REPUBLIC OF TURKEY
YILDIZ TECHNICAL UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

3D VISUALIZATION OF OBJECT ORIENTED SOFTWARE METRICS

20011079 — ERAY GÖKÇE

SENIOR PROJECT

Advisor
Asst. Prof. Dr. Yunus Emre SELÇUK

June, 2025

ACKNOWLEDGEMENTS

I would like to thank our esteemed faculty member Asst. Prof. Dr. Yunus Emre SELÇUK and Yıldız Technical University Computer Engineering Department for the valuable time they spent and the support they provided to me.

ERAY GÖKÇE

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

API	Application Programming Interface
BCEL	Byte Code Engineering Library
CBO	Coupling Between Object Classes
CK Metrics	Chidamber and Kemerer Metrics
CPU	Central Processing Unit
CSV	Comma-Separated Values
DFDs	Data Flow Diagrams
DIT	Depth of Inheritance Tree
GPU	Graphics Processing Unit
GUI	Graphical User Interface
JDK	Java Development Kit
JSON	JavaScript Object Notation
MVC	Model-View-Controller
NOC	Number of Children
REST	Representational State Transfer
TXT	Text File
UML	Unified Modelling Language
WMC	Weighted Methods per Class
XML	eXtensible Markup Language

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

3D Visualization of Object Oriented Software Metrics

ERAY GÖKÇE

Department of Computer Engineering
Senior Project

Advisor: Asst. Prof. Dr. Yunus Emre SELÇUK

This project was implemented to evaluate the structural quality of object-oriented software systems. It aims to analyze software metrics. These metrics are visualized in a three-dimensional environment to make them easier to interpret. Previously developed tools produce numerical outputs for metric analysis. These outputs make it difficult to intuitively identify design errors at the class level.

At this point, the visualization of software metric numerical data offers significant advantages in identifying architectural problems and class-level density in complex software systems. Various metrics are calculated from the compiled .class files of projects written in Java. Analysis was performed on the bytecode for the metric calculation algorithms. At this point, the Byte Code Engineering Library was used. Metric results are stored in a database. This allows developers to compare the project with previous versions. During the visualization phase, developers can select three metrics.

The calculated metrics are represented in a three-dimensional city metaphor in the Unity game engine. Each class is modeled as a structure. Properties such as color, width, and height will reflect the metric results. This allows the user to interpret the project not only through numerical data but also visually.

The project is designed with a multi-layered architecture. The architecture consists of three basic layers. These layers are designed as backend, frontend, and visualization (Unity). The Spring Boot framework is used on the backend side. By creating services

with REST API, metric calculation algorithms are performed in the service layer. The frontend was designed and implemented using React.js. The user interface performs operations such as file uploading and viewing past analyses. The Unity side listens to requests from the backend. When the user requests visualization, data is sent to the Unity side in JSON format, and visualization is performed. The JSON data is processed and reflected on the scene as 3D objects.

Application tests have shown that the project can be successfully analyzed in projects of different sizes. Verifications of quality factors such as cohesion, complexity, and dependency prove that the project works reliably and consistently. The database feature, which was added later as an additional feature to the project, has become an effective feature that allows past analyses to be reloaded.

In conclusion, this project ensures that the structural integrity of software projects is presented in a more understandable, interactive, and intuitive manner by visualizing various software metrics. It will make an important contribution to increasing sustainability and quality in large-scale software projects.

Keywords: Software Metrics, Object-Oriented Software Design, 3D Visualization, Software Quality Assessment, Bytecode Analysis, Software City Metaphor

ÖZET

Nesne Yönelimli Yazılım Metriklerinin 3 Boyutlu Görselleştirilmesi

ERAY GÖKÇE

Bilgisayar Mühendisliği Bölümü
Bitirme Projesi

Danışman: Dr. Öğr. Üyesi Yunus Emre SELÇUK

Bu proje, nesne yönelimli geliştirilen yazılım sistemlerinin yapısal kalitesini değerlendirmek amacıyla gerçeklenmiştir. Yazılım metriklerinin analinizi hedefler. Bu metriklerin daha kolay yorumlanabilmesi adına üç boyutlu ortamda görselleştirilmesi amaçlanmıştır. Daha önceden gerçekleştirilen araçlar, metrik analizlerini sayısal çıktılar üretmektedir. Bu çıktılar, sınıf seviyesindeki tasarım hatalarını sezgisel olarak fark edilmesini zorlaştırmaktadır. Bu noktada, yazılım metrik sayısal verilerinin görselleştirilmesi; karmaşık yazılım sistemlerinde oluşan mimari problemleri ve class düzeyindeki yoğunluğu tespit etmek için önemli avantajlar sunmaktadır.

Java ile yazılmış projelerin hali hazırda derlenmiş .class dosyaları üzerinden çeşitli metrikler hesaplanmaktadır. Metrik hesaplama algoritmaları için bytecode üzerinden analiz gerçekleştirilmiştir. Bu noktada , Byte Code Engineering Library kullanılmıştır. Metrik sonuçları veritabanına kaydedilmektedir. Bu sayede, geliştirici projesinin daha önceki zamanlara kıyaslayabilecektir. Geliştirici, görselleştirme aşamasında üç metrik belirleyebilmektedir.

Hesaplanan metrikler, Unity oyun motorunda üç boyutlu bir şehir metaforu içerisinde temsil edilmektedir. Her sınıf birer yapı olarak modellenmiştir. Renk, genişlik ve yükseklik gibi özellikler metrik sonuçlarını yansıtacaktır. Bu sayede kullanıcı sadece sayısal veriler üzerinden değil, aynı zamanda görsel olarak da projesini yorumlayabilecektir.

Proje çok katmanlı mimari olarak tasarlanmıştır. Mimari üç temel katmandan oluşmaktadır. Bu katmanlar arka uç (backend) , kullanıcı arayüzü (frontend) ve görselleştirme(unity) şeklinde tasarlanmıştır. Backend tarafında Spring Boot framework'ü kullanılmıştır. REST API ile servisler oluşturarak, metrik hesaplama algoritmaları servis katmanında gerçekleştirilmektedir. Frontend, React.js ile tasarlanıp gerçekleştirilmiştir. Kullanıcı arayüzünde dosya yükleme, geçmiş analizleri görüntüleme gibi işlemler gerçekleştirilmiştir. Unity tarafı backend tarafından istek dinlemektedir. Kullanıcı, görselleştirme istediğinde Unity tarafına JSON ile veri yollayarak görselleştirme gerçeklenir. JSON verisi işlenerek sahneye 3D objeler olarak yansıtılmaktadır.

Uygulama testleri sonucunda projenin farklı boyutlardaki projelerde başarılı şekilde analiz edildiğini göstermiştir. Kohezyon, karmaşıklık ve bağımlılık gibi kalite faktörleri üzerinde yapılan doğrulamalar, projenin güvenilir ve tutarlı olarak çalıştığını kanıtlamaktadır. Projeye ek özellik olarak sonradan eklenen veritabanı özelliği sayesinde, geçmiş analizlerin yeniden yüklenebilmesi etkili bir özellik olmuştur.

Sonuç olarak, bu proje yazılım projelerinin yapısal bütünlüğünü çeşitli yazılım metriklerini görselleştirerek daha anlaşılır, etkileşimli ve sezgisel şekilde sunulmasını sağlamaktadır. Büyük ölçekli yazılım projelerinde sürdürülebilirliği ve kaliteyi arttırmaya yönelik önemli bir katkı sunacaktır.

Anahtar Kelimeler: Yazılım Metrikleri, Nesneye Yönelik Yazılım Tasarımı, 3D Görselleştirme, Yazılım Kalite Değerlendirmesi, ByteCode Analizi, Yazılım Şehri Metaforu

1
INTRODUCTION

Today, there are numerous software projects designed to serve specific purposes. These projects are initially designed to fulfill the desired and requested functionalities. After delivery, the customer begins using the software, and over time, new requirements emerge. At this point, developers begin working to add new features to the existing project. However, numerous issues may arise during this process. If the initial design of the software was not developed with future expansions in mind, the code can become complex and difficult to manage. In the field of Software Engineering, the design, maintenance, and evolution of large-scale software systems constitute an important area of research and application. As software systems grow and evolve, maintaining quality becomes a significant challenge [1]. Projects with expanding scopes often become more complex and fragile, making maintenance and restructuring processes time-consuming and costly. At this point, it is particularly important to emphasize Object-Oriented Programming. In such software systems, elements such as inter-class relationships, method and variable distribution, and inheritance structures significantly affect software quality. To measure these structural characteristics, Chidamber and Kemerer proposed a set of metrics widely adopted for object-oriented design (WMC, DIT, LCOM, RFC, etc.) [2, 3].

At this point, it is clear that software systems need to be defined using specific metrics. Software metrics are tools that quantitatively express the structural characteristics of a system and play an important role in characterizing and evaluating the system. Based on these metrics, developers can analyze the overall structure of the program. However, interpreting these metrics solely as numerical values is often insufficient. Visualizing metrics provides a powerful method for making software systems more understandable and quickly identifying design flaws. Lanza and Ducasse [4] introduced the concept of "polymetric views," which enable the visual representation of structural metrics such as class size, cohesion, and complexity. When these visualized metrics are examined by the human eye, they can be interpreted more effectively from a holistic perspective. In recent years, 3D visualization approaches

using the “software city” metaphor have gained popularity. Wettel and Lanza [5] proposed the CodeCity system, which renders software components as buildings in a virtual city. In this system, visual features (e.g., height and color) represent different metric values. This approach helps identify code smells and architectural flaws early on by visualizing the software architecture in an overhead and intuitive manner.

1.1 Objective

The main objective of this project is to analyze software metrics used to evaluate the design of object-oriented software systems and visualize the resulting metric data in a three-dimensional format. The analysis will be performed on the bytecode of projects developed using the Java programming language.

Within the scope of the project, at least one new metric will be proposed in addition to the metrics currently in use, thereby increasing the system’s metric-based evaluation capacity. This new metric will be integrated into the existing measurement tool. The current software displays metrics using an Excel-based graphical user interface. One of the main goals of our project is to present these metrics in a three-dimensional environment. The metrics will be visualized using engines that create a city-like graphical landscape [5, 6]. This will allow the structure of the software system to be analyzed holistically in an aerial and intuitive manner, resembling an urban settlement plan. This visualization approach will enable developers to identify complex classes and potential design issues earlier and more effectively. As a result, it will provide a guiding perspective for maintenance and restructuring processes.

1.2 Preliminary Review

The project includes a previously developed Java application. This application performs software quality measurements on .class and .jar files using CK metrics commonly used in object-oriented programming [2]. The current system provides a graphical user interface that allows users to select files and export the results in various formats. The system architecture is structured in accordance with the MVC design pattern.

As a result of the software review, it was found that byte code analysis was performed using BCEL (Byte Code Engineering Library) [7]. The results obtained can be saved in .xml, .csv, or .txt formats.

As part of this project, a new metric will be integrated into the existing system, thereby increasing the system’s metric-based evaluation capabilities. Subsequently, numerical

metric data will be visualized using a graphics engine.

2
SYSTEM ANALYSIS AND FEASIBILITY

2.1 System Analysis

The purpose of system analysis is to identify and define the basic components and functions in order to determine the most appropriate solution for the project. In this section, the objectives of the project are detailed, and information sources and system requirements are presented. At the end of the section, the goal is to clearly define all requirements related to the project and determine the most appropriate solution for transitioning to the design phase.

The research and data collection methods used to determine these requirements (e.g., interviews, surveys, etc.) and the results obtained should be presented in this section. System modules, users, and roles should be defined based on the identified requirements. The requirements analysis model can be prepared using a functional or object-oriented approach.

If a functional approach is preferred, workflows explaining how the system will work and data flow diagrams belonging to the analysis model should be prepared. These data flow diagrams should be detailed up to the second level, and system modules should be analyzed. When an object-oriented approach is adopted, requirements should be determined through use case analysis, and the analysis model should be created using a conceptual class diagram.

As a result of the system analysis, a framework will also emerge that allows testing whether the final project meets the specified needs. Therefore, the performance metrics to be used in the project (e.g., speed, accuracy, number of features, etc.) should also be defined in this section.

This project is an application that evaluates the design of object-oriented software systems through software metrics and visualizes these metrics in a three-dimensional environment [6]. System administrators or developers will be able to upload their projects to this application and examine their metric scores with a three-dimensional

representation. Additionally, it will provide significant advantages during application development, debugging, and maintenance phases.

- **Project Selection Interface:** Users will be able to select the project they want to analyze using the file browser.
- **Metric Results Display Interface:** The results of all metrics, including newly added metrics, will be displayed.
- **Visualization:** These metrics will be represented through structures (buildings) within a visualization engine. Users will be able to explore metric results in an environment with a virtual city view.

2.2 Requirements Analysis

This section summarizes the functional and non-functional requirements necessary for the implementation of the project. The requirements are defined to cover all software and hardware components necessary for the successful operation of the system.

2.2.1 Functional Requirements

- The system will perform bytecode-level analysis for Java-based projects.
- Users will be able to select the project to be analyzed through the user interface.
- The system will calculate existing metrics (e.g., WMC, DIT, NOC, CBO, etc.).
- The three metrics selected by the user will be visualized as three-dimensional rectangular prisms for each class.
- Users will be able to map each metric to a specific visual dimension (height, width, color intensity).
- A new metric can be integrated into the system, and the system will be able to calculate and visualize this metric.
- The visualization will be presented interactively through a graphics engine such as OpenGL, QT, or Unity; it will support operations such as zooming, rotating, and selecting [8].
- During visualization, clicking on any prism/class will display basic information about that class (e.g., class name, metric values).

2.2.2 Non-Functional Requirements

- The system must perform analysis and visualization tasks within a reasonable amount of time.
- The application must run reliably on Java projects of different sizes.
- The user interface must be intuitive, user-friendly, and easy to navigate.

2.3 Feasibility

This section details the feasibility analyses related to the implementation of the application for the visualization of software metrics.

2.3.1 Technical Feasibility

This project is based on the measurement of software metrics obtained from object-oriented software systems and the visualization of these metrics in a three-dimensional environment. The software projects to be analyzed will be developed using the Java programming language, and metric evaluations will be performed on the compiled byte code.

For byte code analysis, the Apache BCEL library, which provides access to the internal structure of Java classes, will be used. The metric data obtained as a result of this analysis will be presented visually through 3D shapes representing software classes.

During the visualization phase, each selected metric will be mapped to visual properties such as the width, height, or color intensity of class representations, thereby creating a virtual “class neighborhood.” For this purpose, the Unity game engine will be used as the 3D graphics platform.

2.3.1.1 Hardware Feasibility

In this project, the analysis of Java projects and the extraction of software metrics will be performed on compiled bytecode. Bytecode analysis involves lightweight operations running on the CPU, so it does not require high processing power. Therefore, a mid-range computer will be sufficient for this phase.

However, the metrics obtained in the second phase of the project will be presented using three-dimensional visualization. Graphics engines such as Unity operate more efficiently with GPU support. Especially in scenes containing a large number of objects,

GPU performance becomes critical. In this context, meeting the minimum hardware requirements for the visualization phase is important.

Estimated Minimum Hardware Requirements:

- **Processor (CPU):** Intel i5 or equivalent AMD processor
- **Memory (RAM):** 8 GB (preferably 16 GB)
- **Storage:** SSD recommended; approximately 1–2 GB of space is required for the project
- **Graphics Card (GPU):** NVIDIA GTX 1050 or higher; performance limitations may occur on systems with integrated GPUs

In addition, a system with DirectX or OpenGL support is recommended for Unity to work optimally. If the system specifications are below the recommended thresholds, performance drops or delays in rendering 3D models may occur during the visualization process.

2.3.1.2 Software Feasibility

The software tools, libraries, and development environments required for the successful development of the project consist of widely used, open-source, or freely accessible technologies. This significantly increases the software feasibility of the project.

The software systems to be analyzed within the scope of the project have been developed using the Java programming language. In this regard, Apache BCEL has been chosen as a bytecode analysis library compatible with Java. Apache BCEL is a powerful tool that facilitates the extraction of software metrics by reading and interpreting Java class files.

The Unity game engine will be used to visualize the obtained metrics in three dimensions [9]. Unity was chosen for its ability to produce visually satisfying results and for facilitating the association of metrics with geometric shapes.

The basic software tools to be used in the development process are as follows:

- Java Development Kit (JDK 8 or higher)
- Eclipse IDE or IntelliJ IDEA (Development environment)

- Apache BCEL (Bytecode analysis library)
- Unity (3D visualization engine)

2.3.2 Legal Feasibility

The majority of software components used in this project are distributed under open source licenses. Since these licenses generally allow individual or academic use and development, there are no legal obstacles to the implementation of the project.

Additionally, since no personal data or proprietary information will be processed within the scope of the project, there are no concerns regarding data privacy or information security.

In conclusion, all technologies used in this project are protected by legally compliant licenses, and there are no legal restrictions that would prevent the project from being carried out.

2.3.3 Economic Feasibility

The project is economically feasible as it is developed primarily using open-source and free tools. All software components to be used are either available free of charge or provided under open-source licenses, eliminating software licensing costs.

The Java development tools (JDK, Eclipse IDE) and the Apache BCEL library to be used in the analysis phase are open-source. Similarly, the Personal Edition of Unity, which is offered for individual and academic projects, does not require any license fees.

In terms of hardware, while metric calculations do not require high-performance computing resources, the 3D visualization phase using Unity may require a mid-range GPU. If the personal computer does not meet these performance requirements, there may be a potential cost for upgrading the graphics card.

In summary, the project's software and hardware requirements can be met with low-cost solutions, making it economically feasible.

2.3.4 Time Feasibility

The project schedule has been carefully structured to ensure that all stages, from requirements analysis to final adjustments, are completed within approximately 2–3 months. This schedule is presented in Figure 2.1.

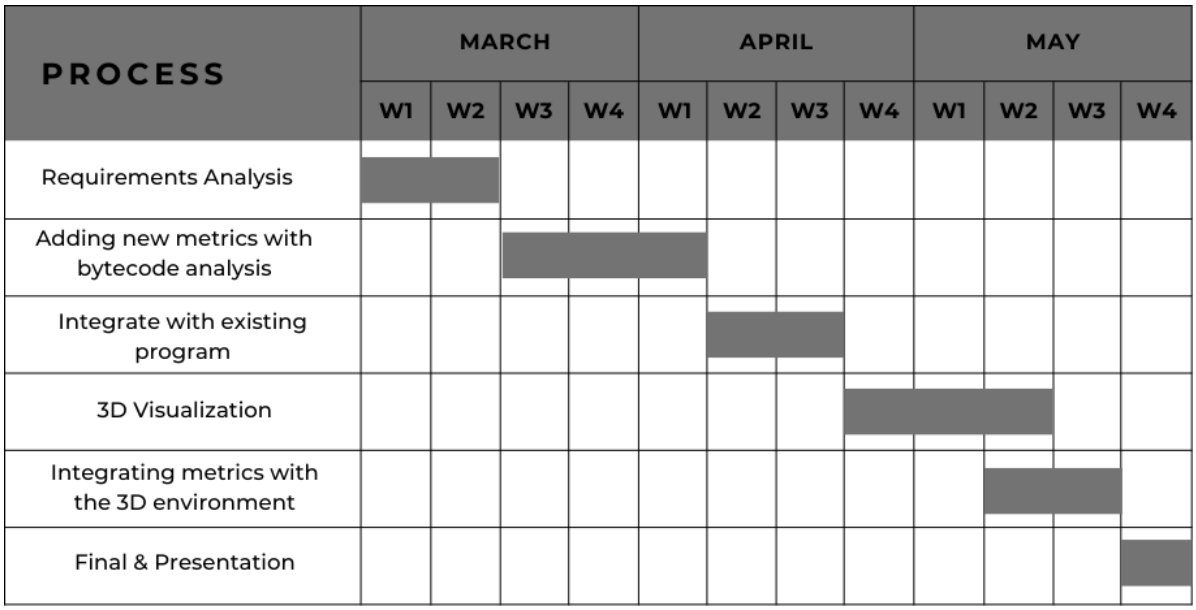

Figure 2.1 Gantt Chart

3
SYSTEM DESIGN

In this section, the system design of the project is explained under the following subheadings.

3.1 Use Case Diagram

Use Case diagrams are behavior diagrams that model the functions of a software system, the actors responsible for these functions, and the interactions between the actors and the system. The use case diagram for the project is presented in Figure 3.1.

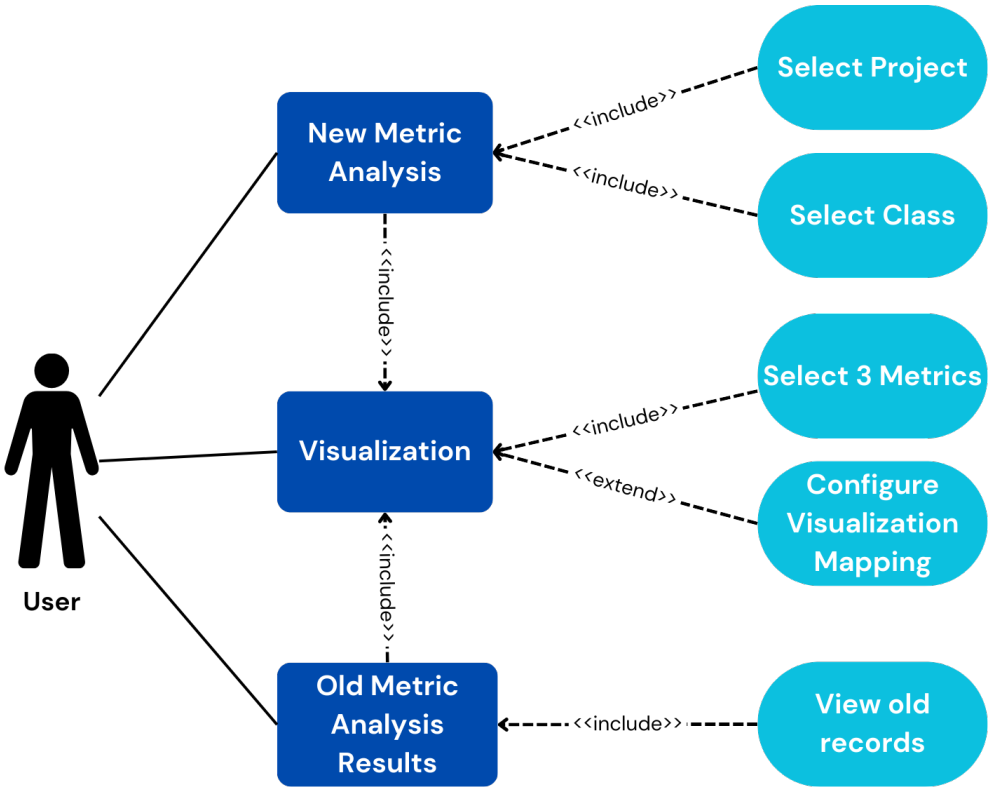

Figure 3.1 Use Case Diagram

There are three main functional structures in the project. When the user wants to perform a new metric analysis, the system allows the user to select a project folder

or a specific class file. Based on the calculated metric results, the system provides the user with visualization capabilities. Additionally, thanks to the database structure, the user can view previously performed analysis results and visualize these historical records. This enables the user to compare the past state of a project with its current state.

3.2 Use Case Scenario

Use case scenarios are descriptions that define the interactions between the software system and users step by step and explain the operation of the system with example situations. Two basic use cases for the project are shown in Figure 3.2 and Figure 3.3.

Usage Scenario	Start New Analysis
Primary Actor	User
Interested Parties and Their Expectations	User : Calculation and visualization of software metrics of the project he/she is interested in.
Prerequisites	It must exist in user, project or class files.
Final Conditions	Project or class files have been calculated and visualized successfully.
Main Scenario	1. The user starts a new metric analysis. 2. The user selects the project they want to analyze. 3. The user selects three metrics they want to visualize from the metrics calculated by the system. 4. The user displays the visualization environment on the screen where the classes are represented according to the selected metrics.
Alternative Scenario	2a. Instead of a project, the user can start an analysis specific to just this class by selecting a single .class file. 3a. The user can manually specify which dimension in the visualization the metrics they select will correspond to.

Figure 3.2 Usage Scenario 1

Usage Scenario	Viewing Old Analyses
Primary Actor	User
Interested Parties and Their Expectations	User : The user wants to view the project for which he/she previously calculated and visualized metrics.
Prerequisites	A project or class must have been visualized before and must be registered in the database.
Final Conditions	Project or class files have been calculated and visualized successfully.
Main Scenario	1. The user lists the old records. 2. The user selects the project he wants to view from the records. 3. The user selects three of the calculated metrics that he wants to visualize. 4. The user displays the visualization environment on the screen where the classes are represented according to the selected metrics.
Alternative Scenario	-

Figure 3.3 Usage Scenario 2

3.3 Object-Oriented Design

The UML diagram for the software classes used in the project is presented below.

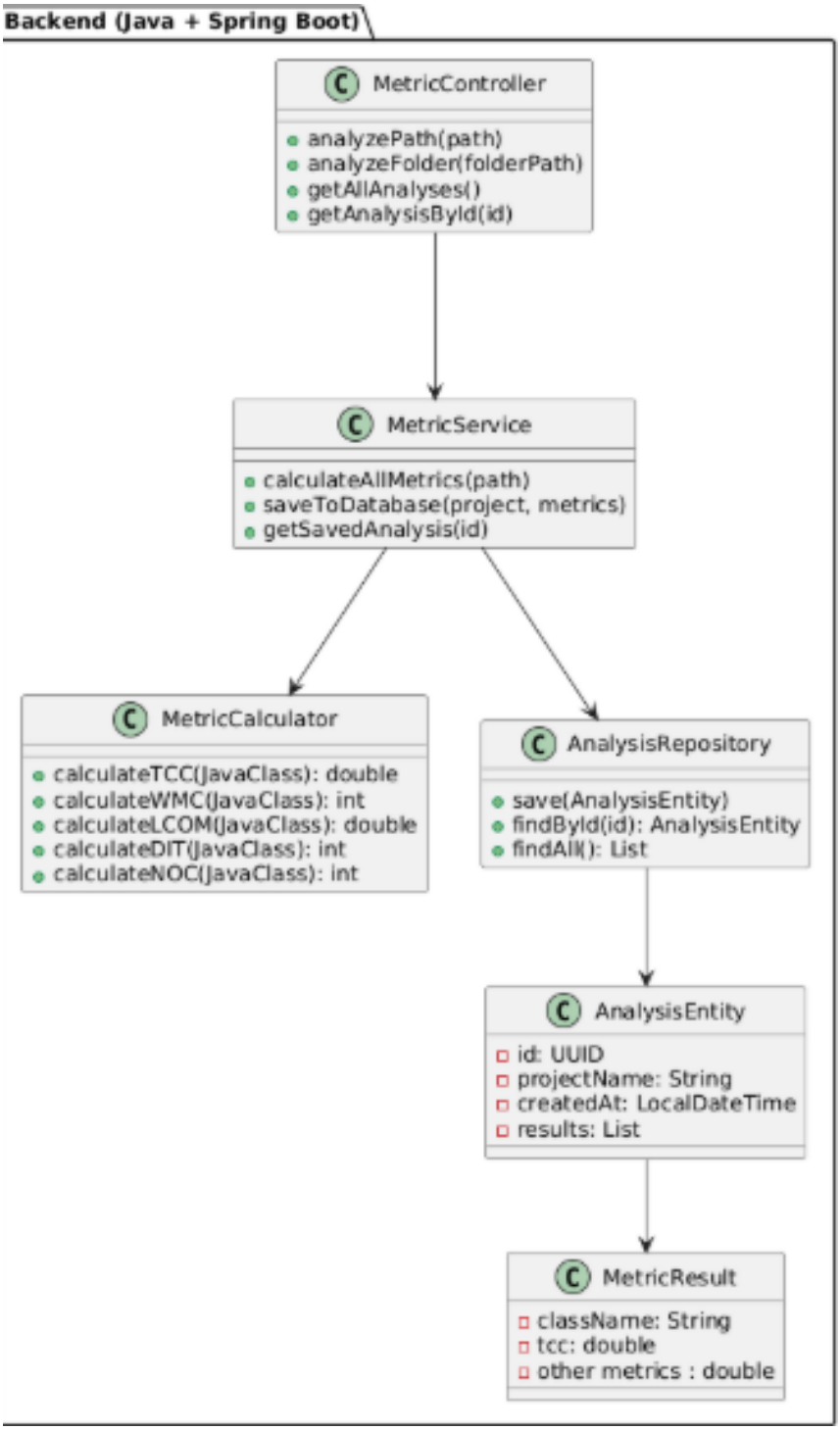

Figure 3.4 Java (Backend) UML Diagram

As shown in Figure 3.4, the API structure that communicates with Unity has been built using the Spring Boot architecture. The 'MetricsController' class contains REST endpoints that respond to requests from the user. When the user wants to start a new project or class analysis or view previous records, these operations are directed

to the ‘MetricService‘ class through the control layer. Metric calculations and database operations are performed through the ‘MetricService‘ and ‘AnalysisRepository‘ classes. This API transmits the analysis results to Unity in JSON format.

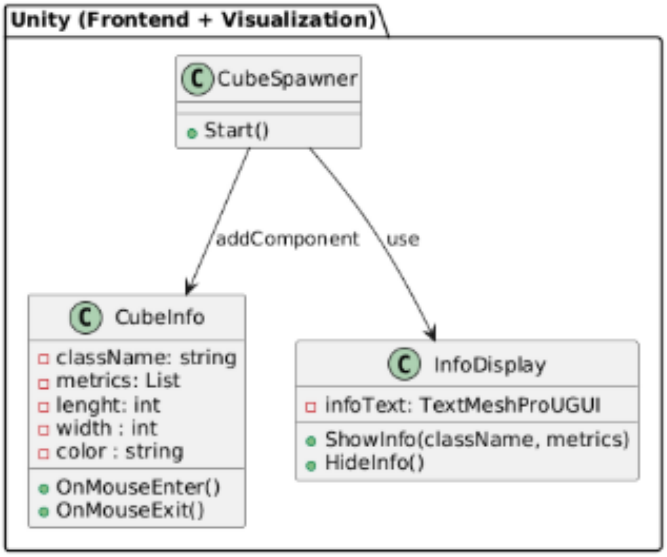

Figure 3.5 Unity (Frontend) UML Diagram

As shown in Figure 3.5, the Unity side is planned to handle the user interface and visualization functions. In terms of visualization, the ‘CubeSpawner‘ class reads data from the JSON file to create new cubes and assigns this data to the ‘CubeInfo‘ class. The ‘CubeInfo‘ class is responsible for displaying the properties of each structure as the user navigates between structures in the scene.

3.4 Sequence Diagram

Sequence diagrams are structural diagrams that show the interactions between components in a software system over time. The interaction sequence for the project is shown in Figure 3.6.

The frontend user interface will be clarified in the later stages of the development process. Whether this interface will be integrated into Unity or be a Java-based web interface will be decided as the project progresses. On the backend side, APIs are being developed under the Java Spring Boot framework. Metric calculations are performed through the *MetricService* class defined on the Java side.

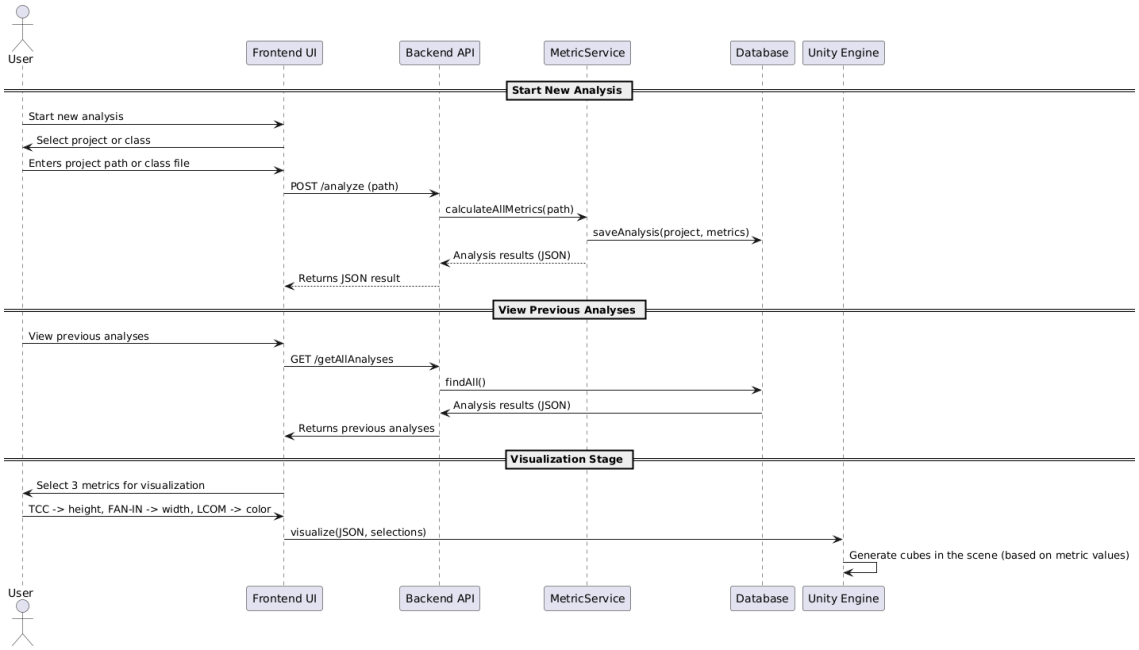

Figure 3.6 Sequence Diagram

3.5 Interface Design

The system interface presented to the user is designed to facilitate user interaction and make the metric analysis process more understandable. The interface is built on a modular structure consisting of four main panels. The functions and capabilities offered to the user by these panels are summarized below:

- **Main Menu Panel**: This panel serves as the start screen for the user and provides access to the system's basic functions. From this screen, the user can start a new metric analysis or view previously performed analysis records.
- **New Analysis Panel**: In this section, the user is asked to select the project folder or specific class file they wish to analyze. Additionally, the user can specify the metrics to be calculated during the analysis (e.g., TCC, WMC, LCOM, etc.). This panel includes a button to initiate the analysis process.
- **Recorded Analyses Panel**: This panel provides a scrollable structure that lists past analysis records. The user can select records from previous analyses to initiate re-visualization or review processes.
- **Visualization Panel**: This panel allows the user to visually compare three different metric values that have been selected in advance. Through this panel, the user can determine which metrics correspond to dimensions such as height, width, or color on the stage. For example, TCC can be assigned to the height

axis and FAN-IN to the width axis. This structure enables intuitive analysis of complex metric data.

3.6 Database Design

The database design of the project is shown in Figure 3.7. This structure was developed to store the metric results of the projects analyzed in the system and to provide easy access to these results later.

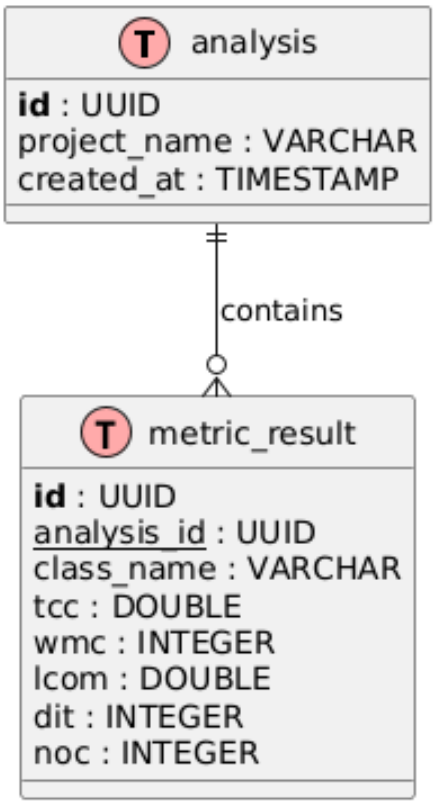

Figure 3.7 Database Diagram

As shown in Figure 3.7, there are two main tables in the system: **analysis** and **metric_result**. When a user performs an analysis on a project, metric calculations are performed for each class in the project, and the metric results for each class are stored as separate records in the **metric_result** table. This results in a one-to-many relationship between the **analysis** and **metric_result** tables. When a user wants to view past analyses, this database structure allows for efficient querying of analysis records.

The system design of the project has been planned according to this structure and is open to further development and optimization in the future. The design is expected to be further expanded before the final presentation.

4
APPLICATION RESULTS AND TESTS

4.1 Application Introduction

The system developed in this study aims to perform metric analysis on compiled Java class files of object-oriented software projects and visualize the analysis results in a three-dimensional environment [10].

The user can initiate the analysis by selecting .class files through the user-friendly interface provided by the system. At the end of the analysis process, the metrics calculated per class (e.g., WMC, DIT, LCOM, TCC) are stored in the system's database and transferred to the visualization module in JSON format.

In the application, the user can view past analysis results and re-visualize them. Additionally, the user can select three metrics obtained from the analysis results and map them to graphical properties such as dimension (height, width, depth) and color. This flexibility enables metrics to be interpreted in a more intuitive and comprehensive manner [11].

4.2 Application Interface

In order to enhance the user experience of the project, the interface has been developed with React to have a modern and intuitive structure. When the application is launched, Unity automatically runs in window mode and is ready for the visualization module.

The user starts the process by selecting one of the options "Start New Analysis" or "View Past Records" on the main screen. This structure enables the application to go beyond merely performing analysis functions, offering the ability to view and compare past records like a version control tool. This feature is particularly valuable for sustainability and code evolution. The main screen is presented in Figure 4.1.

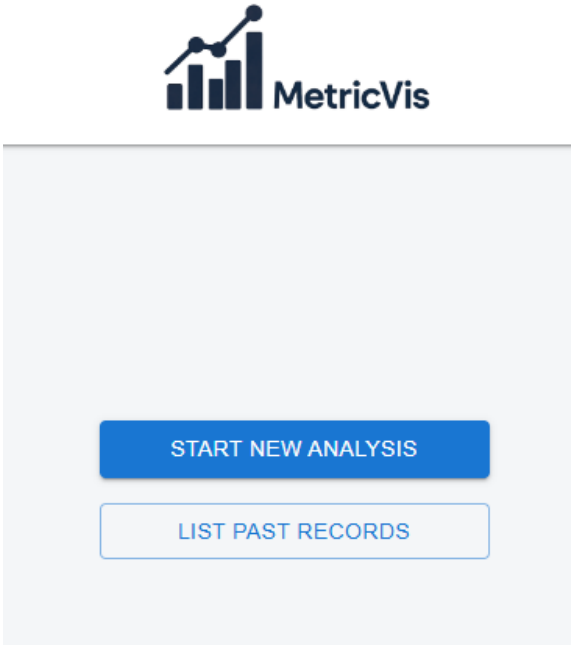

Figure 4.1 Home Page

4.2.1 Start New Analysis

When the user creates a new class or project, they can start the analysis process through this module. The interface allows the user to select the .class file or the entire project folder they wish to analyze. The selected file path is sent to the Spring-based backend service, and the metric calculation process begins. This workflow is visualized in Figure 4.2.

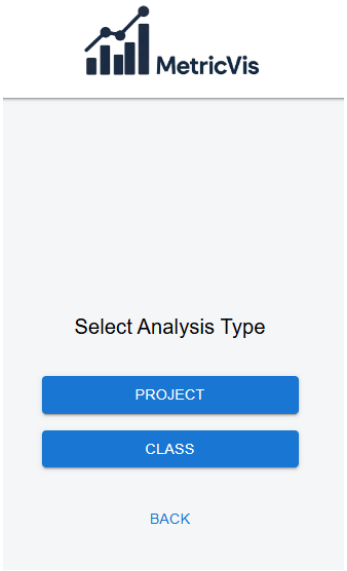

Figure 4.2 Create New Analysis

4.2.2 View Past Records

The user uses this module when they want to access past analysis results. Previously performed analyses are listed by project or class. This feature enables comparative analysis of software versions and provides great convenience for developers who want to track code evolution.

In terms of performance, each analysis record is not returned with metric results; instead, it has been optimized to only include the necessary basic information. This ensures that system performance is maintained even in databases containing a large number of records. The relevant interface is shown in Figure 4.3.

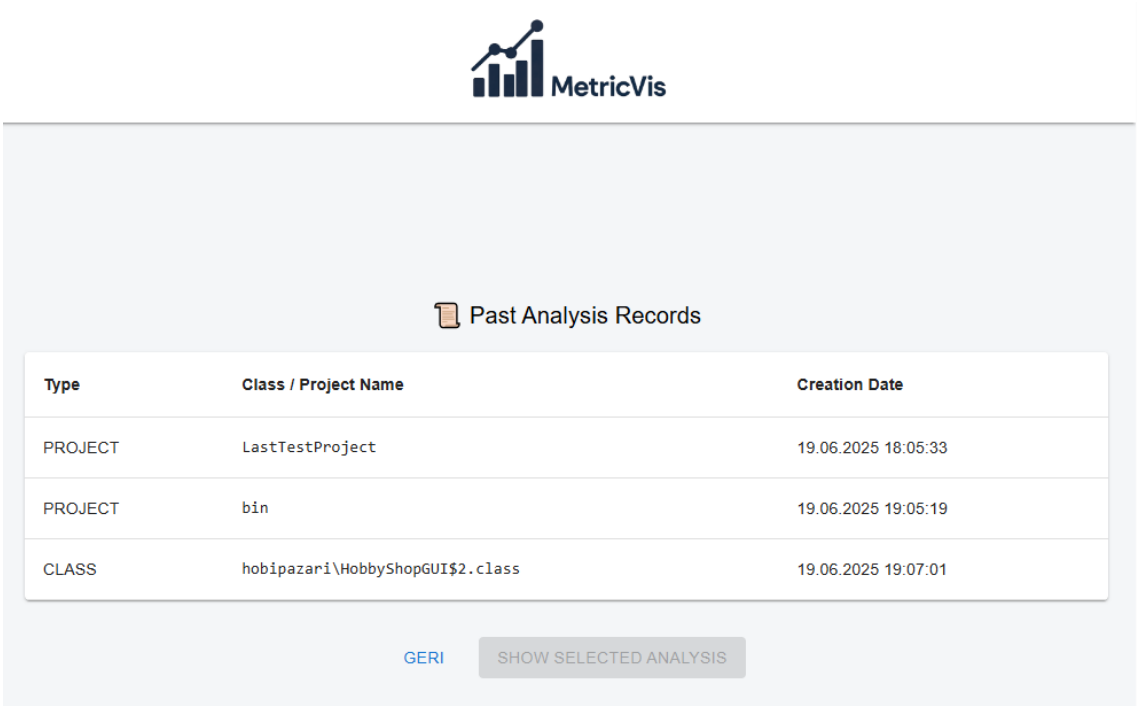

Figure 4.3 Past Records

4.2.3 Visualization

After the analysis process is complete, the user has the option to visualize the selected metrics in a 3D environment. The visualization process is based on mapping buildings created to represent software components according to three different properties: height, width, and color.

The user initiates an interactive analysis in the Unity-based scene by specifying which visual feature to use for each of the three metrics. The metric mapping interface is shown in Figure 4.4.

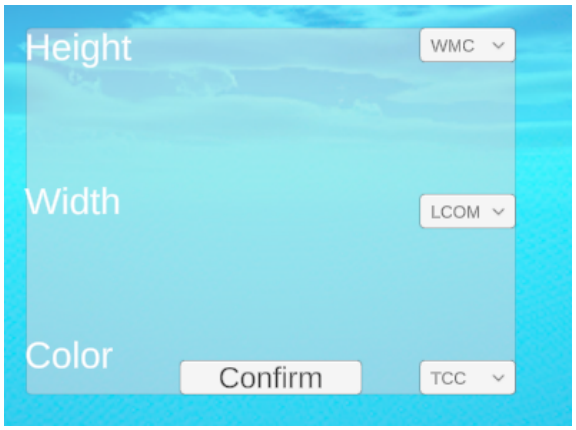

Figure 4.4 Metric Mapping

A 3D city structure is created based on the selected metrics. When the user hovers over the structures, they can directly see the metric information belonging to the relevant class. This makes it easier to analyze the project intuitively and identify problematic classes. A visualization example is shown in Figure 4.5.

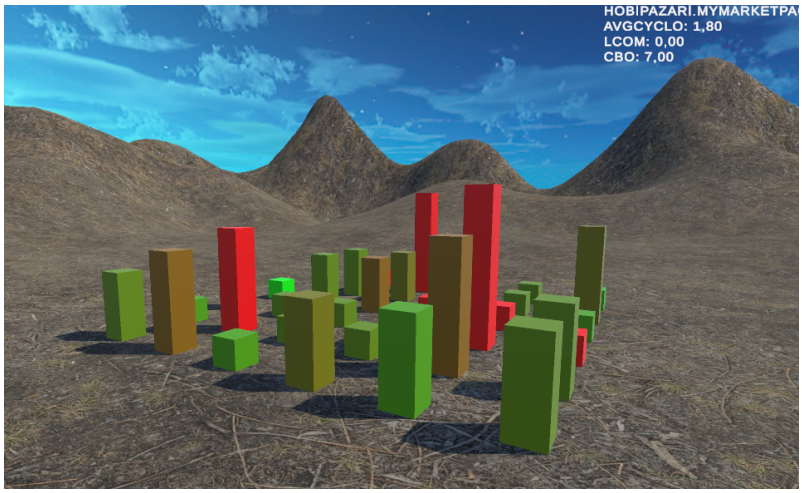

Figure 4.5 Visualization

4.3 Test Results

During the testing process of the application, analyses were performed on Java projects of different sizes, and the consistency, accuracy, and timing performance of the system were evaluated. Both small-scale sample projects (containing 10-20 classes) and medium-scale projects (containing 100+ classes) were used in the tests.

The test scenarios include the following steps:

- Uploading .class files from different projects to the system,

- Successful completion of metric analyses,
- Serialization of metric data in the correct JSON format,
- Access to past analysis results and replayability check.

All test scenarios have been observed to work as expected. In particular, analyses performed using the Apache BCEL library in the metric calculation module have yielded successful results in terms of accuracy and stability. Analysis times vary depending on the size of the project, ranging from an average of 1–2 seconds for small projects to 5–10 seconds for large projects. These durations were found to be appropriate for the application purpose and not to keep users waiting.

4.4 Verification

The accuracy of the system was tested by comparing it with both the theoretically expected metric values and the analysis results obtained from sample classes. In this context, a specially prepared class named MetricTest was analyzed, and the following metric results were obtained:

```java
public class MetricTest {
 private int a;
 private int b;
 private int c;

 public MetricTest(int a, int b) {
     this.a = a;
     this.b = b;
     this.c = calculateC();
 }

 private int calculateC() {
     return a * b;
 }

 public int sumAll() {
     return a + b + c;
 }

 public int multiplyAB() {
     return a * b;
 }

 public boolean isPositive() {
     return sumAll() > 0;
 }

 public static double average(int... values) {
     int total = 0;
     for (int v : values) {
         total += v;
     }
     return (double) total / values.length;
 }
}
```

Figure 4.6 Metric Test Class

Metric	Value
TCC	0.5
WMC	8
LCOM	0
DIT	0
CBO	2
MAXCYCLO	2
AVGCYCLO	1.33

Table 4.1 Analysis output belonging to the MetricTest class

TCC (Tight Class Cohesion): The TCC metric measures whether there is access to common areas between methods. There are three fields named a, b, and c in the MetricTest class. The calculateC, sumAll, multiplyAB, and isPositive methods use these fields directly or indirectly. The average method is static, so it does not access class variables. Since approximately half of the method pairs access common fields, the TCC value is calculated as 0.5.

WMC (Weighted Methods per Class): WMC is the sum of the complexity values calculated based on the control structures (if, for, switch, etc.) of the methods. The isPositive method contains a conditional statement, and the average method contains a for loop, which add a total of 8 units of decision complexity to the class.

LCOM (Lack of Cohesion in Methods): LCOM measures cohesion based on how methods use fields together. Since most methods in the MetricTest class use common variables, the LCOM value is 0. This indicates that the class has high internal consistency.

DIT (Depth of Inheritance Tree): Since the MetricTest class does not inherit from any user-defined superclass, the DIT value is 0.

CBO (Coupling Between Object Classes): The CBO metric expresses the dependency of a class on other classes. Although there are no significant external dependencies in the MetricTest class other than the average method, the CBO value is calculated as 2 depending on the language structures used and possible system libraries.

CYCLO (Cyclomatic Complexity): It measures the number of logical paths, i.e., branches (if, for, while, switch-case, etc.), within a method to indicate the testability and complexity of the code. Even the simplest method (one with no branches) has a complexity value of 1. Each decision point (branch) increases this value by 1. The highest value in this list is 2. This indicates that the most complex method in the class

has 2 logical paths. Therefore, MAXCYCLO is calculated as 2. The total complexity of all methods is 8. Since there are 6 methods in the class, the average complexity is 8 / 6 = 1.33. Therefore, AVGCYCLO is calculated as 1.33.

As a result of this evaluation, it was observed that the metric values analyzed by the system largely correspond to the actual structure of the class. This indicates that the metric calculation module works correctly and reliably.

5
CONCLUSION AND DISCUSSION

This study aims to analyze software metrics used to measure the structural quality of object-oriented software systems and visualize these metrics in a three-dimensional environment. The project goes beyond being merely an analysis tool that calculates metrics and aims to provide users with an interactive environment where they can intuitively evaluate software architecture.

The implemented system consists of three main components: the analysis module (backend), the user interface (frontend), and the 3D visualization engine (Unity). The system analyzes the .class files received from the user using the Apache BCEL library and produces metric results. These metrics include classic object-oriented metrics proposed by Chidamber and Kemerer, such as WMC, DIT, LCOM, TCC, and CBO. The calculations of these metrics were performed using customized algorithms and transitive analysis within the system; in particular, the relationships between methods were examined in depth in the cohesion and coupling calculations.

The tests conducted revealed that the system delivers accurate and reliable results in projects of various sizes. When the manually calculated metric values were compared with the results automatically generated by the system, a high degree of consistency was observed. This demonstrates both the accuracy of the BCEL-based analysis algorithms and the robustness of the data processing logic. In addition, features such as the ability to export metrics in JSON format and reload previous analyses from the database increase the flexibility and sustainability of the system.

In the operations performed through the user interface, importance has been given to the ease of use of the system and user interaction has been taken as a basis. Users can select the projects they want to analyze, specify the metrics they want to calculate, and map these metrics to visualization parameters (height, width, color). This structure allows users to perform metric-based analyses on their own projects and intuitively evaluate complex class relationships.

In the Unity layer, which is the final layer of the system architecture, the goal is to represent the obtained metrics within a software city metaphor. Each class is modeled as a building, and the metric values are reflected in the dimensional properties of the building. For example, the WMC value is associated with the height of the building, the CBO value with the width, and the TCC value with the color intensity. This structure helps to quickly and intuitively understand code complexity and interclass relationships. Thanks to the interactive nature of the visualization engine, users can navigate between classes, access metric details by clicking on a specific class, and easily identify potentially problematic classes.

During the testing phase of the application, both metric accuracy and performance were evaluated. Analysis times ranged from 1–2 seconds for small projects and 5–10 seconds for medium and large projects. These values demonstrate that the system is sufficiently fast for real-time use.

The results obtained show that visualizing software metrics in a 3D environment provides much more than just presenting static analysis results numerically. Creating visual awareness among complex structures facilitates software maintenance processes and provides a holistic view of software architecture [12]. Additionally, this project has the potential to be a useful tool for developers, educators, and students working in the field of software quality.

In the future, the system can be developed to support more metric types, increase visualization options, and analyze projects from different programming languages. Additionally, class-based recommendation systems (e.g., refactoring suggestions) can be integrated to further contribute to software engineering practices.

In conclusion, this study offers a comprehensive platform that brings together the processes of in-depth analysis, systematic storage, and intuitive visualization of software metrics. Suitable for both academic and industrial use, this system can be considered a powerful tool for improving software quality and reducing maintenance costs.

References

[1] M. Lehman, “Programs, life cycles, and laws of software evolution,” *Proceedings of the IEEE*, vol. 68, no. 9, pp. 1060–1076, 1980.

[2] S. R. Chidamber and C. F. Kemerer, “A metrics suite for object oriented design,” *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[3] L. C. Briand, J. W. Daly, and J. K. Wüst, “A unified framework for coupling measurement in object-oriented systems,” *IEEE Transactions on Software Engineering*, vol. 25, no. 1, pp. 91–121, 1999.

[4] M. Lanza and S. Ducasse, “Polymetric views—a lightweight visual approach to reverse engineering,” *IEEE Transactions on Software Engineering*, vol. 29, no. 9, pp. 782–795, 2003.

[5] R. Wettel and M. Lanza, “Codecity: 3d visualization of large-scale software,” in *Proceedings of the 30th International Conference on Software Engineering (ICSE)*, IEEE, 2008, pp. 921–922.

[6] F. Steinbrückner and C. Lewerentz, “Representing development history in software cities,” *Proceedings of the 5th international symposium on Software visualization*, pp. 193–202, 2010.

[7] M. Hitz and B. Montazeri, “Measuring coupling and cohesion in object-oriented systems,” *Proceedings of International Symposium on Applied Corporate Computing*, 1996.

[8] S. Ardigò, C. Nagy, R. Minelli, and M. Lanza, “M3tricity: Visualizing evolving software data cities,” *arXiv preprint arXiv:2202.02589*, 2022.

[9] T. Panas, I. Gorton, and F. Chevalley, “Toward a taxonomy of software visualization,” *2005 ACM Symposium on Software Visualization*, pp. 137–146, 2005.

[10] M. Beck, J. Trümper, and J. Döllner, “Software diagnosis: Bridging the information gap between management and development,” *Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)*, 2011.

[11] N. Chotisarn *et al.*, “A systematic literature review of modern software visualization,” *arXiv preprint arXiv:2003.00643*, 2020.

[12] M. Ó Cinnéide, L. Tratt, M. Harman, and S. Counsell, “Experimental assessment of software metrics using automated refactoring,” *Proceedings of ICSM 2007*, pp. 147–156, 2007.

Curriculum Vitae

FIRST MEMBER

Name-Surname: ERAY GÖKÇE
Birthdate and Place of Birth: 17.07.2002, Kırklareli
E-mail: eray.gokce@std.yildiz.edu.tr
Phone: 0531 960 13 56
Practical Training: Mimsoft ARGE Arşivleme Tek A.Ş.
IBTECH Uluslararası Bilişim ve İletişim Teknolojileri
Yapıkredi Teknoloji

Project System Informations

System and Software: Windows Operating System, Java
Required RAM: 2GB
Required Disk: 256MB