Автономная некоммерческая организация высшего образования «Университет Иннополис»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)
по направлению подготовки
09.03.01 - «Информатика и вычислительная техника»

GRADUATION THESIS (BACHELOR’S GRADUATION THESIS)
Field of Study
09.03.01 – «Computer Science»

Направленность (профиль) образовательной программы
«Информатика и вычислительная техника»
Area of Specialization / Academic Program Title:
«Computer Science»

Тема / Topic	**Визуализация программного кода с применением технологий дополненной реальности / Source code visualization in AR**

Работу выполнил / Thesis is executed by	**Хабибуллин Аскар Расулович / Khabibullin Askar**	подпись / signature
Руководитель выпускной квалификационной работы / Supervisor of Graduation Thesis	**Силлитти Альберто / Sillitti Alberto**	подпись / signature

Иннополис, Innopolis, 2021

Contents

1 Introduction **8**

1.1 Basics of software inspection 8

1.1.1 Static analysis 9

1.1.2 Dynamic analysis 9

1.1.3 Evolution analysis 9

1.2 Visual representation. Human perception. 10

1.3 Augmented reality and metaphors 10

2 Literature Review **12**

2.1 Human perception and AR development 12

2.1.1 Visual perception 12

2.1.2 Augmented reality technologies 14

2.2 Source Code visualization 15

2.2.1 Static analysis 16

2.2.2 Evolution analysis 16

2.2.3 Behaviour analysis 16

2.3 Static Metrics 17

2.4 Conclusion 18

3 Methodology **19**

3.1 Software requirements 20

3.1.1 Overall Description 20

3.1.2 System features for Web Frontend 21

3.1.3 System features for Server 25

3.1.4 System features for Application 27

3.1.5 Non-functional requirements for Web Frontend 29

3.1.6 Non-functional requirements for Server 29

3.2 System solutions 30

3.2.1 User Experience Research: Interview and Statistics . . 30

3.2.2 UX research short results and interpretation 30

3.2.3 Reasons to implement the Upload Frontend 31

3.2.4 Server-side solutions 32

3.2.5 Theoretical model of CodeVisualizer Server 34

3.2.6 Theoretical model of Viroreact Application 34

3.2.7 Set of initially implemented metrics 35

3.2.8 Domain Portals 37

3.3 Conclusion of Methodology 38

4 Implementation **39**

4.1 Website entry point. File upload frontend. 39

4.1.1 General Description of the implementation 41

4.1.2 Technical details of implementation of Upload frontend . 47

4.2 Our approaches to metric visualization 49

4.2.1 Metric calculation 49

4.3 Server application 50

4.3.1 General implementation details 50

4.3.2 Upload endpoint 51
4.4 ViroReact application 53

5 Evaluation and Discussion 59
5.1 Server-side evaluation 59
5.1.1 Single-user scenario 59
5.2 Multi-user scenario 60
5.3 Memory consumption in get project metrics requests 61
5.4 Client-side evaluation 62
5.4.1 Web frontend 62

6 Conclusion and Furhter Work 64
6.1 Further work on server 65
6.1.1 Integration with other utilities 65
6.1.2 Low concurrent performance 65
6.1.3 Real-time updates for User Projects 65
6.2 Further work on Client Applications 65
6.2.1 ViroReact app 66
6.2.2 Web application 66

Bibliography cited 67

List of Figures

3.1 Web Frontend DFD diagram 32

3.2 Initial solution, extracted from the Eclipse Metrics plugin . . . 33

3.3 Particular case of the PMD extracted data, that the server exposes to the clients 36

4.1 Website Entrypoint v0.1.0 40

4.2 Website Entrypoint v0.3.0 41

4.3 Website Entrypoint v0.2.1(minor bug fix, alpha version 2) . . . 42

4.4 Project Uploading procedure 42

4.5 Basic visualization of the LOC metric 44

4.6 Advanced LOC metric with metric filter applied 46

4.7 Example of failed operation 49

4.8 *Here we may observe the normalized LOC metric statistics, multiplied by the value of 5 for each class of the inspected system* . 55

4.9 The following is a code for initial visualization 56

4.10 *An example of the LOC metric, rendered as a part of a PortalScene. In this scenario, user has entered the inspected data domain, through our Domain Portal metaphor* 58

5.1 Example piece of code that we used in out environment measurement 61
5.2 Example piece of code that we used in out environment measurement 62

Abstract

The interest in Augmented reality grows constantly. More and more practical applications of immersive technologies are being researched. This paper is mainly focused on the sphere of source code visualization in Augmented reality. It is describing a software system that the authors have developed, and provides the reader with the detailed description of the solution.

Chapter 1

Introduction

Immersive technologies that alter our surroundings have been well-researched over the last 50 years. Although AR and VR technologies have not yet stepped into their "golden age", there has been a tremendous growth of both commercial and open-source applications. Both technologies that are already altering our perception of computer software by solving many problems in different spheres. Main goal of this project is to study the applicability of immersive technologies in source code visualization visualization, and source code visual representation.

1.1 Basics of software inspection

In software engineering, there are many approaches to solve particular problems. In order to measure how well the implemented solution fits the requirements, the hypothesis can be tested using a set of quantitative measurements. It can not only help to numerically describe the software as it is on the current stage of development, but also estimate the future of the product development, thus increasing the efficiency, and giving essential insights to the

programmers and system designers. Software metrics have been studied (quite a lot).
Before starting with the implementation part, we considered three different approaches to software metrics analysis and visualization:

1.1.1 Static analysis

Structural, or static analysis is performed with the system when we want to discover static dependencies and relation between modules of the software. There are traditional functional decomposition and static analysis approaches, as well as relatively new object-oriented specific metrics, that are giving insight in a field of popular object-oriented paradigm, that has been well-adopted. For this kind of inspection we don't have to run the software.

1.1.2 Dynamic analysis

The behavioral approach is another approach to code studying that is worth introducing. The structural analysis can give certain general insights before the runtime, whereas what happens at the execution phase is out of static analysis' focus and is usually studied in the scope of dynamic software analysis. Depending on the programming language, the execution can be viewed on a higher level of abstraction as functions calling other functions, or objects communicating with other objects.

1.1.3 Evolution analysis

Evolution refers to the development process of the software system and, in particular, emphasizes the fact that program code is changed over time to extend the functionality of the system or simply to remove bugs. The evolution of software was out of our scope, however it is worth mentioning.

1.2 Visual representation. Human perception.

Human brain is far more accustomed to perceiving graphical information if compared to textual. There has been much of research in the sphere and the results speak to themselves - the way we perceive the information is different for textual and visual representation is not quite the same, and the latter can be used in order to effectively increase human studying capabilities. In Chapter 2 we list some recent research in the sphere of Human Cognition and Human Visual Perception, for the reader to get acquainted with a sphere of studies, that Visual Data representation is solely based on.

1.3 Augmented reality and metaphors

Although Augmented reality was first mentioned in late 70s, it's real application has not been clear until the beginning of the decade. Since then, AR and VR field has changed a lot, moving from a few prototypes in research labs to become available to mass market with such companies as Google, Apple including adjacent technologies into their platforms. One of the interesting applications of immersive technologies is data visualization. Even though the source code visualization using the City Metaphor has been studied for a while, there are quite a few solutions that would involve such technologies as Virtual Reality [1] and Augmented Reality [2]. So far the amount of studies in sphere of code visualization with immersive technologies remains fairly limited. This research has been conducted in order to increase the number of visualized characteristics, and extend the existing code analysis tools that by introducing more industrially meaningful metrics, at the same time, decreasing the user-flow complexity and production setup costs. In our research we also try to increasing the number of metaphors to the research, that utilize the Augmented Reality features. We also propose as our goal a general system implementation in JavaScript language.

Chapter 2

Literature Review

2.1 Human perception and AR development

2.1.1 Visual perception

This part of the review is a brief introduction to the large and growing sphere of studies focused on how our brain perceives and processes sensory information. It also will provide the reader with information related to the concepts of textual information encoding, using basic primitives, such as points, lines, areas and volumes, and their properties, which are colors, shapes, positions etc., as well as, more complex presentational approaches, greatly described in an article [3]

There is a large volume of published studies describing the fundamental difference between how our brain perceives, processes, and retrieves visual and textual information. Sperry et. al have suggested that each of the human brain's hemispheres contains a detached processing unit. The fact that each part of the brain responsible for different subtasks, when observing the environment was further studied in [4]. The researchers have conducted a number of experiments demonstrating the peculiarities of the verbal and visual memory. The "dual-coding" theory, proposed by the authors, was verified by PET(positron emission tomography) and fMRI(functional MRI) that compared the results of memory experiments between verbal and visual studying approaches. It was found that learning the textual information along with their images has improved participants' memorizing performances.

There are quite a few visual properties that affect cognitive and perception abilities. Some of them, such as Colors, Shapes are presented to the reader in this part of the Chapter.

Colors

Our eyes see the world in colors, this is the human representation of lightwaves of different lengths. There are some colors that draw our attention more than the others, according to Meier, who have proposed a metaphor color theory in 2005. In a recent research Fatma Cubuk et al. conducted a number of Visual-Spatial Aptitude tests, which have proven that cognitive tests were solved generally better when suggested on white background, and have proven quite the opposite with a grey background, by measuring the brain activity zones during the process. The results have also shown that some of the colors produce the brainwaves that are more likely to be found when the brain is more active. A great research that studies the emotional effect of red color, based on the EEG has been conducted [5] by Kuniecki et. al.

Forms and shapes

Forms and shapes are also properties that can affect the effectiveness of studying the information. In [6], the author suggests that the features of visual objects are preattentive if they are perceived within 200 ms, i.e. the time interval it takes before the eye reacts and moves. Among the properties that are analyzed preattentively, the author mentions shapes, orientation, sizes, numbers of objects etc.

Metaphors

All the above mentioned properties do not guarantee the proper perception, due to the chance of information overload, and structural complexities of data. According to S.Risch “Image schemas are thus theorized to form a crucial link between perception and cognition, with obvious implications for information visualization.” Metaphor usage is an approach to address such issues. Stephan Dhiel defines a visual metaphor as ”analogy which underlies a graphical representation of an abstract entity or concept with the goal of transferring properties from the domain of the graphical representation to that of the abstract entity or concept”. There are many software visualization metaphors, that exist today. The metaphor that we use in this research is City Metaphor. First mention of City Metaphor can be found in Knight and Munro’s work, where the authors exploited Maverik VR toolkit to develop a tool called Software World. After that there were a lot of applications that visualized software characteristics, a complete study of such metaphors was studied by the authors. One of the most interesting studies, that aggregates many solutions in the sphere is [7]

2.1.2 Augmented reality technologies

New technologies often drive the development of the software products. Research into the sphere of Augmented and Virtual realities has a long history.

The first recorded AR or VR reference can be found in L.Frank Baums “The Master Key” book, published in 1901. The author mentions a device that is “spectacles” with which “while you wear them every one you meet will be marked upon the forehead with a letter indicating his or her character”. More than than six decades later, the first prototype of the VR device has been built. The original device was extremely heavy, due to such property was given a name “Sword of Damocles”. In [8], Sutherland and colleagues have developed a first HMD(Head Mounted Display) that started to keep track of the user’s head movement. In late 90’s as AR became a standalone research field, several conferences on AR began. Since then, many technological inventions have been produced and introduced to the wide audience, inducing the developers and researchers to come up with more elegant solutions. For the first time the term AR was used in 1992 by Caudell and Mizell. In these three decades the sensors have become smaller and more affordable, handheld devices productions have started to include AR capabilities in their smartphones. Such libraries as Google AR Core and Apple ARKit have become available for a common audience on the compatible devices.

2.2 Source Code visualization

With much research done in the area, the division of the software visualization into three main directions seems to prevail. According to the Stephan Dhiel, the visualization can be done using three main approaches.

2.2.1 Static analysis

The first approach is to look at the static parts of code, the relations between the components. Such analysis can be done without running the program and requires a set of metrics to be defined. Software metrics have been researched for decades, some of them have been proven to be effective for particular purposes. Static code metrics have traditionally been used to inspect the dependency structure, to achieve code optimization, and consistency in the project implementation.

2.2.2 Evolution analysis

The second approach that has recently become more and more popular to look at is to measure and visualize the evolution of the software. After two decades of continuous growth, many researchers provide a great body of knowledge in this particular area. Most of the implementations of the 3-dimensional City metaphor does not handle Evolution of Software. It is a research gap that this work is not aimed at filling, however, interested reader may find a great article on the topic in [9]

2.2.3 Behaviour analysis

Last, but not the least is the Behaviour visualization, or else called Runtime visualization. The execution can be seen as a sequence of program states, where a program state contains both the current code and the data of the program. This work is mostly focused on building a system for static metric visualization, that would enable one or more languages with multiple metrics.

2.3 Static Metrics

A series of two articles [10] and [11], from Chidamber and Kemerer, authors propose a set of metrics and analyze their applications to evaluating the Object-Oriented languages has seen a great response from colleagues, and received both criticizing opposition and further development. We have started our research and implementation using the Depth of Inheritance Tree (DIT), Coupling between Object Classes(CBO), and Lack of Cohesion in Methods (LCOM), however in the final version of the product we propose a different set of metrics that may be represented easier in the context of Augmented Reality.

Coupling Between Object Classes (CBO): In [11] CBO is described as “a count of the number of other classes to which it is coupled”. From the definition in [11] “two classes are coupled when methods declared in one class use methods or instance variables of the other class”) are considered equal.

Earlier in 70s McCabe in his one of his well-recognized papers “A Complexity Measure,” had proposed a **Cyclomatic Complexity metric** that for long time has been a de-facto standard approach to measuring the contorl-flow complexity of a method. In a recent work of G. Ann Campbell, conducted at SonarSource, author, however claims that original intention of McCabe’s paper “to identify software modules that will be difficult to test or maintain”, did not address the issue of remaining unmaintainability and unsatisfactory intelligibility of methods. With that, the latter author proposes a new measurement of code complexity, that addresses this issue, based on the work of [13] first considering the term of ’Cognitive complexity’, reffering the qualities of maintainability and readability of the software from the perspective of other programmers.

LoC(Lines of Code) is another popular metric has been used in the

3 dimensional source code visualization, as an important visualization characteristic for one of the City Metaphors in one of the first City Metaphor implementations in late 80's in the works of Knight and Munro, who proposed the 'Component City' and 'Software World' metaphors. The latter incremented the stories of Components-buildings by one every 10 Lines of Code. However, our work proposes another metric, which is LoCoM (Lines of codes of Methods), that may be helpful in visualization, thus extending the standard set of utilities.

2.4 Conclusion

Due to increasing affordability of such technology as Augmented Reality for common audience, and lack of cheap and efficient visualization platforms, authors of this paper see a great potential in this field of research.

Chapter 3

Methodology

Technical details

This section describes in detail approaches and solutions that we have found and tries to justify those solutions given the system requirements that our team had. As you will find from the next two chapters, the developed system is quite complicated and consists of many modules that are written in Javascript ecosystem. As the main methodology for this system implementation and design, we have chosen to use the Gane & Sarson method. Which essentially can be used to formally describe the logical model, and involves graphical techniques, that allow every member of the team, get the grasp of how the system works without having any special knowledge. This practise is achieved by top-down implementation, starting from the agents and modules of the system, and going down into the smallest details of the system. The reader may find a great methodology review in an article [14]

3.1 Software requirements

3.1.1 Overall Description

Introduction and conventions

This section presents system constraints and requirements for the Code Visualizer project. It will establish the purpose and features of the system, as well as interfaces of the system and it's modules, defined on the early stages of development process. From now on we will refer to any actor as User, the target system in general as System, Web Frontend - the web application in the System, and as for the Android/IOS application as Application. We will divide the description into three different sections describing the aforementioned components individually that would simplify our product domain for each of the subsystems.

Project's scope and purpose

This system is designed for programmers and researchers to help them visualize software characteristics of a project. In some cases, software metrics are helpful to detect and prevent architectural mistakes from spreading in the system. For some languages such as Java, it is possible to retrieve the similar source code from the archive. For such languages there are also utilities with rich functionality that help user to inspect metrics in a textual format from the command line(CLI). Based on the questionnaire, more than 80% of the respondents are not using static software metrics and find them more intimidating than useful. The proposed software system allow users to upload and further inspect software metrics for the decompiled project in a visually enriched environment of Augmented Reality. This application will provide a pipeline project-to-stand which will provide everyone with capabilities to learn their code in a more efficient way than just by looking at a plain text. In the next subsection, called System Environment accustoms the reader with three main parts of the software system.

System Environment

There is the only main actor of the system, that we will develop, that is the reason we are going to present the reader with different User Stories(User Scenarios) that our actor should be able to fulfil their needs in. This project should not be restricted to any platform and should be cross-platform consistent. The source code should be well-documented and should contain short user guides for a user of Windows/Linux(Ubuntu 16.04 and newer)/Android/iOS. User should be able to run the server both locally, or in a remote server environment.

User Documentation

Short documentation should be provided for all of the system parts.

3.1.2 System features for Web Frontend

Upload Project User Scenario

Each user should be able to upload their project into the system from the Web Frontend. For this scenario the target system should contain an upload feature. This is a feature of critical priority and should work in any runtime.

Stimulus and Response Sequences: User interacts with the User Interface via Choose Project button, which then sends the data to the server counterpart. Such loaded projects should later on be observable from any client, having their unique identifier.

Functional requirements:

1. User can click Choose project button which interacts with the browser File API
2. After the product has been chosen, User should be able to upload the project, by interacting the Upload button. Web Frontend should display the name of the project chosen by User.
3. When the files have been uploaded, Web Frontend should reflect the status that has been received from the server to provide the User with only actual information on the upload action. Application should store the last operation status and close it when the user interacts with the status terminal on Web Frontend.

Inspect Projects User Scenario

Whether the user has uploaded her files in the current section or not, she should be able to inspect all the projects that are now on the server-side and choose one of them. This is a feature of critical priority and should work in any runtime.

Stimulus and Response Sequences: User interacts with a UI element that is designed as a dropdown list. When clicked, the dropdown list should send a GET request to the server counterpart and retrieve the data that server has stored for that User.

Functional requirements:

1. User can click on "Choose project" button, which triggers server data fetch call on Web Frontend.
2. User can inspect every project she had uploaded since the account creation and up to the point the button has been clicked.
3. User may choose the project and retrieve only that projects data.

Choose Metric User Scenario

When the system initializes, it should be provided with a list of possible metrics. User can choose to visualize only one metric at a time and in this scenario description we specify what should be achieved here. This scenario is of standard priority.

Stimulus and Response Sequences: User may observe the UI component that allows to see current metric and switch between metrics in runtime. Application has the current metric chosen saved in it's state.

Functional requirements:

1. User can see the chosen metric on the Web Frontend
2. User can move further in the list of metrics and choose the metric they want.
3. Only the chosen metric should be saved and accessed(visualized) at a time.

Fetch Project Metrics Scenario

User can download the chosen project from the server, after they have gone through Inspect Projects Scenario. This downloaded data is stored on the client.

Stimulus and Response sequences:

When a user clicks on any of Project components that have been shown as a result of inspection function, such project is being requested from the server and stored in the State of the appication.

Functional requirements:

1) User click on the project should trigger fetch of actual data in a Visualizer Module. After user clicks on any other project, the new relevant data on a specific metric, chosen in Choose Metric Scenario form a project that has been saved on the server and parsed to it's own directory. No projects should have identical names, and each project should be parsed individually.

Visualize Project Metrics Scenario

User can see the visual representation of a particular metric as a 3D scene, containing individual measurements and Java class names on the Web Frontend. This feature is essential, because we want to target all the systems, including those, that have limited AR capabilities. In this scenario, we describe the system usability for such users.

Stimulus and Response sequences:

When the user has fetched the project, after fulfilling Fetch Project Metric Scenario, a simple visual representation is given on a canvas component.

Functional requirements:

1. User should see a representation of a class system that had been parsed on the server on an HTML canvas element in Web Frontend.
2. User should be able to interact with this canvas using Mouse Wheel and right mouse button click.
3. User should see actual information near the 3D objects on the screen.

Taking the target systems into consideration, we had to come up with a solution that would have been well-tested on many platforms and would generalize the user-flow for users on different platforms.

One of essential requirements was to find the tools that would be open-source and free for non-commercial usage.

3.1.3 System features for Server

Server should have an internal pipeline for the data parsing. All the Scenarios in this subsection are going to describe the general data pipeline for the server every time user uploads their project. Server should be modular and the parsing modules should be interchangeable with little effort. User should be able to create their own mechanisms of metric generation for the server.

Upload Project User Scenario

User should be able to upload the project, which then gets parsed and stored on the server. This is the first segment of the data pipeline.

Stimulus and Response Sequences: User interacts with the User Interface and accesses the upload endpoint of the server.

Functional requirements:

1. After user has triggered the upload point. User must transfer the data over HTTP protocol and server should store it locally.

2. After storing the data, server should create a folder and a subfolder where it will store the data parsed by our utilities, or open-source libraries. Multiple users must be able to upload files, that is why this folders should have a unique identifier.

3. From a predefined set of metrics, server parses all the data and saves this metric parsing results into a subfolder of the project of the user folder.

Inspect Project Scenario

User should be able to individually request project statistics for each stored project. This scenario is the second in the data pipeline.

Stimulus and Response Sequences: User interacts with the User Interface and loads individual data for each project and each metric. This should be done in a unified fashionm having an internal data schema for all the metrics. In the future, is the user wants to switch parsers or languages, they should only change parser to obtain schema-like results.

Functional requirements:

1. Each project is saved in it's individual subfolder of the user folder on the server.

2. Each of the predefined metrics is saved in an individual file of the project subfolder.

3. When accessing the Server REST API user can retreive parsed data in a unique fashion that is enough for the user to visualize data for a particular metric, without having to parse any metric on the client side. For that purpose data interface should be designed and should have the same schema across the platforms.

3.1.4 System features for Application

Login Scenario

Having the token that user receives on the frontend as a unique identifier, user should be able to login to any of their project in an individual way.

Stimulus and Response Sequences: On the Application, each time user loads the App, their token and their project token should be requested. Having correct username and project token inserted, person can get their project statistics.

Functional requirements:

1. There should be modal or a pop-up window implemented.
2. This popup can be triggered at any time when user wants to switch project. This universal component should be working on any step of the application

Choose project scenario

When user has logged in, their data is fetched. Data should contain all the projects that user has submitted up to this point. The same way server should respond to the client of Web Frontend.

Stimulus and Response Sequences: In the UI there should be an option to choose from the projects that user has uploaded. When clicking on that project, the data should be fetched in real time, and visualized.

Functional requirements:

1. Modal window should take a Project List components as an argument and visualize it. The Project list fetches all the components for this user, and presents them as a clickable list.
2. Every time the user clicks on the project, this project should be fetched from the server.

Choose metric Scenario

User should be able to access project metrics in an individual way for each metric.

Stimulus and Response Sequences: In the UI there should be an option to choose from the metrics of each project.

Functional requirements:

1) UI should provide users with capabilities to change metrics and individually visualize them.

Visualize metric

User should be able see a simple graphical representation of the project.

Stimulus and Response Sequences: In the UI there should be an option to choose from the metrics of each project.

Functional requirements:

1) UI should provide users with capabilities to change metrics and individually visualize them.

3.1.5 Non-functional requirements for Web Frontend

Software quality attributes for Web Frontend

As a modern application, Web front-end should be fully compatible with the "major league" of screen sizes, and should be properly designed for all the modes of operation - portrait and lanscape mode. All the texts should be provided in their entirety, and basic usage of the system should not require any additional experience or knowledge. Whole project should be an open-source solution, that is well-documented and easy to start for a beginner. All the Code should contain Javascript of ES6 standard.

3.1.6 Non-functional requirements for Server

Software quality attributes for Server

Server-side is proposed as a standalone system, that would define it's own framework for working with the data. Well-documented solution should be both robust and clear for understanding of the future user.

3.2 System solutions

3.2.1 User Experience Research: Interview and Statistics

To implement a solution that would satisfy the need, we had to design a questionnaire and collected user data. The set of question is shown below:

1. How necessary it is to use metrics when designing a software?
2. Which metrics do you use on a regular basis?
3. Do you or your team use software metrics when designing the software?
4. In your opinion, can a high-quality solution for analyzing and demonstrating metrics help in training the company's staff?
5. Are you ready to consider using metrics analysis software on your smartphone?

3.2.2 UX research short results and interpretation

- 85% of the respondents have found it inconvenient to install any extra software on their devices in order to inspect their metrics.
- More than half of the people who took part in the UX research have found it redundant to inspect static metrics to assure software quality.
- From the perspective of all software developers with 5+ years of software experience, which is 30% of the respondents, stated that, even static metrics could affect the software quality, and should be used in order to

better understand software. 80% of such developers test their program products every day.

- More than 50% of users are of the view that software metrics could be used in order to increase training efficiency when teaching the staff of the company

In order to implement a user-friendly solution, we have decided to use a set of tools that would minimize the user interaction at the time of the code preparation. We have also decided to set it as a requirement for the user to be able to use cross-platform solution, that would also minimize number of utilities to install. In order to receive the Augmented Reality experience, there has to be interaction. In our system the user has to upload the project files. For this purpose the React for Web ecosystem was chosen.

The original idea to use metrics extracting modules and extensions from such popular IDE's as Eclipse and IntelliJ IDE have not seem sufficient in terms of pre-conditions that would require user to install not only the system, but also find the essential module for metric extraction, then save the output to the client application. This user-flow was reconsidered and other approaches were used.

3.2.3 Reasons to implement the Upload Frontend

Using the traditional UX approach to gather the user data. We have found the general properties for the target system, that are described earlier in the chapter. On the Fig 3.1 the reader may observe the complete general DFD Diagram that was designed for the system to be implemented.

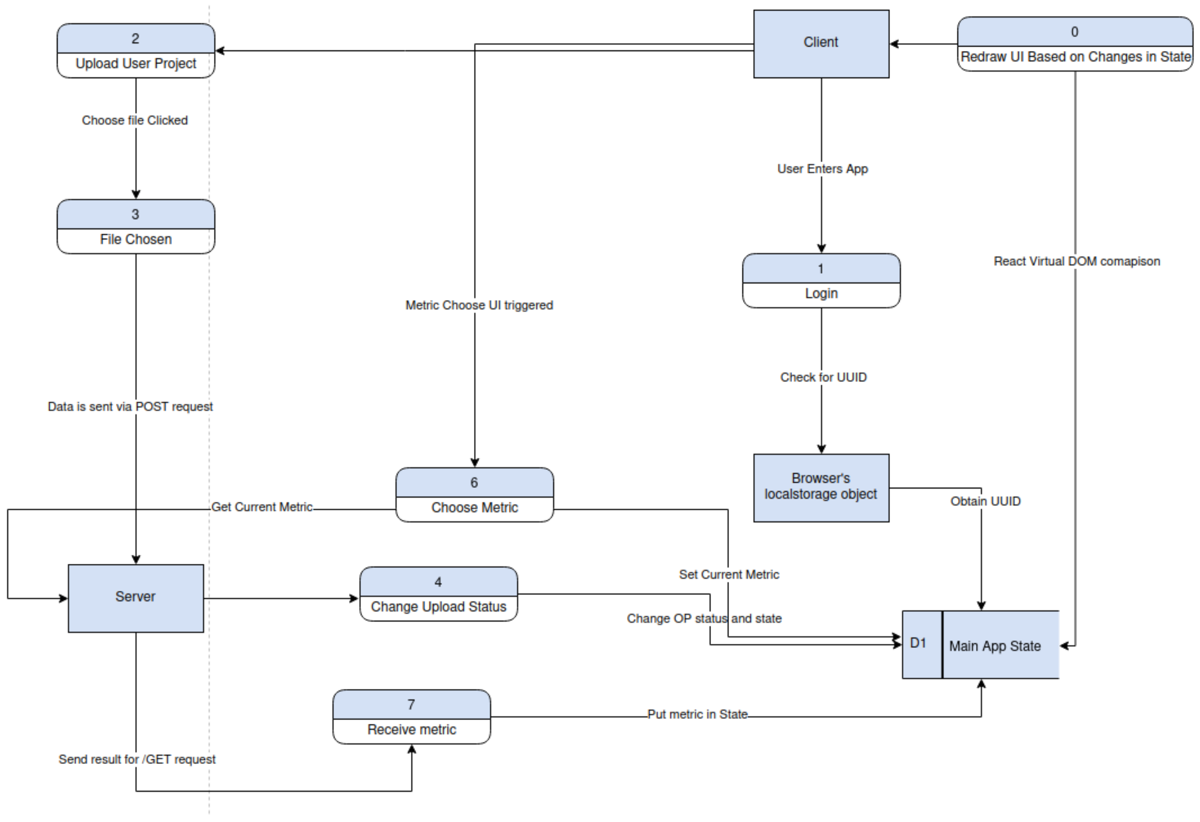
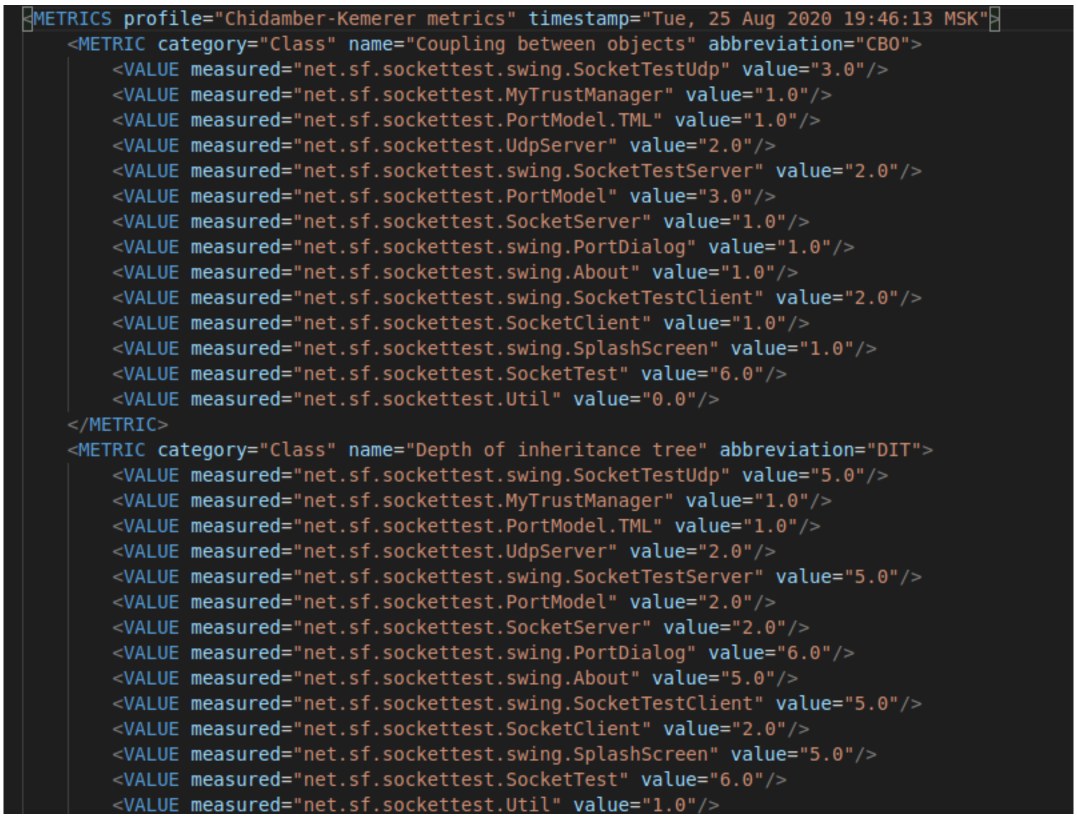


Figure 3.1: Web Frontend DFD diagram

3.2.4 Server-side solutions

Based on the literature review, and taking the summary analysis of the [15]. We have tried different solutions when designing the system:

- **Jarchitect** for Java language constructions provides user with many metrics, but it is not open-source, which was essential for our project.
- **CodeAnalyzer** is one of the greatest tools, however it did not allow to generate output report files, and didn't include some of the essential OOP metrics, that we wanted to include in the project.
- **LocMetrics** is a freeware, that can be used to parse LOC metric, but it is limited only to that metric.



```xml
<METRICS profile="Chidamber-Kemerer metrics" timestamp="Tue, 25 Aug 2020 19:46:13 MSK">
    <METRIC category="Class" name="Coupling between objects" abbreviation="CBO">
        <VALUE measured="net.sf.sockettest.swing.SocketTestUdp" value="3.0"/>
        <VALUE measured="net.sf.sockettest.MyTrustManager" value="1.0"/>
        <VALUE measured="net.sf.sockettest.PortModel.TML" value="1.0"/>
        <VALUE measured="net.sf.sockettest.UdpServer" value="2.0"/>
        <VALUE measured="net.sf.sockettest.swing.SocketTestServer" value="2.0"/>
        <VALUE measured="net.sf.sockettest.PortModel" value="3.0"/>
        <VALUE measured="net.sf.sockettest.SocketServer" value="1.0"/>
        <VALUE measured="net.sf.sockettest.swing.PortDialog" value="1.0"/>
        <VALUE measured="net.sf.sockettest.swing.About" value="1.0"/>
        <VALUE measured="net.sf.sockettest.swing.SocketTestClient" value="2.0"/>
        <VALUE measured="net.sf.sockettest.SocketClient" value="1.0"/>
        <VALUE measured="net.sf.sockettest.swing.SplashScreen" value="1.0"/>
        <VALUE measured="net.sf.sockettest.SocketTest" value="6.0"/>
        <VALUE measured="net.sf.sockettest.Util" value="0.0"/>
    </METRIC>
    <METRIC category="Class" name="Depth of inheritance tree" abbreviation="DIT">
        <VALUE measured="net.sf.sockettest.swing.SocketTestUdp" value="5.0"/>
        <VALUE measured="net.sf.sockettest.MyTrustManager" value="1.0"/>
        <VALUE measured="net.sf.sockettest.PortModel.TML" value="1.0"/>
        <VALUE measured="net.sf.sockettest.UdpServer" value="2.0"/>
        <VALUE measured="net.sf.sockettest.swing.SocketTestServer" value="5.0"/>
        <VALUE measured="net.sf.sockettest.PortModel" value="2.0"/>
        <VALUE measured="net.sf.sockettest.SocketServer" value="2.0"/>
        <VALUE measured="net.sf.sockettest.swing.PortDialog" value="6.0"/>
        <VALUE measured="net.sf.sockettest.swing.About" value="5.0"/>
        <VALUE measured="net.sf.sockettest.swing.SocketTestClient" value="5.0"/>
        <VALUE measured="net.sf.sockettest.SocketClient" value="2.0"/>
        <VALUE measured="net.sf.sockettest.swing.SplashScreen" value="5.0"/>
        <VALUE measured="net.sf.sockettest.SocketTest" value="6.0"/>
        <VALUE measured="net.sf.sockettest.Util" value="1.0"/>
```

Figure 3.2: Initial solution, extracted from the Eclipse Metrics plugin

- **Eclipse Metrics** is one of the tools, that we have initialy implemented our parsers for. It allows **.xml** export, which can be found at picture Although this solution was enough to fulfill some of the requirements, we have decided to move further looking for ways to implement batch processing of user projects, that would require no user interaction. In the github repository that contains our project's code, the reader may find the parser for this utility.

3.2.5 Theoretical model of CodeVisualizer Server

We have chosen NodeJS and ExpressJS to receive, process, and access all the user data. The asynchronous nature of the framework is opposed to a traditional multi-threaded approach to execute multiple concurrent operations. This two approaches are based on two opposite technological solutions that address the issue of handling multiple incoming operations at a time.[16] Rather than acquiring additional threads of execution for concurrent operations, NodeJS utilizes a concept of events. First problems with the traditional multi-threaded approach start to appear with many concurrent operations executed. This problem is known as C10K problem, and was first proclaimed by Kegel in 1999. You may find his works in the following source. [17]

Node is an asynchronous JavaScript runtime, that is operating based on the events. General architecture of NodeJS consists of two essential parts in its core. *libuv* library, that allows NodeJS to interact with an operating system. The second core part of NodeJS is an open-source engine *Google V8.* The server part of the project handles the incoming data, uploaded by the user, and parses the project using any module, that allows batch execution. Due to it's asynchronous nature, the main thread of execution is not blocked by batch project decompilation. This allows multiple users to upload their project data at the same time. The detailed description of each submodule of Server program module is described in Chapter 4. of this document.

3.2.6 Theoretical model of Viroreact Application

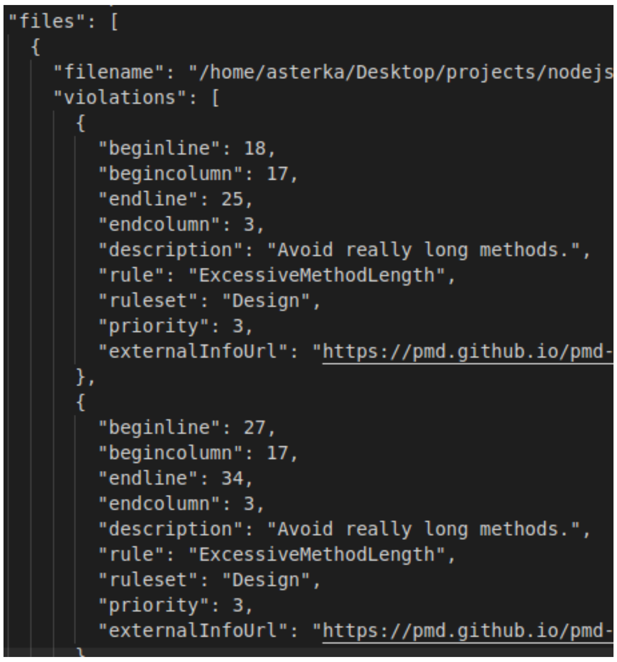
When developing web applications, React 17.0.1 is mainly used in projects with both complex and simple User interface. React is a rich library, that does not enforce any architectural patterns, so the user can choose how to build the application in the ecosystem of the library based on what properties they wish to achieve. It can be rendered on both client and server sides, has many patterns that were developed by the community, that is, according to the github repository is in top-3 in the sphere of Web development. Choosing the framework was an essential part of this project, and we have chosen React for the following reasons:

- React has a great community support. More libraries, tools and modules.
- Allows us to choose our own architectural patterns.
- Has a cross-platform capabilities within the ecosystem.

When designing the Application we have used the same top-down approach to define the data flow, because it best describes the internal component hierarchy, and the process of state management and props passing. More details on the implementation of the ViroReact application are provided in the Implementation[4] chapter of this work.

3.2.7 Set of initially implemented metrics

The outcome utility of this research is aimed at helping developers visualize their code in an AR environment as well in general 3D environment. Hence, we decided to pick a set of metrics, that would be applicable in an everyday development. We provide the set of basic **xml** metrics extraction rules for the PMD utility. Based on the extracted metric, there might be either one or many **violations** in the output JSON object, that authors provide in Fig.3.3. Both frontend applications work with the same parsed results for the project.



```
"files": [
  {
    "filename": "/home/asterka/Desktop/projects/nodejs
    "violations": [
      {
        "beginline": 18,
        "begincolumn": 17,
        "endline": 25,
        "endcolumn": 3,
        "description": "Avoid really long methods.",
        "rule": "ExcessiveMethodLength",
        "ruleset": "Design",
        "priority": 3,
        "externalInfoUrl": "https://pmd.github.io/pmd-
      },
      {
        "beginline": 27,
        "begincolumn": 17,
        "endline": 34,
        "endcolumn": 3,
        "description": "Avoid really long methods.",
        "rule": "ExcessiveMethodLength",
        "ruleset": "Design",
        "priority": 3,
        "externalInfoUrl": "https://pmd.github.io/pmd-
```

Figure 3.3: Particular case of the PMD extracted data, that the server exposes to the clients

Cyclomatic complexity

When writing a piece of software it is considered a great practice to spread the conditional logic. Concentrating such language operators as **if**, **while**, **for**, and **case** inside of a single method creates complexities, that can decrease the performance of the team while code reviewing. Together with the Cyclomatic complexity we provide an automated extraction scenario for Cognitive Complexity, because it is not always justify, to break the control flow by referring to other entities.

Lines of Code in Methods

Comparing to a traditional LoC metric, that has long been considered standard to describe code efficiency of the team, this metric is no longer considered effective. Instead of using the LoC metric, authors inspected using the LoCoM - Lines of Code of Methods, and visualization with dynamic UI interaction with such metric. By using the rules, created for the PMD utility, we create a data structure, that contains all the methods of the class and their length. We calculate length from the violation description, having the *beginline* and *endline*. From this metric user will get the information about long methods of the classes.

Excessive Class Length

Excessive class lengths are usually indications that the class may contain too much logic. When reusing extra logic, and setting it apart, we could increase readability, although this should be done, considering Cognitive complexitiy metric too. We visualize classes of exceeding length based on the threshold parameter, set in the UI.

Dependency extraction for Java classes

Due to the fact that we deal with user archives in JAR format, we have opted for the standard **jdeps** utility, that would help us extract class level and package level dependencies for the target project. For such output, we also create the output DOT file parser with $O(n^2)$ complexity. Analyzing the dependency tree helps us detect the Coupling between classes of the project. We try to visualize the dependent classes as in the traditional City Metaphor, by using the 'electric lines' between 'buildings' of the scene.

3.2.8 Domain Portals

When we consider an aggregate of data, we are used to logically characterizing it's 'data domain' - it characterizes the set of all possible attribute values in that aggregate. Traditional UI interactions require an interaction, that involve input device clicking. Alongside with the traditional metaphor, this research proposes the Domain Portal metaphor, that may be effectively used for data domain navigation. In our implementation, we use the technical solutions that track user's physical movement and visualize data, based on this interaction.

3.3 Conclusion of Methodology

In this section authors attempted to give the most detailed explanation for the theory, that lays under the [4] section solutions.

Chapter 4

Implementation

In this chapter, a detailed description of the process of system implementation is provided. It has been divided into sections, which aim at describing each part of the piece of software in particular, therefore, by going through this part of the paper, one can obtain knowledge about program product in general, as well as the details of each particular implementation. Rather than describing the system chronologically, the authors order the products by their first appearance in the user-flow or, rather, User Stories, which have been described in detail in the previous section.

4.1 Website entry point. File upload frontend.

When designing the system, we wanted to achieve a better user experience. In terms of implementation-related decision making, that meant delivering the final product which would provide the user with the shortest user-flow path possible alongside with the simplest design. In order to create a flexible and self-sufficient design, authors have created all the layouts in Figma, a tool that is a standard tool in sphere of product design, that describes every system component, and provides commercial level design of each component of the system, alongside with each screen design.

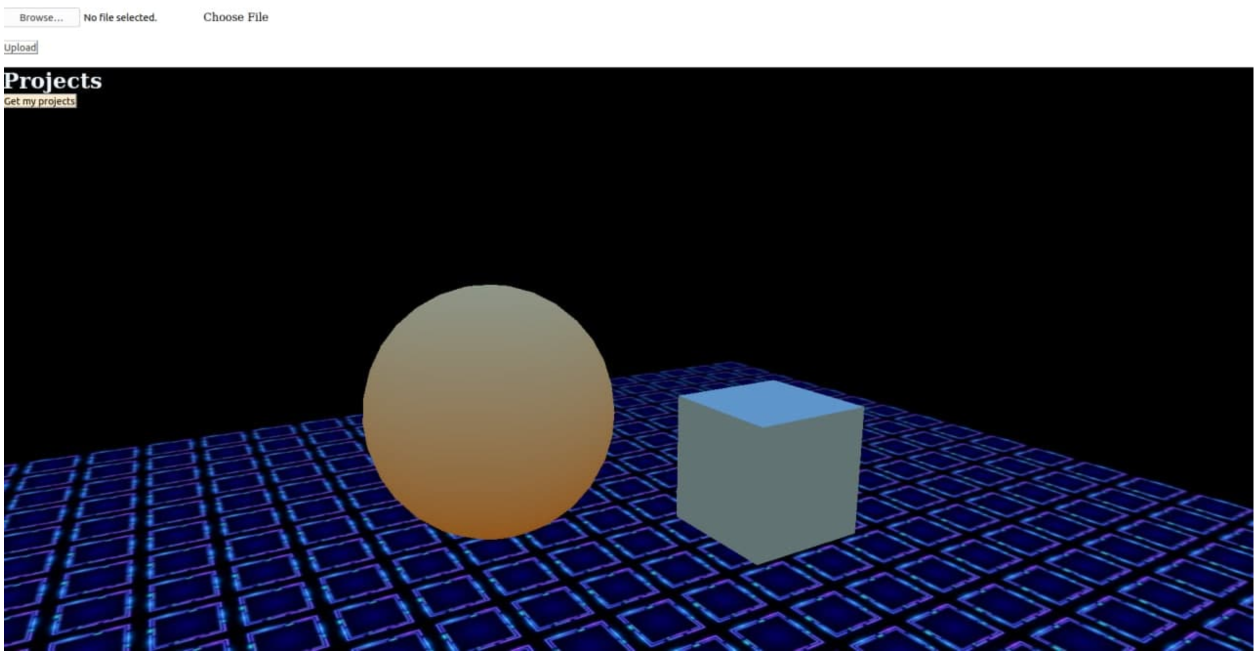


Figure 4.1: Website Entrypoint v0.1.0

The Website Entry point, that you may observe in the figure 4.1 is an early alpha version v0.1.0, which was designed in order to reduce the User Story of "Upload user file" to two main actions, that involve user decisions. Continuous incremental development has resulted in to more alpha versions of the product. The latest details of the implementation can be found in the repository of this project.

Requirements fulfillment

With version 0.3.0 we have introduced two more functional React components to the System, which covered Upload User Project, Inspect User Project and Choose User Metric scenarios completely.

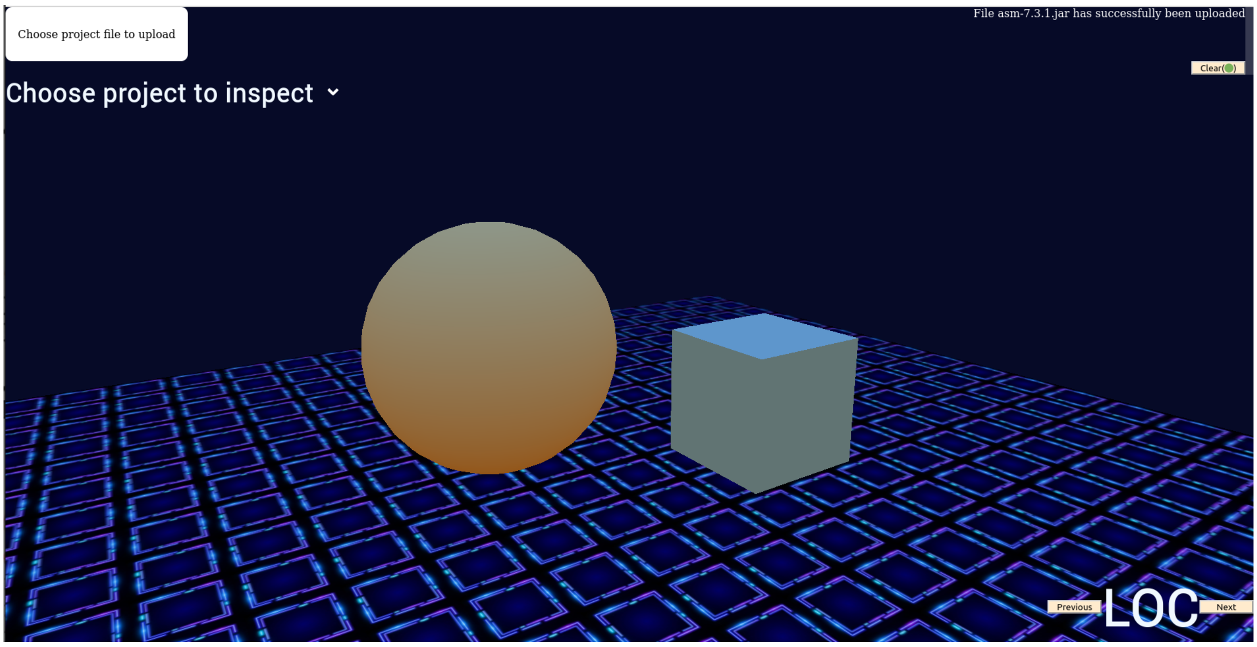

Choose project file to upload
File asm-7.3.1.jar has successfully been uploaded
Clear
Choose project to inspect
Previous
LOC
Next

Figure 4.2: Website Entrypoint v0.3.0

4.1.1 General Description of the implementation

Login procedure

As you will find from the section 4.2 of this chapter, each user has a unique ID, associated with all the data stored on the server for that particular persona. The Login Procedure that the user goes through at the beginning of her user-flow is obtained by checking on the browser's localStorage at the time when the user enters the front-end. In case there is no data, the browser will generate a new token for such user, and save it for the next login. That is how the user does not dissapear into thin air, as long as she does not clear her browser's localStorage. Authorization was not so far implemented due to it's redundancy in local software exploration.

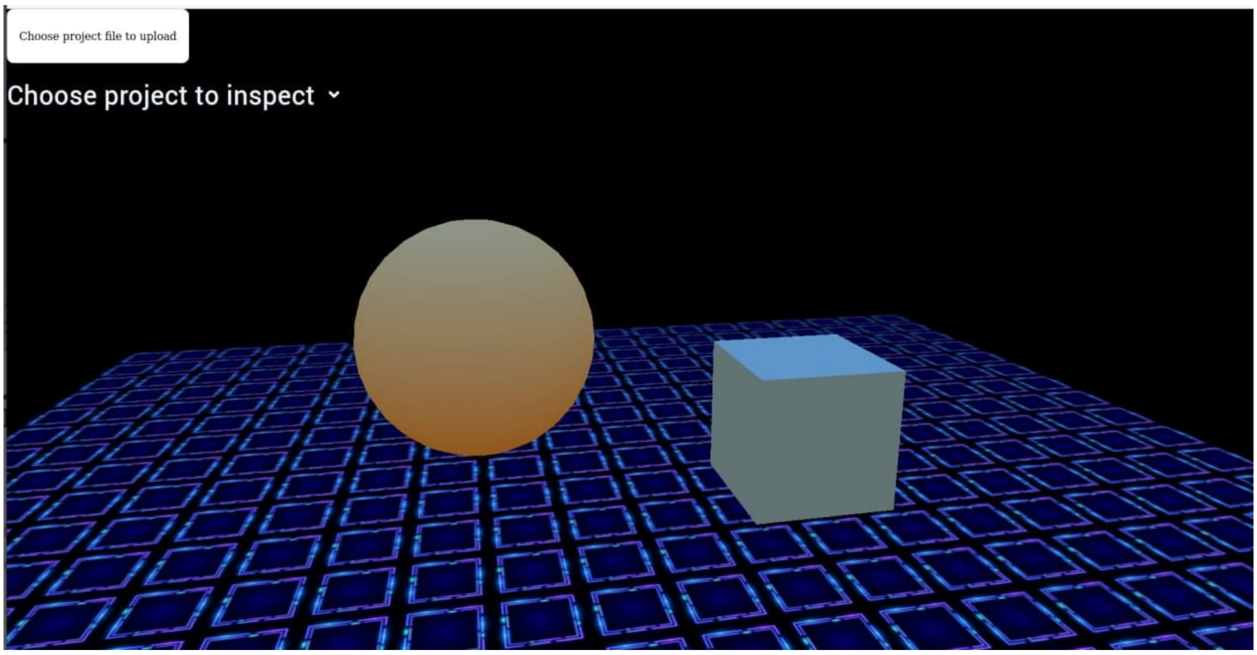
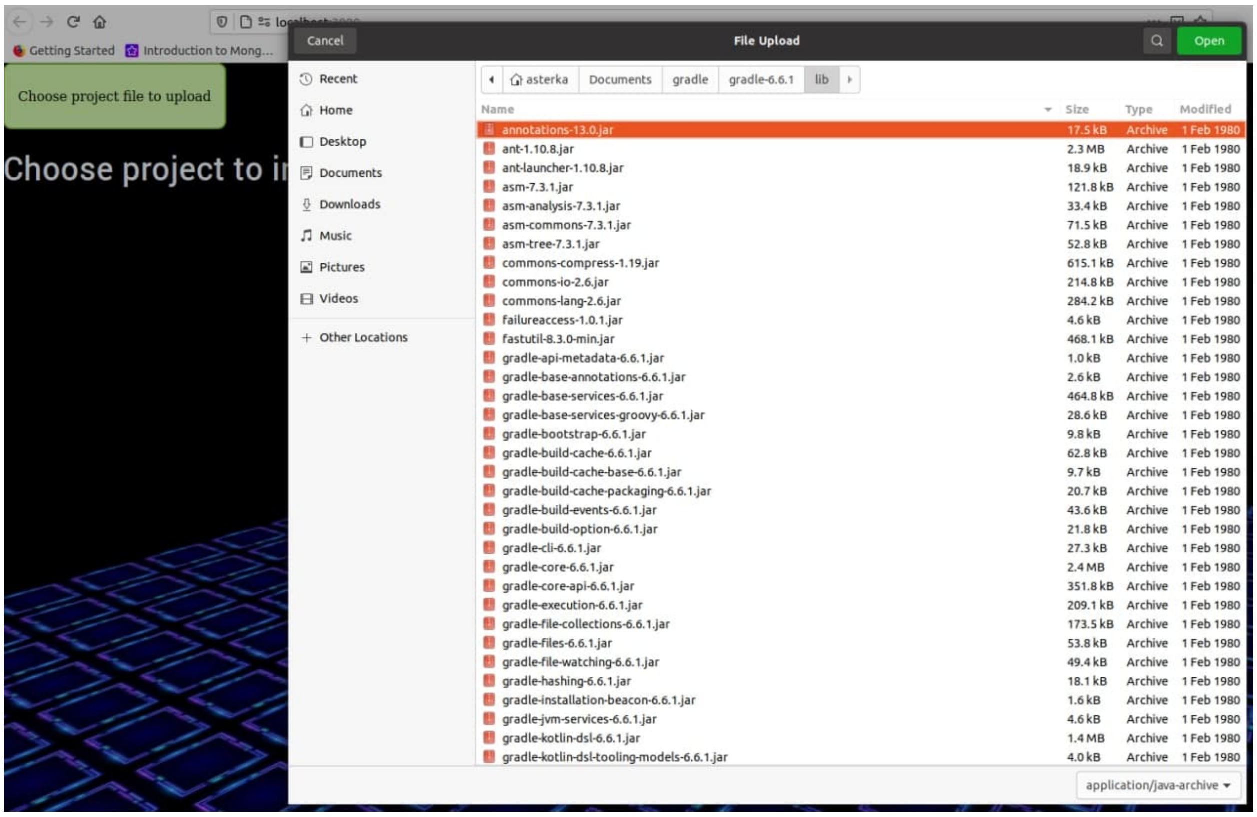


Figure 4.3: Website Entrypoint v0.2.1(minor bug fix, alpha version 2)



Name	Size	Type	Modified
annotations-13.0.jar	17.5 kB	Archive	1 Feb 1980
ant-1.10.8.jar	2.3 MB	Archive	1 Feb 1980
ant-launcher-1.10.8.jar	18.9 kB	Archive	1 Feb 1980
asm-7.3.1.jar	121.8 kB	Archive	1 Feb 1980
asm-analysis-7.3.1.jar	33.4 kB	Archive	1 Feb 1980
asm-commons-7.3.1.jar	71.5 kB	Archive	1 Feb 1980
asm-tree-7.3.1.jar	52.8 kB	Archive	1 Feb 1980
commons-compress-1.19.jar	615.1 kB	Archive	1 Feb 1980
commons-io-2.6.jar	214.8 kB	Archive	1 Feb 1980
commons-lang-2.6.jar	284.2 kB	Archive	1 Feb 1980
failureaccess-1.0.1.jar	4.6 kB	Archive	1 Feb 1980
fastutil-8.3.0-min.jar	468.1 kB	Archive	1 Feb 1980
gradle-api-metadata-6.6.1.jar	1.0 kB	Archive	1 Feb 1980
gradle-base-annotations-6.6.1.jar	2.6 kB	Archive	1 Feb 1980
gradle-base-services-6.6.1.jar	464.8 kB	Archive	1 Feb 1980
gradle-base-services-groovy-6.6.1.jar	28.6 kB	Archive	1 Feb 1980
gradle-bootstrap-6.6.1.jar	9.8 kB	Archive	1 Feb 1980
gradle-build-cache-6.6.1.jar	62.8 kB	Archive	1 Feb 1980
gradle-build-cache-base-6.6.1.jar	9.7 kB	Archive	1 Feb 1980
gradle-build-cache-packaging-6.6.1.jar	20.7 kB	Archive	1 Feb 1980
gradle-build-events-6.6.1.jar	43.6 kB	Archive	1 Feb 1980
gradle-build-option-6.6.1.jar	21.8 kB	Archive	1 Feb 1980
gradle-cli-6.6.1.jar	27.3 kB	Archive	1 Feb 1980
gradle-core-6.6.1.jar	2.4 MB	Archive	1 Feb 1980
gradle-core-api-6.6.1.jar	351.8 kB	Archive	1 Feb 1980
gradle-execution-6.6.1.jar	209.1 kB	Archive	1 Feb 1980
gradle-file-collections-6.6.1.jar	173.5 kB	Archive	1 Feb 1980
gradle-files-6.6.1.jar	53.8 kB	Archive	1 Feb 1980
gradle-file-watching-6.6.1.jar	49.4 kB	Archive	1 Feb 1980
gradle-hashing-6.6.1.jar	18.1 kB	Archive	1 Feb 1980
gradle-installation-beacon-6.6.1.jar	1.6 kB	Archive	1 Feb 1980
gradle-jvm-services-6.6.1.jar	4.6 kB	Archive	1 Feb 1980
gradle-kotlin-dsl-6.6.1.jar	1.4 MB	Archive	1 Feb 1980
gradle-kotlin-dsl-tooling-models-6.6.1.jar	4.0 kB	Archive	1 Feb 1980

Figure 4.4: Project Uploading procedure

Project files upload

When finished the login procedure, the user is ready to submit a project file, that must be a file with '.jar' extension. Since we are defining necessary MIME-types from IANA official registry, that most of the browsers fully support, the user will only see .jar files in the opened window. Having chosen the file, and the button 'upload' clicked, the user will now get the status of her operation on the screen by seeing 'File was uploaded' text in the left top corner as soon as the operation status is returned from the server. In case there is an internal error while running the decompilation procedure, the server will return error opcode to the frontend as well as full description of an error.

Projects choosing

Whether the last upload operation has succeeded or not, user can still watch all of her successfully decompiled and parsed projects by clicking on "Choose project" dropdown menu. In this menu, every time the user clicks, the information is fetched over again. That is how we provide only the actiual information. This might not be an optimal solution in terms of network engagement, but was considered to be quite easy to implement and practical. In the evaluation and further discussion section authors provide other strategies of updating project information in real time systems, which in our stage of development were considered redundant.

Extra features. Canvas

In order to give the best user experience we've come up with an idea to use threejs library on the Website Entrypoint, that uses the same data and

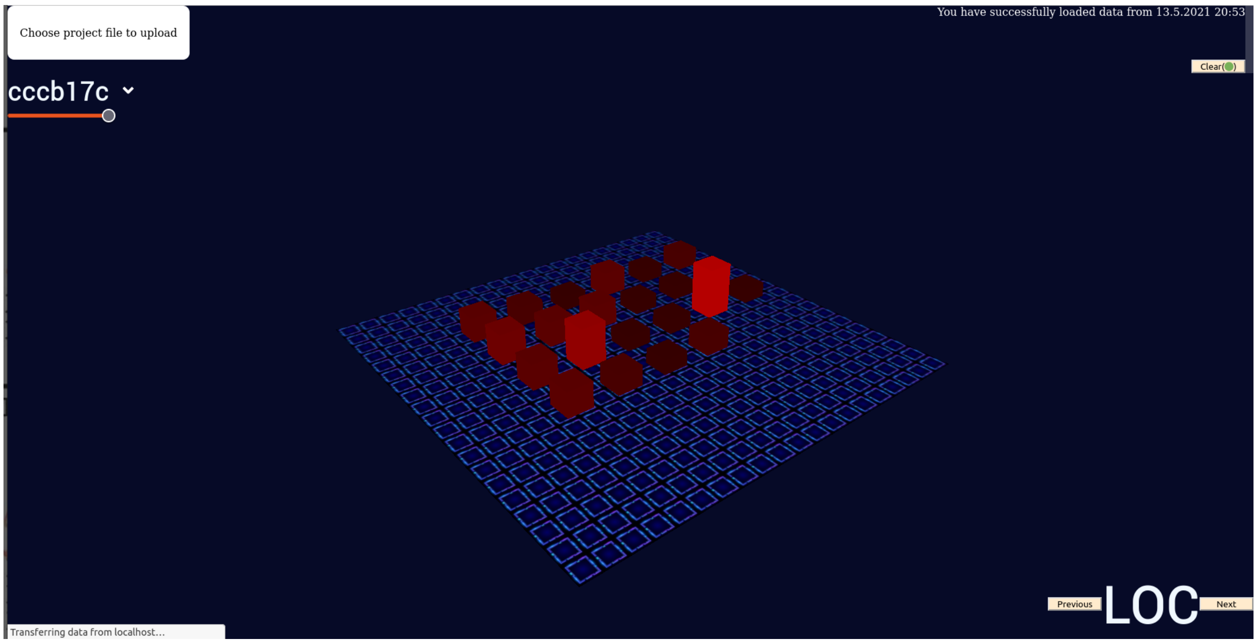
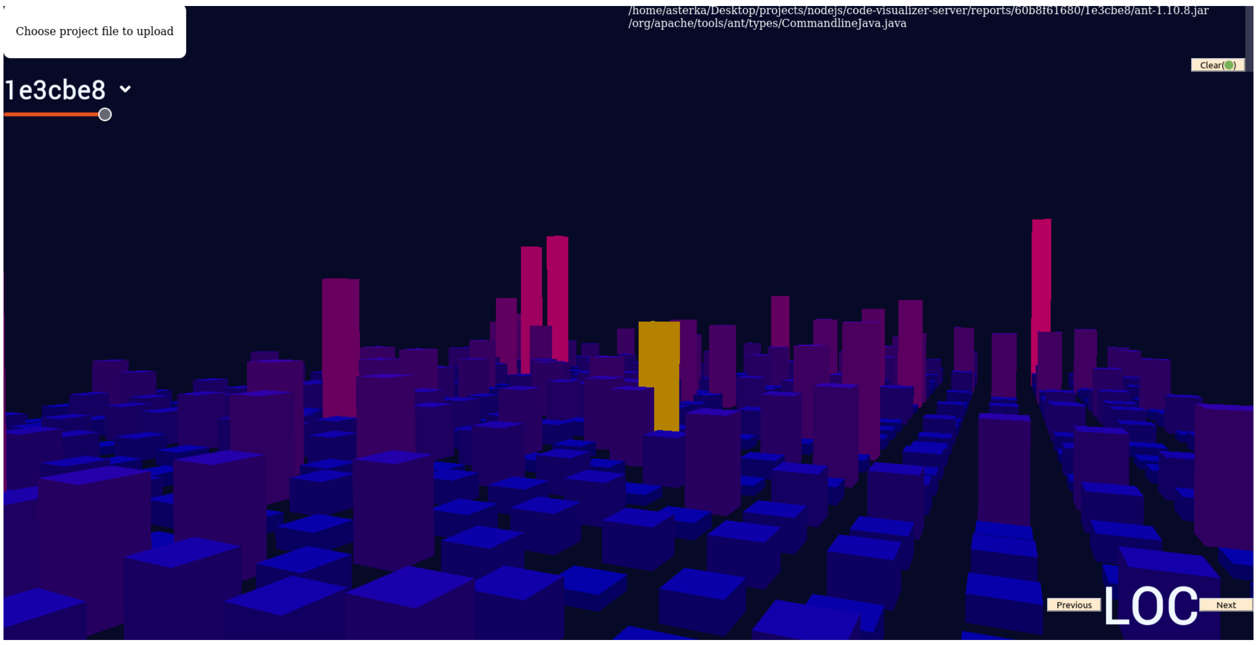

Choose project file to upload
You have successfully loaded data from 13.5.2021 20:53
cccb17c
Clear
Previous
LOC
Next
Transferring data from localhost...

Figure 4.5: Basic visualization of the LOC metric

can visualize it in a similar manner to what is shown on the ViroReact application(section 4.3). In case target device does not have any AR capabilities, user can still interact with parsed data in the browser, and that covers whole target group of users of this system. The modularity of the software allows us to use server capabilities without any frontend, for example, completely remove this Components, still having all the UI interface capabilities. The Canvas compoment only depends on the properties(data) passed to it from the root component. The Canvas renders the same picture as Viroreact Application, because they use the same helper modules that generate the scene and operate in the same data domain.

Canvas itself is a component, that takes all the properties of the scene, such as data, state of the UI controls, selected project, and selected item in the project. In the canvas, based on the data incoming, and a comparator module, invoked in main switch method, the data is being rendered. It is rendered in a so-called Scene if to refer to ThreeJS' terminolody. The scene is an object that can contains children. Children are rendered into the scene from the data, that has been passed from the main component, and take into an account the Metric, that comes from the App component, based on the useState hook. With the help of ThreeJS.Raycaster, that allows us compute intersections for the elements of the scene, taking the camera position, and mouse coordinates, retrieved from mouseclick event, we can obtain the object that is being right in front of the camera by checking the smallest distance. This makes it possible to choose objects of the scene, and change state of the UI, and the rendered picture based on the chosen object. Comparing to the Viroreact Framework usage, which provides any element of the rendered scene, which the user interacts with, with attributes, that can invoke callbacks onClick events, web approach is little more complicated, since there are no DOM elements other than canvas, and the only way to detect an object within the scene is to use a 'laser beam' which what raycasting essentially is. Writing a lot of logic and being able to track the state of the component and pass it in the renderer is a complicated task. In this case, reader might be interested in a completely declarative way of data insertion, which is to use such library as react-fiber. This library will help for the beginners, who have grasped the basic concepts of threeJS library, and React.

To allow user interaction with the objects of the scene, everytime the scene renders it registers an event, which is being triggered on 'click', and is self-destructing event. Based on the results of ThreeJS raycast, we detect the closest element, and trigger setInspectedClass Hook, that tells the scene to rerender, taking the chosen element into an account.



```
/home/asterka/Desktop/projects/nodejs/code-visualizer-server/reports/60b8f61680/1e3cbe8/ant-1.10.8.jar
/org/apache/tools/ant/types/CommandlineJava.java
```

Figure 4.6: Advanced LOC metric with metric filter applied

Extra features. Operation status window

On the right-top corner of Fig 4.5, the reader may observe the Operation Terminal, which was designed in order to provide user with instant feedback. The right top corner is a good place to visualize secondary information, according to [6]. Researchers have tracked the eye movement during website exploration which has resulted in a statistical value of more than 22 percent for the First Fixation in the left-top corner. Around 10 percent of the test participants, though, have started their exploration on the top-right corner, which makes it more than twice more popular than at any of the bottom-positions. Based on this information, we have found it justified to place our extra statistic, that may be used to solve some rare issues, or duplicates the main information on the top-left position of the screen, on the top-right corner, where it still can be easily observed.

4.1.2 Technical details of implementation of Upload frontend

For the Upload frontend we have implemented all the functionality using the React framework and it's Functional components. Functional components allow us to simplify the code by introducing Hooks, that provide us with data binding abilities, and simplify project's state management. Since we have used React ecosystem on both of our frontends (Web frontend and ViroReact application), some of the terms and their definitions will be described in this section that are also present in the 4.3 section of this chapter, explaining the Application Frontend. We have used the latest version of the React Library, but generally, any version, that is later 16.8 will allow hooks, which are essential part of this project.

App Component

The main component, that mounts into the web-page and provides it's children with both Hooks to set the state, and the state itself, is the App component, that is rendered at the component with 'root' id in the main page. Then, there are two main Child components, that are rendered as children in the DOM, which are Canvas and Controls. Canvas is the Components that holds the image being rendered at the screen of the user. User-Controls is a Component, that helps user interact with the software. Every time User interacts with the system, the state, which is passed to the User Controls gets modified by hook. Because of the way that React works with the props and state, it is not recommended to change props in a child component directly. That would make some of the Child nodes, having different state from the

Parent node. By passing the hook instead we have assured that the data would change only in Parent node, which is App, and then the changes would be propagated back to it's children in a consistent way across all the components.

Canvas

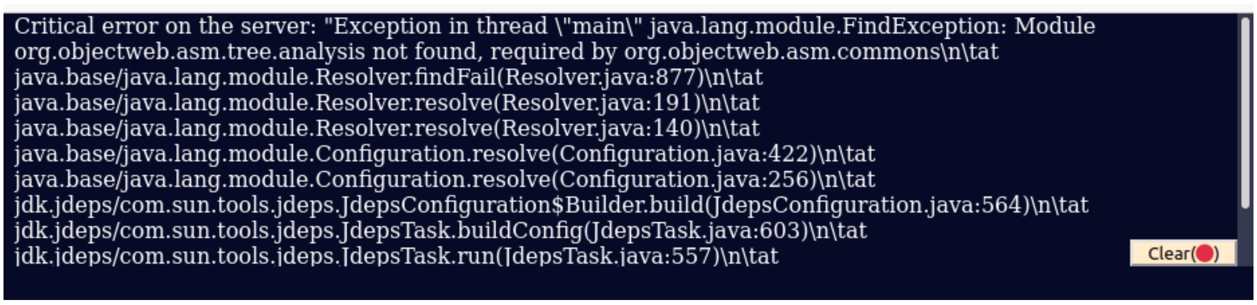
Canvas holds all the data and utilizes useRef to the rendered <canvas> HTML element on the frontend. The useEffect React hooks helps calling renderer function on updates of some parts of the state of the component. This Approach might not be that performant. We tried to optimize the click listeners, that trigger ThreeJS raycaster, but there are still some performace issues, which will be addressed in the Evaluation and Discussion section.

UI components

UI state terminal This UI component contains all the information about the last operation. It has been deigned to handle tuples of data, that is {**opcode**: [1/0], **msg**:"Example"}. The first field of this tuple is the **opcode** that reflects user interaction status. In case User's operation has Failed, the "0" **opcode** would be returned. In case successful operation has been executed, the component will receive and render "1" **opcode** status. In the second field the message that goes with our operation status is saved. That way User will also find some useful information about the action right after it's execution.

Proposed style management

When designing a component system we have thought of making a highly customizable system that any user could adjust for themselves. That is why we implemented all the styles in as a SCSS project, that is transpiled into plain



```
Critical error on the server: "Exception in thread \"main\" java.lang.module.FindException: Module
org.objectweb.asm.tree.analysis not found, required by org.objectweb.asm.commons\n\tat
java.base/java.lang.module.Resolver.findFail(Resolver.java:877)\n\tat
java.base/java.lang.module.Resolver.resolve(Resolver.java:191)\n\tat
java.base/java.lang.module.Resolver.resolve(Resolver.java:140)\n\tat
java.base/java.lang.module.Configuration.resolve(Configuration.java:422)\n\tat
java.base/java.lang.module.Configuration.resolve(Configuration.java:256)\n\tat
jdk.jdeps/com.sun.tools.jdeps.JdepsConfiguration$Builder.build(JdepsConfiguration.java:564)\n\tat
jdk.jdeps/com.sun.tools.jdeps.JdepsTask.buildConfig(JdepsTask.java:603)\n\tat
jdk.jdeps/com.sun.tools.jdeps.JdepsTask.run(JdepsTask.java:557)\n\tat
```

Figure 4.7: Example of failed operation

CSS, and imported in the App component. In order to change styles, user can change the styles folder's contents and recompile new styles for themselves, we have provided the style generation. Project styles for the Upload frontend are compiled using the npm utility called node-sass, via **npm scss** command.

4.2 Our approaches to metric visualization

We have tried to stick to the standard City metaphor, described in the 2 Literature review chapter.

4.2.1 Metric calculation

For the LoCoM, CBO, ECL, and CYCOMP metrics, we calculate the height of out target parallelogram (ViroBox and THREE.boxGeometry) by taking the highest and the lowest value of the metric, compared to the classes in the project, and multiplying the target value by a constant. We then draw boxes of equal width and depth to fit the target space, that we initially provide for the scene.

4.3 Server application

The server application is what the other parts of the system fully rely on. This section describes in detail how this part of the system was designed and what are the purposes of its REST endpoints:

4.3.1 General implementation details

The server part has been implemented in NodeJS using an existing framework ExpressJS to implement static file serving and REST API. Server is configured via config.js. In this file user can set up the local environment, using variables:

parsedResouceHomeDirectory,
dependentModulesDirectory,
runtimeResources

In the first of the aforementioned, User can set up the directory, where files will be saved to and served from. The default value for this variable is */reports*

In the **dependentModulesDirectoy** User can choose the project files directory, where the helper utilities will be installed. The default value for this variable is */project-files*

In the **runtimeResources** User can choose the temporary directory, where the projects would be compiled. The default value for this variable is */res*

4.3.2 Upload endpoint

If we continue on the User Story Upload Project, which we introduced in the previous chapter, the server endpoint /upload is what allows user send their project files, invokes the parsing logic from the parser module, creates all the necessary files in the file structure defined in the Methodology section. Using the Express-Fileupload module, server loads the file into it's subfolder, which is defined by the project-token generated using the uuidv4 utility for each upload request, which is collision free for small number of projects, and a retreived user-token. When the project has been saved server executes an asynchronous callback-function, which itself invokes three different methods from the projectUtilites module, which are

decompileUserProject,
generateMetrics,
parseDependencies.

Method projectUtilites.decompileUserProjects

This method relies on the JD-CLI library, that is a wrapper around JD-Core, allowing us to extract dependencies from Java source code and generate the output in a file, only interacting with the Command Line. This method is executed in an asynchronous context with the help of await-exec library, which is a wrapper around NodeJS *child_process#exec* to empower the execute operation with capabilities to handle promises. We have chosen the jd-cli utility, because it allows batch decompilation, and allows background execution on the server, without any user interaction.

Method projectUtilites.generateMetrics

This function allows user to generate metrics in the same asynchronous way. We address the same batch execution problem in an asynchronous manner by using await-exec library which helps us handle these operations with JavaScript Promises API. This module was designed to execute asynchronously, and to be replaceable with any other CLI-based dependency extractor. For Java projects we have tried two approaches different approaches. First was to use Intellij IDE Metrics Reloaded plugin, that allowed us to export and inspect OOP metrics of Java projects. This tool had it's own advantages, however we have decided not to restrict user to the longest User Flow. In order to get the results user had to first install the IDE itself, then install the plugin, then export the metric, after that upload the data in an .xml format to the server. Many steps have been reduced by turning to a command line utility, called PMD. PMD analyzer helps developer analyze software quality based on static software metrics. It also may help detect programming flaws.

Because of this instrument's flexibility, we provide user with abilities to write their own metric extractors. This is fully described in the github repository of the Server project. In order to add their own metrics, User will only have to place their xml metric description in a way PMD is describing them in the rulesets folder under project resources directrory. The detailed guide to writing custom rules is described at [18]

This step is also asynchronous. Using the Javascript Promises, we map every metric rule file from the pre-defined ruleset, and apply it to the data, that we received from the user calling **Promises.all()**. This starts the parallel execution and effectively increases the time for each metric extraction.

4.4 ViroReact application

ViroReact library is a tool that helps us visualize data that takes interaction with native AR SDK's of both Andriod and iOS systems. The platrform allows developers build React Native-based applications in a consistent way, handling the ARCore and ARKit. Apart form the React Native, as the main architectural concept, we have chosen ViroReact, because of the system requirement that the target system should consist of open-source software. There are proprietary software examples, however. Some of the are quite effective. In case the reader is interested, such projects as Vuforia and [19] should also be considered.

System details

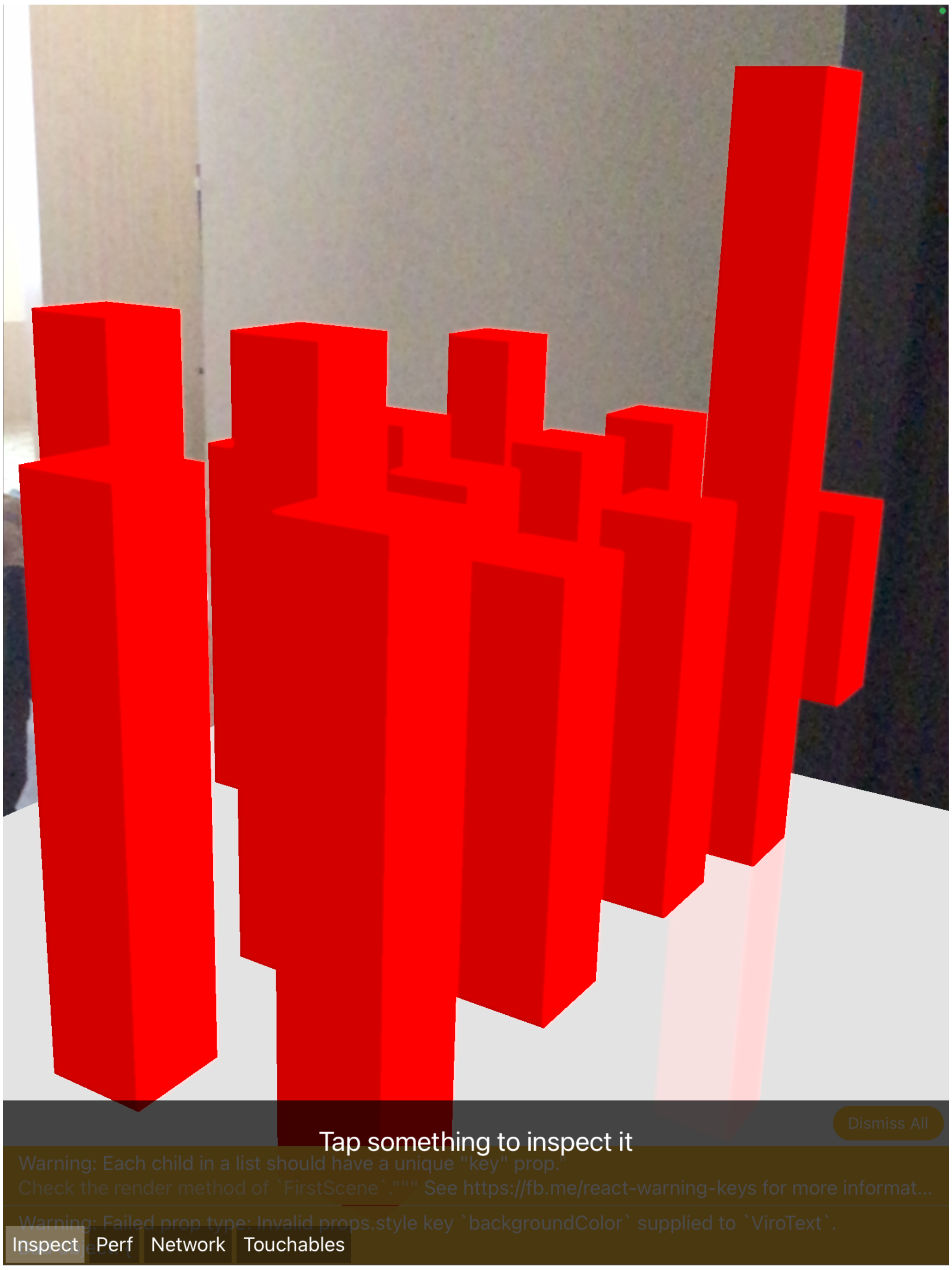
The platform supports various features of the augmented reality by implementing AR-specific components. Here's a short list of the features that we have tried from the target library when implementing the software:

- Camera position handling and scene updates in real-time
- Concurrent background video updates
- AR-based Hit Tests that detect touchable components onclick events
- Virtual 'Portals' between the scenes to allow deep AR experience
- Testbed application that flattens the learning curve and simplifies system architecture

Each of this features, together with a server-side package to provide the application to the testbed application have made the framework our main target for the development. The main Component of the mounted application defines the state of the application and returns the **ViroARSceneNavigator** along with all the scenes that are being passed. The latter is the entry point for the project, that gets the FirstScene component as a target, and passes as a property the **sceneNavigator** object, that allows us interact with the scene. The **sceneNavigator** object contains a stack and the object that is located on top of the stack is being displayed by the Renderer. To maintain and change the state , we pass properties to the next scene in the same object **sceneNavigator** by using the dictionary object called **viroAppProps**. The state management can further be improved, the reader may find the details in the Evaluation and Discussion section of this paper. Initially the FirstScene contained a simple visualization of the Lines Of Code metric, that could be thought of as a simplification of the City Metaphor.

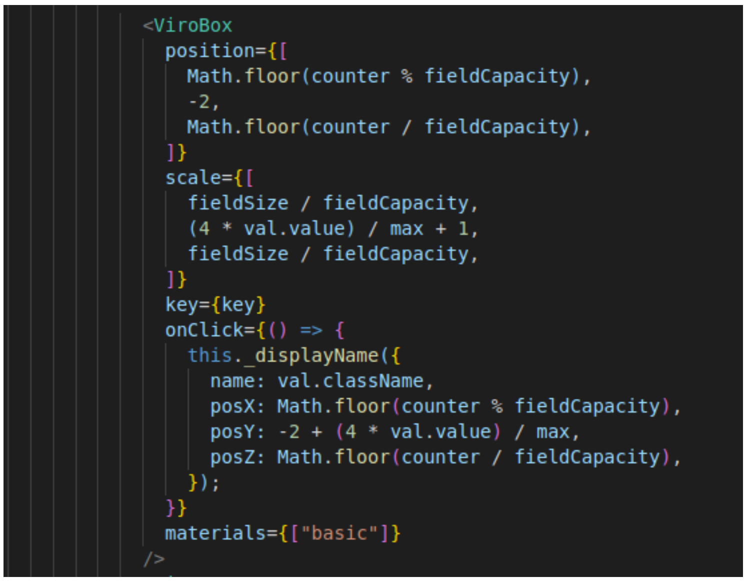
On the Fig 4.8 we may observe our simplifiaction of the city metaphor, that is produced by the code given in Fig 4.9

We iterate over every object of the data, retrieving their value. ViroBox Component contains position, scale and key attributes. It also contains onClick attribute that takes a callback as an argument, which triggers the **this._displayName** function that is set and binds it's context on the parent FirstScene component. This callback sets the current displayed name of the inspected class, and also changes the position to the place right above the ViroBox for that component. That is how the user is able to interact with the retreived data.



```
Warning: Each child in a list should have a unique "key" prop."
Check the render method of `FirstScene`.""" See https://fb.me/react-warning-keys for more informat...
Warning: Failed prop type: Invalid props.style key `backgroundColor` supplied to `ViroText`.
```

Figure 4.8: *Here we may observe the normalized LOC metric statistics, multiplied by the value of 5 for each class of the inspected system*



```
<ViroBox
  position={[
    Math.floor(counter % fieldCapacity),
    -2,
    Math.floor(counter / fieldCapacity),
  ]}
  scale={[
    fieldSize / fieldCapacity,
    (4 * val.value) / max + 1,
    fieldSize / fieldCapacity,
  ]}
  key={key}
  onClick={() => {
    this._displayName({
      name: val.className,
      posX: Math.floor(counter % fieldCapacity),
      posY: -2 + (4 * val.value) / max,
      posZ: Math.floor(counter / fieldCapacity),
    });
  }}
  materials={["basic"]}
/>
```

Figure 4.9: The following is a code for initial visualization

Basic scene navigation

There are many way to graphically represent the differences of the data domains. One approach that we have come up with to visualize our different metrics, is no navigate through the scene, using the sceneNavigator object of the props that are passed to any scene from the ARSceneNavigator Component, of the ViroReact component ecosystems.

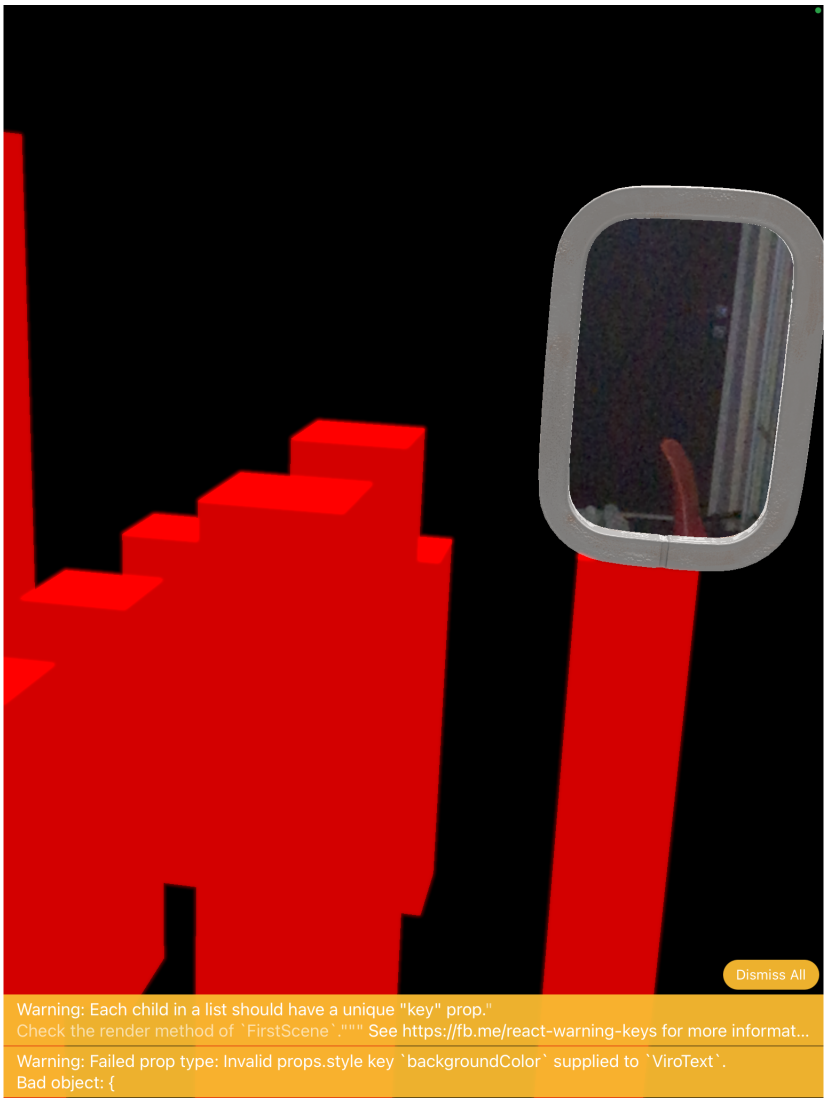
In order to implement such soluion, we have defined a set of basic scenes, as Modules of the software and imported them in our main component, passing them all to the ARSceneNavigator. This basic approach allows us to change scenes based on the click in the UI components.

Advanced scene navigation

The other solution that we've found interesting during our reseach was to use the Component, called Portal from the ViroReact library. The Portal Component allows user to enter a new Protal scene by physically moving into a small window-like object. We've generalized this approach calling it the Domain Portal in Augmented Reality. Intead of switching the domains of the data on user's inputs we propose track user's movement and use it as an input criteria for the data visualization. The ViroPortalScene object makes the scene behind the portal only accessible for such interaction and rendering through that Portal. Each of the Portal Components in this case renders their own part of PortalScene, that includes only a particular metric. This solution gives user a feeling of physically entering the inspected dimension and 'travel' between different dimensions of their source code. Each 'room' that User enters can represent either metric, or user project. The latter option is discussed in the project navigation section in Augmented Reality.

Extra features

As an extra feature, we use the ViroText component, of the ViroReact ecosystem. There is a special endpoint of the server-side, that helps us access the parsed java classes in text format and visualize them as plaintext. When clicking on each of the ViroBox classes, we can inspect the details of the code, as it has been saved at the time of the project decompilation phase. This feature has a great potential of becoming a detached IDE window.



```
Warning: Each child in a list should have a unique "key" prop."
Check the render method of `FirstScene`.""" See https://fb.me/react-warning-keys for more informat...
Warning: Failed prop type: Invalid props.style key `backgroundColor` supplied to `ViroText`.
Bad object: {
```

Figure 4.10: *An example of the LOC metric, rendered as a part of a PortalScene. In this scenario, user has entered the inspected data domain, through our Domain Portal metaphor*

Chapter 5

Evaluation and Discussion

The evaluation of the project can only be considered by the general evaluation of all of it's parts. Here we list the number of test approaches that we have used to test our software during the process of implementation. For all the server-side tests, authors used local network connection and Intel Core i7-7447 processor with 3.1 GHz operational CPU rate(4.2 peak CPU rate). The rendering GPU is GeeForce gtx 1050Ti in it's mobile version.

5.1 Server-side evaluation

5.1.1 Single-user scenario

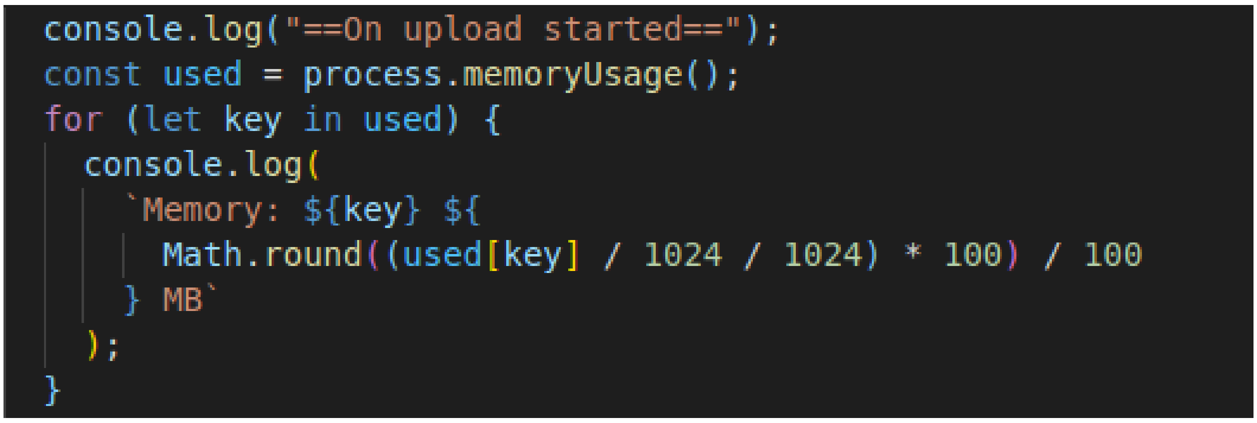
A simple test suit was developed to test out the performance of the server side. We wanted to achieve the concurrent upload for the users. With the same project, containing 118 classes, the project has shown the performance in average total execution time of **5.5** seconds for single upload from the current user. With 20 tries and minimal execution time of **3.492** seconds. This results were obtained by executing **console.time()** and **console.endTime()** operations at the first moment of file receiving and metric file parsing had finished respectively, giving us the total average time, in which the internal parsing processes can execute the commands and finish. Known issues: Due to a mistake in the development, when testing the concurrent uploads from the user, we have found that the removed file at the process of execution of the complexity extraction tool can kill the process of file interaction with the same file. This is rather unusual in production because the user had should submit the same file in the same time, that would force these two procedures to collide. Collision between two different users can not happen, since we generate different tokens for each user, as described in the 4 section.

5.2 Multi-user scenario

To test the scenario with multiple system users we have used the Postman utility. For this scenario, both of the files register the beginning of the operations, by setting the timestamp in the beginning of the request. And registering the end of execution phase, reporting it to file. Average execution time for the same project parsing for 2 users had increased and for this scenario is 3.7 seconds. However, when the number of user increases, the server starts to behave in a quite unexpected way, executing four concurrent operations in a total of 20.39 seconds, as being not concurrent. We've managed to localize this problem down to the module parceMetrics, and the reason might be in the parser module's resource demands at the execution time.

5.3 Memory consumption in get project metrics requests

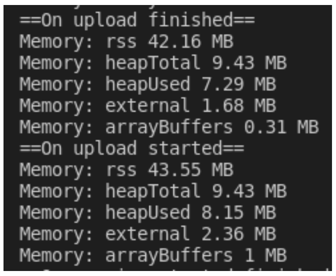
We have also measured the Node processes memory consumption during execution time in our server-side application, using the NodeJS process.memoryUsage(), and interpreting results.



```
console.log("==On upload started==");
const used = process.memoryUsage();
for (let key in used) {
  console.log(
    `Memory: ${key} ${
      Math.round((used[key] / 1024 / 1024) * 100) / 100
    } MB`
  );
}
```

Figure 5.1: Example piece of code that we used in out environment measurement

HeapTotal and heapUsed are the V8 engine's memory usage. The Resident Set Size, or rss represents the main memory consumption on the device for the target process, including all C++ and JavaScript objects and code. This amount can grow drastically in our application, when the number of large files grow. We should address this issue in the future, because that may indicate a memory leak in the application, when the heap size is not growing.



```
==On upload finished==
Memory: rss 42.16 MB
Memory: heapTotal 9.43 MB
Memory: heapUsed 7.29 MB
Memory: external 1.68 MB
Memory: arrayBuffers 0.31 MB
==On upload started==
Memory: rss 43.55 MB
Memory: heapTotal 9.43 MB
Memory: heapUsed 8.15 MB
Memory: external 2.36 MB
Memory: arrayBuffers 1 MB
```

Figure 5.2: Example piece of code that we used in out environment measurement

5.4 Client-side evaluation

5.4.1 Web frontend

For the client-side we have measured the performance of our own page with respect to different factors.

1. Visual performance
2. Resource demand
3. UI interaction

Visual performance and Memory management

To control the FPS on the Canvas, we have attached the simple FPS counter to our rendering function. Every Now that the system rendered the components with the counter, we were able to calculate the average FPS. The results have proven that for the big project with many classes and dependencies, the threejs canvas starts to slow down. To inspect this issue, we've tracked the script's allocated resources and found out that the reason might be in that we don't clear the resources for the ThreeJS scene, in our further work, we will inspect that issue, however, it is not critical for the project, as it is an Extra

Feature.

ViroMedia evaluation

When evaluating the software on the ViroMedia platform, which is a stand for the application, we have considered the rendering performance and re-render implications to the genreal performance. For the same project with 118 classes, and metrics, the rendering cycle took average of 160ms according to the performance monitor statistics. Re-rendering happening in the first implementation is smoothed by the blur effects of the scene changing, so should not affect the overall visual performance. Being on the local network, average delay for the rendering due to the data fetch is 35m.

Chapter 6

Conclusion and Furhter Work

This paper has covered the spheres of data visualization in sphere of Augmented Reality. We have proposed an inexpensive solution to interpret and represent data with the help of immersive technologies. Based on the investigated user feedback, and in accordance with the latest research in thee sphere of human cognition the authors have proposed software requirements for the system that visualizes a subset of metrics for Java programming language. In accordance with the requirements the authors implemented a software and stated the future work that might be conducted in the area. The software was then evaluated against real projects, tested and documented. The designed project is completely open-source, as any of it's components, and extendable. Our systems targets all the ARKit- and ARCore-compatible devices, and provides an inspection tool that is based on the web-browser to imitate the 3 dimensional environment. Authors of the project propose a new UI metaphor for the user interaction in the sphere of augmented reality. Although, a great amount work had been performed, there are many directions to look at when considering the further development. Based on the performance measurements, we propose the following extensions that should be considered.

6.1 Further work on server

6.1.1 Integration with other utilities

In future, server could be integrated with many more utilites. Reader should take it as a note, that different parsers producing different output have theoretically been considered, and the last operation of the server could be re-implemented to work with different static parsers outputs.

6.1.2 Low concurrent performance

Tests have shown, that Server solution might have problems when handling concurrent operations due to costly parsing process. We have also discovered a memory leak in the application, that should be inspected and fixed.

6.1.3 Real-time updates for User Projects

Realtime communications can also be considered in the future. Using the [20] library, or simple Websockets.

6.2 Further work on Client Applications

General improvement in visualization and UI capabilities might be done. We suggest using more metaphors in further research and improve visual characteristics of the scene, that would allow encoding more than one parameter and its value in the scene. For example, ViroBox's and THREE.boxGeometry width and depth could potentially used to encode more quantitative data. Color scheme should also be considered.

There are two main directions to continue the achieved progress on the Client applications in particular:

6.2.1 ViroReact app

It might be useful for production releases to organize the application development, avoiding the development ViroMedia stand app. We also may increase the number of user interactions on the Application Frontend

6.2.2 Web application

The Canvas component needs to be fixed to acquire less resources. More metrics of software Design quality might be added. To follow the same Declarative React patterns, the Canvas Component might be reworked in the future.

Bibliography cited

[1] J. Vincur and I. Navrat Pavol Polášek, “Vrcity: Software analysis in a virtual reality environment, in international conference on soft-ware quality,” *International Conference on Software Quality, Reliability and Security Companion(QRS-C)*, pp. 509–516, 2017.

[2] R. Souza and et al., “Skyscrapar.2222em an augmented reality. visualization for software evolution,” *Brasilian Computing Society*, pp. 17–24, 2012.

[3] R. T. Azuma, “A survey of augmented reality. presence: Teleoperators and virtual environments 1997,” pp. 23–32, 2018.

[4] A. Paivio, “Mental representations: A dual coding approach,” 1990.

[5] M. Kuniecki, “The color red attracts attention in an emotional context. an erp study,” 2015.

[6] R. Grier, “Visual attention and web design,” pp. 63–64, 2004.

[7] C. Jeffery, “The city metaphor in software visualization,” Jan. 2019.

[8] I. E. Sutherland, “A head-mounted three-dimensional display,” pp. 757–764, 1968.

[9] L. Voinea, A. Telea, and van Wijk, “Cvsscan: Visualization of code evolution,” 2005.

[10] R. Chidamber and F. Kemerer, "Towards a metrics suite for object-oriented design," pp. 197–211, 1994.

[11] R. Chidamber and et al., "A metrics suite for object-oriented design," pp. 476–493, 1994.

[12] T. J. McCabe, "A complexity measure," December 1976.

[13] S. Cant, D. Jeffery, and B. Henderson-Sellers, "A conceptual model of cognitive complexity of elements of the programming process," *Information and Software Technology*, vol. 37, no. 7, pp. 351–362, 1995.

[14] Avgerou, "A review of the methodologies movement," pp. 277–286, 1993.

[15] L. Aversano, C. Grasso, P. Grasso, and M. Tortorella, "Investigating differences and commonalities of software metric tools.," 2017, pp. 249–256.

[16] M. Welsh, S. Gribble, E. Brewer, and D. Culler, "A design framework for highly concurrent systems," 2000.

[17] "Kegel, dan: The c10k problem," Tech. Rep., 2021. [Online]. Available: http://www.kegel.com/c10k.html.

[18] "Making rulesets," Tech. Rep., 2021. [Online]. Available: https://pmd.github.io/pmd-6.35.0/pmd_userdocs_making_rulesets.html.

[19] "Easyar commercial software.," Tech. Rep., 2021. [Online]. Available: https://help.easyar.com.

[20] "Socket-io utility," Tech. Rep., 2021. [Online]. Available: https://socket.io.