainability\ Index = MAX(0, (171 - 5.2 * \ln(Halstead\ Volume) - 0.23 * (Cyclomatic\ Complexity) - 16.2 * \ln(Lines\ of\ Code)) * \frac{100}{171})$$

Purpose
The purpose of the Maintainability Index is to provide a relative measure of the ease of maintaining the code. Code is easier to maintain when it has fewer lines, elements, and flow control paths (Jones & Hogan, 2022).

Effect on Project Quality
Focusing on the Maintainability Index helps to ensure that the program will be split into easy-to-understand modules that will be accessible for the maintenance team, and easy to understand for someone who may need to update the code in the future and was not part of the original project team. Its inverse relationship to Halstead Program Volume, Cyclomatic Complexity, and Lines of Code drives good performance in those three metrics, as well (Jones & Hogan, 2022).

7. CYCLOMATIC COMPLEXITY

Collection Method
M is the Cyclomatic Complexity as calculated by Equation 2, where E is the number of edges in the control flow graph, N is the number of nodes in the control flow graph, and P is the number of connected components (GeeksForGeeks, 2021).

Equation 2 - Cyclomatic Complexity formula (GeeksForGeeks, 2021)

$$M = E - N + 2P$$

Purpose
According to GeeksForGeeks (2021), "Cyclomatic complexity of a code section is the quantitative measure of the number of linearly independent paths in it." The development of a Cyclomatic Complexity metric necessitates the development of a flow control graph with nodes and edges, and the identification of independent paths, which in turn drives the design of test cases.

Effect on Project Quality
A reduction in Cyclomatic Complexity directly improves the Maintainability of the program, and also reduces the number of independent tests required to fully test all the command paths (GeeksForGeeks, 2021). A Cyclomatic Complexity of 4 is considered good, while 5 to 8 is medium complexity, 8 to 10 is high complexity, and anything greater than 10 is extreme complexity (Savage, 2013).

8. DEPTH OF INHERITANCE

Collection Method
This metric will be calculated utilizing the functionality of Visual Studio Code, as with the other class metrics.

Purpose
The Depth of Inheritance is the maximum length of a class to a root class in an object-oriented system. It measures how many different classes affect the most derived class. The depth of classes influences most aspects of a program, sometimes positively and sometimes negatively (Arisa).

Effect on Project Quality
While higher depth of inheritance can have some positive effects on software, the likelihood of unfound bugs increases and the ability of developers to learn the class code and effectively reuse it decreases. Since depth of inheritance negatively affects maintainability, the development team should target a depth of no more than 5 or 6, since the program may be maintained by a local team member who may not specialize in coding (Jones, Hogenson, & v-rajagt-zz, 2022).

9. CLASS COUPLING

Collection Method
Much like the other class metrics utilized for this project, the Class Coupling metric will be automatically calculated utilizing the functionality of Visual Studio Code.

Purpose
Class Coupling measures the degree to which classes are dependent on each other. A class that calls another class within its definition will increase the Class Coupling metric by one for each class called. An upper-limit value of 9 is optimal, after which class coupling is considered excessive. (Jones, Hogenson, and Ovhal, 2022).

Effect on Project Quality
According to Jones, Hogenson, and Ovhal (2022), "Class coupling has been shown to be an accurate predictor of software failure." By utilizing this metric, it ensures that the code being developed maintains an acceptable level of class coupling. While the maximum acceptable amount of class coupling is 9, the goal of the developers is to minimize this metric.

10. LINES OF SOURCE CODE

Collection Method
This is just a pure line count of the code.

Purpose
"Lines of Code or LOC (also known as Source Lines of Code - SLOC) is a quantitative measurement in computer programming for files that contains code from a computer programming language, in text form. The number of lines indicates the size of a given file and gives some indication of the work involved" (Atlassian, 2018).

Effect on Project Quality
Lines of Code is less of a reliable indicator of success or failure as an isolated metric, but rather is necessary for calculating the maintainability. Since fewer lines of code contribute to a higher maintainability score, the development team should seek the most elegant solution with the fewest lines of code (Atlassian, 2018).

11. LINES OF EXECUTABLE CODE

Collection Method
This metric will be collected using the Visual Studio Code metric generation tools.

Purpose
Lines of Executable Code is very closely related to Lines of Code, except that it does not count comments and white space, and counts each operation as a separate line (Vasani, 2019).

Effect on Project Quality
Aiming to minimize executable lines of code will reduce memory requirements and program execution time, which will improve server response time and reduce program complexity. This will make it easier for the customer's on-site program maintenance team to maintain and update the code, since it will be simpler to understand.

12. CONCLUSION

In conclusion, the target metrics for this project are contained in Table 1, below.

Table 1 - Summary of metrics and associated targets.

Metric	Value
API Availability	99%
API Error Code	All error codes captured and logged correctly
API Response Time	≥200ms
API Latency	≥30ms
Maintainability Index	≥20
Cyclomatic Complexity	≤4
Depth of Inheritance	≤5
Class Coupling	≤9
Lines of Source Code	Minimize
Lines of Executable Code	Minimize

The overall goal for the metrics of this energy management dashboard is to drive simplicity, maintainability, and a reliable and instantaneous user experience. This is especially important since the source code will be maintained by the end user after launch, and the department that will use the software does not have dedicated coders.

13. REFERENCES

Arisa. (2009). Depth of Inheritance Tree ( ). Arisa.Se. Retrieved June 5, 2022, from http://www.arisa.se/compendium/node101.html#:%7E:text=Depth%20of%20Inheritance%20Tree%20(DIT,applicable%20to%20object%2Doriented%20systems

Atlassian. (2018, October 25). About the Lines of Code Metric | Fisheye Server 4.8. Atlassian Documentation. Retrieved June 5, 2022, from https://confluence.atlassian.com/fisheye/about-the-lines-of-code-metric-960155778.html#:%7E:text=Lines%20of%20Code%20or%20LOC,indication%20of%20the%20work%20involved

Better Stack Team. (2022, May 4). *7 Best Open-Source Website Monitoring Tools in 2022*. Better Stack Community. Retrieved June 5, 2022, from https://betterstack.com/community/comparisons/open-source-website-monitoring/

GeeksforGeeks. (2021, June 21). *Cyclomatic Complexity*. Retrieved June 5, 2022, from https://www.geeksforgeeks.org/cyclomatic-complexity/

Gilling, D. (2022, January 26). *13 API Metrics That Every Platform Team Should be Tracking*. Moesif Blog. Retrieved June 5, 2022, from https://www.moesif.com/blog/technical/api-metrics/API-Metrics-That-Every-Platform-Team-Should-be-Tracking/

Jones, M., & Hogenson, G. (2022, April 30). *Code metrics - Maintainability index range and meaning - Visual Studio (Windows)*. Microsoft Docs. https://docs.microsoft.com/en-us/visualstudio/code-quality/code-metrics-maintainability-index-range-and-meaning?view=vs-2019

Jones, M., Hogenson, G., & Ovhal, P. (2022, April 30). *Code metrics - Class coupling - Visual Studio (Windows)*. Microsoft Docs. Retrieved June 5, 2022, from https://docs.microsoft.com/en-us/visualstudio/code-quality/code-metrics-class-coupling?view=vs-2022

Jones, M., Hogenson, G., & V-rajagt-zz. (2022, April 30). Code metrics - Depth of inheritance - Visual Studio (Windows). Microsoft Docs. Retrieved June 5, 2022, from https://docs.microsoft.com/en-us/visualstudio/code-quality/code-metrics-depth-of-inheritance?view=vs-2022

Mauro, T. (2020, October 2). Is Your API Real Time? Test Its Latency with the rtapi Tool from. NGINX. Retrieved June 5, 2022, from https://www.nginx.com/blog/api-real-time-test-latency-responsiveness-nginx-rtapi-tool/#:%7E:text=For%20your%20API%20to%20be,to%20be%20less%20than%2030ms.

MuleSoft. (n.d.). What is an API? (Application Programming Interface). Retrieved June 5, 2022, from https://www.mulesoft.com/resources/api/what-is-an-api

Savage, B. (2013, May 23). Code complexity and clean code. BrandonSavage.Net. Retrieved June 5, 2022, from https://www.brandonsavage.net/code-complexity-and-clean-code/#:%7E:text=For%20most%20routines%2C%20a%20cyclomatic,above%20that%20is%20extreme%20complexity.

Sematext Group. (2022, April 21). What Is Response Time & How to Reduce It. Sematext. Retrieved June 5, 2022, from https://sematext.com/glossary/response-time/#:%7E:text=A%20web%20response%20time%20ranging,and%20needs%20to%20be%20fixed.

Vasani, M. (2019, September 23). *Update documentation for feature enhancements in VS2019 16.4 and Microsoft.CodeAnalysis.Metics (2.9.5) - both are now released · Issue #3979 · MicrosoftDocs/visualstudio-docs*. GitHub. Retrieved June 5, 2022, from https://github.com/MicrosoftDocs/visualstudio-docs/issues/3979