University of Szeged

Institute of Informatics

Visualizing static source code analysis results

Master's Thesis

Author:
Günter Peter Manz
Student of Software Engineering

Supervisor:
Dr. István Siket
Assistant Professor

Szeged
2020

Proposal

The topic of this thesis is to create a graphical viewer for the OpenStaticAnalyzer command line static source-code analyzer. The GUI will be responsible for displaying information generated by the analyzer, including various metric values and violations, as well as code duplications in an easy-to-read and informative way. The project should be well maintained and easily extendable with new features. The solution should work the same on all mainstream operating systems (Windows, Linux, IOS) and should be easy to install and use.

Summary of content

- Topic title: Visualizing static source code analysis results.
- The given task: Getting to know the OpenStaticAnalyzer in order to learn how to process its output data. Research of related literature and of related software that performs similar tasks. Based on the acquired knowledge, develop and implement a graphical interface that is able to transparently display the metrics and code duplications calculated by OpenStaticAnalyzer. Selecting a programming language and ecosystem suitable to develop the application. Designing the software architecture, developing and releasing it.
- Solution: Design and implement the software responsible for the graphical display of the information according to the specification prescribed at the workplace[1] and using the acquired knowledge.
- Used technologies and methods: OpenStaticAnalyzer, TypeScript, WebPack, React, Redux, NodeJS, HTML, CSS, npm, Git, Python, BASH, Linux, Windows
- Achieved results: Based on the task description a web application was created which is able to process and display the output of OpenStaticAnalyzer. It is able to display metric values for building blocks of code as well as the syntax highlighted sourcecode of projects with emphasis on coding rule violations and duplicated source code snippets. It can display code duplications, coding rule violations and software metrics as well. The software paints a picture of the overall code quality of a project and the change of quality with time. It is able to handle single or multi-lingual projects (C/C++, C#, Java, Python, Javascript, RPG)
- Keywords: server-client, GUI, source code qulity management, TypeScript

[1] My workplace (at the time) was the Department of Software Szoftware Engineering at the University of Szeged

Contents

Introduction

Thesis structure

Chapter 1 discusses source code quality assurance and introduces the concept of static source code analysis as a tool for quality tracking. It discusses when and why it is important to assure the quality of software using automated tools and what kind of different tools are currently available on the market. The OpenStaticAnalyzer static analysis tool is introduced here, on which the project-work of thesis relies upon.

Chapter 2 explores simmilar works. Such as a similar project I worked on previously called SourceMeter plugin for SonarQube is introduced [14]. Deniel Dzadik, a former Masters student at SZTE, also developed a program with similar functionality [18]. These two solutions and the SM-Riport as a third option are compared here with their pros and cons listed.

Chapter 3 is about planning and the design of the project. The following questions are discussed here: What specifications and what information was available before the project was started. Where does the input come from, how will it be processed? How will the extracted data be displayed to the user? What kind of different technologies will be helpful for this task? How will the finished product be deployed to the end user?

Chapter 4 discusses the technical details by highlighting some important code snippets and the overall source code structure.

Finally in chapter 5 the finished product is presented, along with the implemented features accompanied by some screenshots.

Chapter 1

Topic Introduction and Motivation

We live in an age, where almost anything can be done virtually. Virtual bank accounts, online shopping, online gaming, research, teaching and studying, social life, all of these things can be accomplished online. When you get a job with a higher education diploma, your work will most likely consist of sitting in front of a computer, using various programs like Microsoft Office, a web browser, Skype and so on. Software today truly is everywhere. But what makes software good? How can you measure software quality? And more importantly how can you ensure good software quality and its consistent maintenance? These are the topics explored and answered in this chapter.

Every software is created using some kind of source code written in a programming language (just to name a few popular ones[1]: Python, Java, JavaScript, C, C#)[7, 6] The source code contains a list of precise instructions that describe the complete behavior of the software. On a big project, there is usually multiple people working on the same code simultaneously. Obviously this enables an earlier release date, but it also complicates management of the source-code greatly, as

- no developer will know ALL of the code in the software, since he is not the sole person writing it. This is why there is usually a software architect making all the design decisions in such a project.
- duplicated snippets of code may occur, these are parts of code, that more or less provide the same functionality and a single one instance of them could be reused to provide the functionality of all the other ones.

[1] in order of popularity in 2020

- the source code may become difficult to understand (eg.: too complex, or too many undocumented components and algorithms). This is simply because different people think and solve problems in different, not mutually obvious ways.
- some small, overlooked and forgotten errors from half a year ago, can snowball into deal breaking issues over time, that could have been fixed in 5 minutes originally, but now consume several hours of debugging and fixing.
- usually there is also a stylistic issue: every programmer is used to a different code formatting style, if code style is not being controlled somehow, reading the code becomes very tiring.

So if the project is not carefully managed, the source code will very quickly become a confusing mess, that

- is full of bugs and unintended side effects
- might have low performance
- possibly has security vulnerabilities
- is very hard and costly to maintain or add new features.

There is severl ways how to address theese problems:

- with a good version management tools like git [10]
- by project management tools like GitLab or GitHub
- proper automated and, if nescessary, manual testing of the sofware
- using quality measurement tools like SonarQube, FindBugs or various linters
- with automatic code formatters like prettier.

The focus we will shift our attention to is the quality measurement tools. Theese tools mainly utilize 'static program analysis', which means that analysis of the software is performed without actually executing it. Instead the analysis only looks at the source code directly and tries to compile some useful information on its overall quality and what kind of improvements can be made in the code and where.

Static analyzers can extract various information from the source code:

- various metrics - these are numerical values that are calculated for most components of source code. Metrics measure various aspects of quality:
  - **Size metrics** measure the basic properties of the analyzed system in terms of different cardinalities (e.g. TLOC “Total Lines of Code” és LLOC “Logical Lines of Code”).
  - **Documentation metrics** measure the amount of comments and documentation of source code elements in the system.
  - **Complexity metrics** measure the complexity of source code elements (typically algorithms). A commonly known complexity metric is McCC “McCabes’s Cyclomatic Complexity” metric [11].
  - **Cohesion metrics** measure to what extent the source code elements are coherent in the system. HEFF [8] “Halstead EFFort” or MI “Maintainability Index” are typical examples.
  - **Coupling metrics** measure the amount of interdependencies of source code elements.
  - **Inheritance metrics** measure the different aspects of the inheritance hierarchy of the system.
- possible bugs and vulnerabilites - a list of positions in the source code, where bad practices have been found. Usually not only the problem is highlighted, but a possible way to fix it is also provided. Things like unused variables, unclosed streams, unclosed files or network connections fall under this category.
- duplicated code lines - simmilar code lines that dont have to be exactly identical, but their functionality could be moved into a common superclass or a single function reused multiple times, just with different parameters.

Quality can be measured and quantified using the metrics, the number and severity of bugs and violations. Using these metrics - which have been calculated solely from the sourcecode itself and are thus always up to date - it is easy to form a realistic and unbiased opinion of the systems quality and value. Having this kind of unbiased and always up-to-date picture of code quality is key to creating good software.

1.1 Why is continuous quality assurance important?

Every software degrades with time, no matter how well planned, its quality will degrade, because:

- the continuously changing needs of the customer introduces more and more unforeseen changes to the code, which are thus harder to implement and introduce unforeseen errors and malfunctions during operation.
- because of the short deadlines developers often have to rush their work which impacts code quality in a negative way.
- inexperienced developers often use the wrong tools, or not the most optimal tools for a task.
- quality of the source code is often not measured objectively using quality assurance tools.
- the software lacks automated testing of features.

Every time the source code changes, especially if new features are introduced, the frequency of errors will increase. Later, recognised errors are fixed (which means changing the code again!). But not all errors are obvious to see and so a small amount survives this stage of development and accumulates over time. In order to avoid this, it makes sense to define project standard thresholds for different metrics, for example: the ratio of documented code lines to the lines of code that contain program instructions, has to be above 0.4. Or in order to keep the program simple, elegant and understandable, keep the Halstead difficulty of every method below 10. A list of these thresholds can be defined and automatically tested prior to every release and even after the tiniest modification of source code (because its automated, that means it is basically for free).

1.1.1 Cost-effectiveness of quality assurance

Releasing reusable, easily maintainable, completely bug-free, simple and clear code is surely the ideal goal of a software engineer. But is it feasible? Setting up the development environment, installing and configuring the code-quality tools takes a lot of precious work-time. Every code change needs to be double checked and most of the time reworked

again and again until the code conforms to the defined metric thresholds. This consumes work-time, so a company might ask the question: “why should i care about the quality of my source code, if it is going to take two or three times longer to develop the same program with the same functionality, with the only difference being that the quality of its source code is - perhaps just slightly - better?” It is difficult to find a good model and give a definite answer to this question.

One way to think about source code is an investment in the future. Imagine you have a team working on a software for three years. The company wont want to throw that code away after spending all that money into its development. That code should be reused, maybe the whole project can’t be used again as a whole, but bits and peaces of its code, some algorithms, components and the general software architecture are generally reusable in other projects. Source code reusability can be measured using coupling metrics, so you can measure your sourcecode reusablity during development and strive towards reusable code. Also, who wants to reuse bad and buggy code? The code you write today is an investment into the code you will use over and over again in the future. This is, in my opinion, a solid argument for the nescessity of monitoring and ensuring software quality.

1.2 Tools for static analysis of source code

There are many tools providing static analysis for source code. I would like to highlight and introduce some of these.

1.2.1 SonarQube

SonarQube is a very popular static analyzer, written in Java and mainly supporting the analysis of Java programs, however support for other languages can be added via plugins. With the use of plugins the platform is able to scan projects written in many different programming languages including Android (Java), C, C++, CSS, Objective-C, COBOL, C#, Flex, Forms, Groovy, Java, JavaScript, Natural, PHP, PL/SQL, Swift, Visual Basic 6, Web, XML, Python and more. The SonarQube platform has a web based front end. An online demo is available at `https://sonarcloud.io/explore/projects`. It has a very well organised graphical user interface, (especially when compared to most other open source static analyzers) that aims at handling every language in the same way,

providing the same high level metrics. A single global quality metric gets calculated for each project, which tells the user whether or not the project meets the quality standards (yes/no value). this makes it easy to quickly assess that the quality of a project is okay but a very in-depth look at the finer aspects of quality, like for example component metrics is not possible, since the platform only supports file level metrics - in order to have the same interface for every language. However many languages are structured much finer than just the file level (eg. Java has packages, files, classes, interfaces and methods). Metrics of this finer granularity are unfortunately lost within SonarQube.

1.2.2 Code Warrior

Code Warrior[3] is a multi language scanner for Linux/BSD/MacOS. It supports C, C#, PHP, Java, Ruby, ASP, JavaScript. It comes with a web interface, where the code violations are listed. It only finds code violations and does not calculate metrics or find duplicated code lines. It is said to have a low rate of false positives[17].

1.2.3 SpotBugs

SpotBugs is the successor of the FindBugs tool. It is an open-source static code analyzer written for Java. It detects possible bugs in java programs and classifies them into four severity categories. It has a standalone GUI application and can be integrated in form of a plugin into most popular Java IDEs like Eclipse, Netbeans, IntelliJ IDEA. Additional rules can be added to customize the scanner.

1.2.4 OpenStaticAnalyzer

This tool is special in a sense, because it is developed at Szeged [16] and the project of this thesis focuses on visualizing the output of this analyzer. OpenStaticAnalyzer supports popular programming languages such as Java, Python, JavaScript, C/C++, C# and RPG. It comes in the form of command line utilities, a different one for every language. The complete output is saved to an xml file, but some additional csv files containing metrics for components are also generated to make at least part of the output human readable. There is no official graphical viewer for the tool at the time of writing. The tool calculates various metrics at the component level, uses third party language specific utilites to find

bugs and coding rule violations. Moreower it is able to identify and track so called Type-2 syntax based code duplications throughout the project lifecycle. This means, that these so called “clones” can be tracked during the whole software lifecycle ensuring that later discovered flaws get fixed in all clones as well [1].

Chapter 2

Related Similar Projects

In this chapter I would like to introduce a few other projects that provided inspiration to this thesis.

2.1 SourceMeter plugin for SonarQube platform

OpenStaticAnalyzer is a free and open source fork of SourceMeter, therefore the two are very simmilar and can be substituted with each other. The difference between them is that OpenStaticAnalyzer is completely open-source and for free, where SourceMeter has additional features and these come with a price. I have previously developed the GUI of the "SourceMeter plugin for the SonarQube platform" [14]. I have also written my Bsc thesis around the work on that project [13]. This plugin incorporates some features of SourceMeter into the SonarQube platform. When properly configured the platform manages the scanning of projects and displays the results in the SonarQube web based front end. Since SonarQube only supports display of file level metrics and SourceMeter is able to calculate metrics down to the function/method level, therefore not all of the information generated by SourceMeter could be displayed. To fix this the plugin API of SonarQube was put to its very limits[1] in order to add the functionality of displaying more results under a new menu page of every project. This page displays the metrics for the package, class and method level, meanwhile it also lists the code duplications. The main issue with this project was that over the course of time some essential parts of the SonarQube plugin API (parts that our plugin relied on) were deprecated and removed. It was hard to keep

[1] This means that the API was used in ways it was propably not designed to be used, because there was no other way.

up and as the API was already heavily used it became very hard to continue development and adding new features to this plugin [2]. A lot of the SourceMeter output could not be displayed in the SonarQube platform, for the reasons already described in section 1.2.1, the SonarQube platform operates with a subset of metrics that are common to every programming language, thus many language specific metrics would not appear on some parts of the GUI. Same thing with the code duplications: even though SonarQube has support for code duplications, there is no API that allowed registering duplicated code lines for the plugins, so the duplicated code lines recognised by SourceMeter can not be displayed in the SonarQube way. Because of these deal breaking issues there are no new features planned for the "SourceMeter plugin for SonarQube" project. It is still being actively maintained, so that it still works, but essentially development was halted.

2.2 A simple viewer for SourceMeter graph files

Deniel Dzadik Zozémusz implemented a basic Viewer using cutting edge (at the time) technologies (like React and Redux, which are still considered modern). This viewer was very basic:

- it is able to extract data from the binary output file of a SourceMeter analysis
- it displays a few basic charts
- most of the data is displayed in a raw table format, like an excel sheet (metrics and vulnerabilites).

It is a very lightweight viewer, it is simple and fast. It lacks many features when compared to the "SourceMeter plugin for SonarQube platform".

Chapter 3

Planning and Design

In this chapter the whole planning process of the software is documented. Used technologies are introduced along with the motivation behind using them. The reasons for deciding upon the software architecture and the programming language are also argued.

3.1 Architecture

I have been heavily inspired by the SonarQube Platform, during the time I developed a plugin for it. I liked the idea of a server-client solution, where the server manages all of the data and the client solely manages the display and visualization. In this model the server runs on a separate machine and one or more clients can connect to that machine and view the analysis data through their web-browser. This model has a lot of flexibility, it is possible for instance, to view source-code analyses of a project without having the source-code actually downloaded and available. This is useful when a company has to keep source-code of a project and the raw analysis files confidential. This approach also allows the software to be installed on a server machine, which is configured to daily fetch the new version of a projects source code, analyze it and serve up-to-date quality information to multiple clients The whole development team can then browse the analysis data on their web-browser, without having to set up the same environment on their computer. There is a bit more planning involved to get this architecture right, but it is definitely worth it for the flexibility that is gained. It can still be deployed in the form of a standalone application, in that case the server and the client is simply on the same machine.

3.2 Language

3.2.1 Java

Since this is a project started from scratch, a decision had to be made what programming language we choose to create the software with. Selecting a language is final and cannot be changed later on during the projects life-cycle. Therefore this has to be done right. Based on the requirement, that the finished product has to run on multiple operating systems, with a minimal number of dependencies, really only two languages came into consideration, it had to be either Java or JavaScript. Java runs on any popular operating system (Windows, MacOS, Linux...) and the Java virtual machine is pre-installed on most personal computers, which speakes in its favor. Both the server and the front end can be developed using Java and allthough Java is perfect - in my opinion - for developing server-side headless applications, it pales next to HTML + JavaScript, when it comes to developing GUI applications. Nevertheless Java is a valid choice for this kind of application and it was the default go-to choice for some time (pre 2010).

3.2.2 JavaScript

HTML + JavaScript, aka a web page, is great for developing front end applications, but it used to lack back end functionalites. This means, that the server side code had to be written using a different language, like for example PHP, Java or C#. This also meant that objects or interfaces had to be defined twice (once in the back end language and once in the front end language), resulting in longer code that is more difficult to maintain, because when the interface changed code had to be changed manually in multiple places. This changed in 2009 with the first release of NodeJS. NodeJS brings JavaScript to the back end, making it possible to develop web servers or any kind of software written in JavaScript that also needs access to operating system calls. It was now finally possible to reuse back end code on the frontent or vice-versa using JavaScript. JS also has very good package support. Packages can be installed simply via the “npm” command line package manager[1] and packages exist for almost anything one could need. This also makes JavaScript a viable choice.

[1] by simply typing “npm i package-name” into the console

3.2.3 TypeScript

TypeScript is a superset of JavaScript mainly introducing statically typed types to the language. But it also extends the language in a way that makes it much easier to work with object oriented design paradigm, by introducing class syntax simmilar to Java, interfaces and enums. This fixes one of the largest issues I personally have with JavaScript: it gets very difficult to keep track of a variable's type on large projects. With TypeScript this is no problem, since every variable has a type assigned when it is created. Since TypeScript is a superset of JavaScript and compiles down to plain JavaScript, it is possible to use plain JavaScript code within TypeScript, which means that when developing using TypeScript one inherits the benefits of the complete JavaScript ecosystem and its package manager (‘npm‘). So far TypeScript is the best language suited for the job, because:

- it is possible to develop both the server and the client using TypeScript
- code can be re-used between the server and the client.
- the language has great package support (inherited from JavaScript)
- it is a high level language that makes development fast and comfortable.

3.2.4 Other aspects to consider

The data that will be displayed is provided in form of a binary “*.graph” file generated by OpenStaticAnalyzer. A custom “GraphLib” library is used by OpenStaticAnalyzer to create this file and the same GraphLib can be used to extract information from it. This GraphLib library is written in Java, which is a reason to use Java instead of Typescript after all. Luckily someone else pawed the way already: Deniel Dzadik has found a way to transpile the GraphLib from Java to TypeScript preserving its full functionality [18]. In summary, for the reasons described above TypeScript is best suited for this project on both the server and the client side.

3.3 Used technologies

3.3.1 Git

Every project needs version managing. The go-to version management software for source-code today is Git. It also happens to be my favorite, as a developer, I prefer Git above all other tools around today. Git really changed the way developers think of merging and branching. With Git, these actions are extremely cheap and simple and they are considered one of the core parts of your daily workflow. In contrast to CVS/Subversion, where merging and branching has always been considered to be somehing scary, that only advanced users should would do only every now and then. With git this is basically the first thing you learn. It is very cheap to experiment with different versions using git, since the changes are only local at first. If the changes prove to be good, they can be pushed to the origin repository, where the code from all developers comes together. The developers can review every change and after some maintainers approve as well, the change is persisted on the main branch of development. Git allows for easy back-and-forth switching between different versions of development. Git is a good tool, but as it is with every tool: you can use it right and you can use it wrong. If it is done right, the work of an arbitrary number of people can be coordinated with its help. If it is done wrong, it will be difficult to keep track of even two developers work and synchronizing their work. There is going to be at least two of us working on this project in the future, so a good branch model is needed. I plan to adopt the branch model called git-flow.[5]

Figure 3.1 illustrates how the git-flow model is designed to make parallel development of different features simple, while also keeping continuous releasing a simple strait forward procedure that can even be done in parallel with active development. Every feature is developed on its own feature branch, after completion it is reviewed by other developers. At this point some problems are usually found in the code, that the original developer will then fix. When all problems have been fixed and “the review passed” and the change is merged into the develop branch. When release is due, a new release branch is started on the current develop branch. This can be done at any time during development. Testing starts on the release branch and fixes to bugs are pushed to this branch. When the branch gets stable (it works and has no known bugs to fix), the release branch will be merged into master. This way master is always stable and contains releasable source-code.

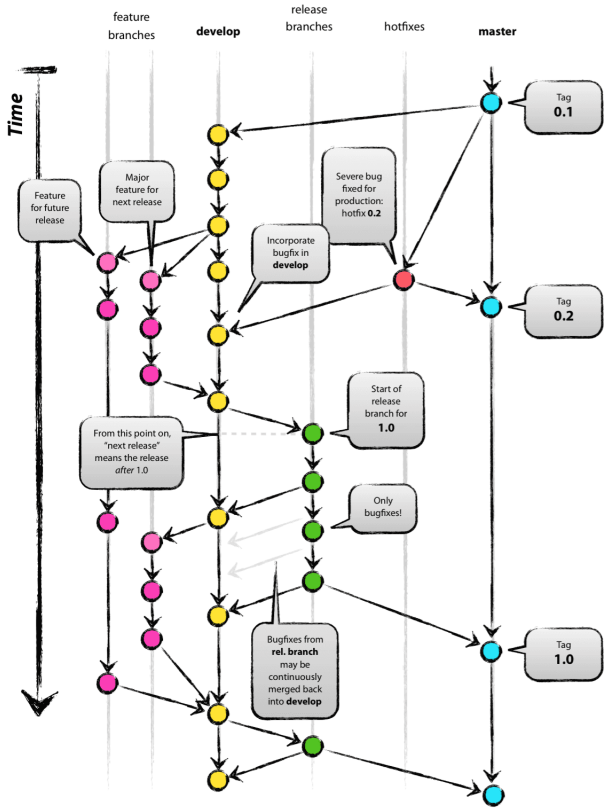

Figure 3.1: The git-flow branching model

3.3.2 GitLab

Git is decentralized by design, but it is still the norm to have a central repository that developers sync with from time to time. GitLab is an online service providing such a central git repository. GitHub is a simmilar service, but there was no real choice in this matter, my workplace had an internal Gitlab server and that is what had to be used. But GitLab provides much more, than just a central repository for git. It is a fully featured project management solution, with a very good workflow. For every development task an issue is created. This issue describes explicitly what has to be fixed/implemented/changed. Various tags can be attached to the issue, like for example priorities, "todo", "doing", "under review", "closed" and so on. When the issue gets the "todo" tag, it means that it can then be picked up by any developer that likes it, this means its "todo" tag will get removed, instead it gets assigned the "doing" tag and its assignee changes to the person who picked it up. This ensures, that there is never two people working on the same issue. In our model every issue must have its own branch in git (called feature-branch), where the changes to resolve the issue are implemented in the source code. Once development on that issue is finished, a merge request has to be submitted, asking to merge the feature-branch into the

main develop branch. This merge request will then be reviewed by a senior developer, to make sure that there is no obvious problems with the code. If an issue is found with the new code it will be discussed and has to be resolved by the original developer. This way new developers can learn from their mistakes and improve. When the senior developer finally approves of the changes in the merge request, the branch gets merged into the develop branch and the merge request and the linked issue will automatically get closed.

3.3.3 Prettier and git hooks

Prettier is an opinionated code formatter. It is able to automatically format source-code to be consistentently formatted and therefore more easy to read. Git hooks is an advanced feature of git. With the help of hooks it is possible to extend the functionality of git, customizing it for every project. The user can specify scripts that git will automatically execute on certain events. For example, a script can be called every time someone wants to commit new code to the repository, which performs a check on the new code. If the code is badly formatted, the commit will be set to fail. To ensure code readability and consistent formatting, the project will have a git hook set up, that only allows code to be committed that complies to prettiers coding style standard. This guarantees that the whole source code of the project is always nicely and consistently formatted without having to check every source file all the time.

3.3.4 Web front end

The front end technology that will be used is React from facebook. React is one of the currently modern web frameworks on the market. There are three simmilar frameworks for JavaScript/Typescript to choose from: Angular, Vue and React [4]. Angular and React have a large and strong community behind them, they are very popular. All three are component based frameworks, which makes code-reuse easily accomplishable.

Angular is the most mature of the three, it was first released in 2010. It is developed by Google and used in many google products. The framework has a steep learning curve, the user has to learn and understand a lot about how the framework works before he can start developing something. But it is a complete package, once mastered, it fullfills every need.

React was initially released back in 2013, it is developed by Facebook. React is simple and intuitive to use, because the component behavior is not separated from the UI in the code. For a long time it was the most popular framework on the market. Stability is a very important aspect for Facebook during the development of React, since it is used by big companies like Twitter, Airbnb and well, Facebook itself.

Vue is a third contender. It is not backed by any large company, like React and Angular. It was created by ex-Google employee Evan You in 2014. Development is slow, the project and mainly gets funding on Patreon. Over the last three years, Vue has seen a substantial shift in popularity, even though it doesn't have the backing of a large company like Google or Facebook.

3.3.5 Redux

Large applications need some sort of clear protocol, where and how all of the state data is saved. There is a lot of this data and most of the time it is just being saved as global variables, but that brings too many problems to the table. It is better to use a product specifically designed for this task. Redux helps you write applications that behave consistently. It centralizes your application's state and logic, enabling powerful capabilities like undo/redo, state persistence and much more. It also works very well together with React components, making it possible to access the Redux store with the same ease, as the internal Component state.

3.3.6 Webpack

Webpack is a bundler for web applications. The project consists of source files written in TypeScript and it also uses React. While it is possible to use React with TypeScript, it is not very straight forward. Webpack is a very powerful tool - if it is configured in the right way - and is used to orchestrate the whole compilation process of the project, on the front end as well as on the back end. In order to compile the usable front end and server from the source files Webpack performs the following steps:

- compile *.ts files to JavaScript using the tsc typescript compiler.
- load other *js files as if they were already compiled *.ts files.

- resolve and load dependencies of the source files as well.
- Link and merge all of the compiled JavaScript source code into a single file called “bundle”
- minify (“compress”) the generated source code, to be as compact as possible.
- generate source maps, so that when an error occurs while running the finished bundle, the error can be traced back to a position in the original source-code, where it can then be fixed.

Webpack has a powerful tool called webpack-dev-server. It is an integrated development server which supports compilation and hot-reloading code - that means that when the sourcecode changes, the changed files and only those are automatically recompiled and the running application is reloaded with the up to date code. This feature makes development using webpack much faster and easier. Webpack will also be used on the back end, to compile a single runnable javascript server file, from the many input files.

3.3.7 Input

The input data for the project is given in the following way: There are multiple projects that have been analyzed multiple times using SourceMeter or OpenStaticAnalyzer. The analysis results for every project are saved in a specific directory structure, see Figure 3.2. A project can be written in multiple languages, which will result in there being multi-

```
project name
└─ language ..... one of {'java', 'javascript', 'python', 'cpp', 'csh', 'rpg'}
   └─ datetime ..... timestamp
      ├─ project name.graph ..... binary file, contains all analysis data
      ├─ project name.xml ..... text file, contains all analysis data
      └─ ...other OpenStaticAnalyzer output files
```

Figure 3.2: Data structure of a projects analyses generated by OpenStaticAnalyzer

ple language directories. Every project has at least one or more analyses. Every analysis is done at a certain point in time, this datetime is the name of the directory where OpenStaticAnalyzer stores the data of that analysis, for example: `ProjectA/java/2019-12-19-12-00-00`. The input for our application is a directory, containing multiple project analyses of the format described in Figure 3.2 All the data, like metrics,

rule violations and duplicated code lines will are stored in the .graph file, this file contains the complete analysis data. This file will be used to extract data from, since it is more efficient to extract data from a binary file than it would be to extract it from the xml file, that contains the same data in a textual format. OpenStaticAnalyzer uses a custom library called “graphlib” to create, read and edit this .graph file. The graphlib was written in java, which makes it hard to use in this project, since we use TypeScript. Luckily the graphlib was transpiled from java to TypeScript during a previous project [18] and now there exists a functionally equvivalent version of it for TypeScript, that we can use.

3.3.8 Deployment

Once deployed the finished product will be runnable on Linux, MacOS via a shell script, as well as on windows via a batch script. The only dependency is having nodejs installed, which is available for any platform. The script launches the server and the front end as well. The input is a folder on the host machine, the folder location can be customized inside of a config file.

Chapter 4

Implementation

In this chapter the structure of the source code, its most important components and the internal operation of the system is described. I will cover the parts that are my work, the parts which I did not write, I will at most mention in the description of file organization - since organizing was done by me.

4.1 Source-code Ownership

There are three developers in total (András Balog, Bence Nagygyörgy and me, Günter Manz) who have committed to this project myself included. Let me precicesly define the extent of my contribution to this project and which parts of code I authored. I was the one who took over the role of the architect, designing its structure and configuring the entire environment. Later two developers joined to help me with the project, as it had grown quickly. I have mainly worked on the client sub-project. The server was written almost entirely by Balogh András and later Nagygyörgy Bence, I just made a few minor adjustments to the server code sometimes and performed code reviews on it. The projects-generator subproject was my idea and also written almost entirely by my hands. See the list of files that have been written by me in Appendix A. Figure 4.1 shows the number of added/deleted/modified source-code lines per developer. The values used have been generated using the tool git-quick-stats [12] on the source code repository. Notice the amount of my contribution to the project being significantly larger than in the case of the other developres.

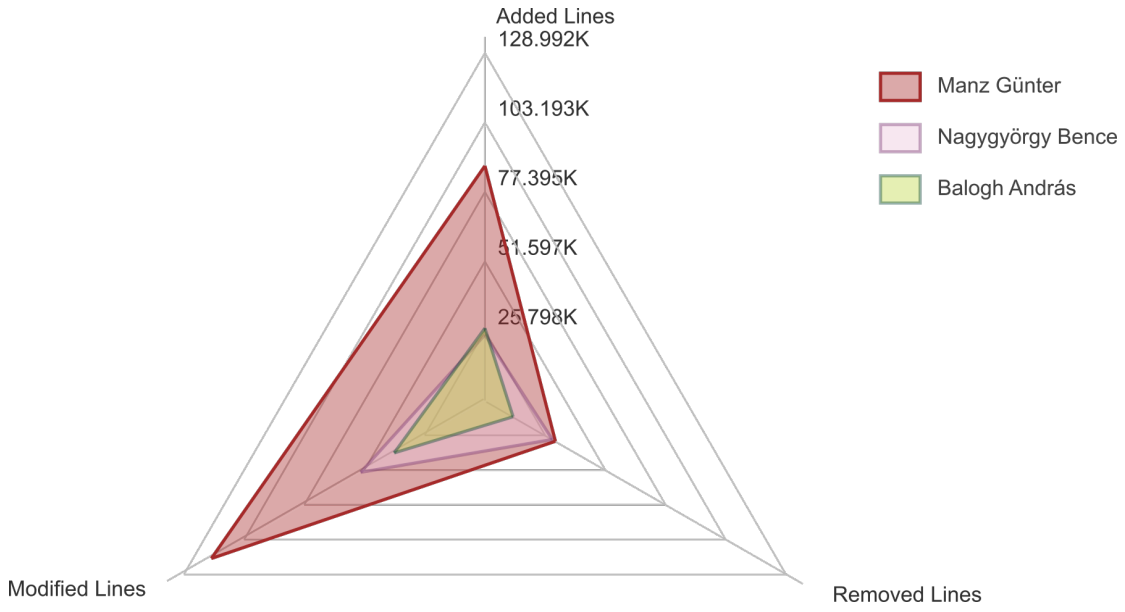

Figure 4.1: Spiderchart showing the number of added/deleted/modified source-code lines per developer

4.2 Structure

4.2.1 Sub-projects

The main application can be separated into 3 distinct parts:

- **Server**
  - loads data from the file system and in some cases modifies it
  - processes and manages the data
  - provides API endpoints that the client uses to fetch data
- **Client**
  - displays data
  - handles user interaction
  - communicates with the server using its web API
- **Common**
  - contains common data structures and global constants used by both the server and client

There is also a 4th sub-project, which is not part of the main application:

- **projects-generator** - automated testing project set creator
  - automatically downloads the source code of various software, with respect to every supported language of OpenStaticAnalyzer
  - checks out multiple versions of the downloaded projects source code and saves them on the file system
  - analyzes every version with OpenStaticAnalyzer, creating a set of analyses that can be used for testing. This set of analyses will contain the same data on every machine, independant from the OS. This is nescessary for consistent testing of the application.

4.2.2 File Structure

in Figure 4.2 you can see a partial file structure, showing the coarse structure of the project and its sub-projects (server, client, common, projects-generator).

```
sm-riport
├── server ..................................... Server source files
├── common ..................... Contains files used by both server & client
│   ├── interfaces ................. Data structures describing the server API
│   ├── model ..................................... Other common data structures
│   ├── libs/graph ..... Used for reading the input. Imported from [18]
│   ├── consts.ts ..................................... Global constants
│   └── util.ts ..................................... Global utility functions
├── client ..................................... Client source files
│   ├── src
│   │   ├── components ..................... All UI building blocks
│   │   │   ├── routes ..... Components describing complete web pages
│   │   │   └── App.tsx ..... Entry parent component
│   │   ├── redux ..... Contains files for application state and data management
│   │   └── util ..... Various utility functions
│   └── index.html ..... Entry GUI file
├── projects-generator ..... project for generating test-input set
├── scripts ..... git hook scripts
└── deploy.py ..... deploy script written in python
```

Figure 4.2: Data structure of a projects analyses generated by OpenStaticAnalyzer

4.3 Client Data Management

Accessing data from the server is a core functionality of the Client and was designed and implemented by me. The client, in other words the UI uses Redux for internal data management. Data can originate from user input and the server. Some data is only stored on the component level using component state. More persistent data gets saved in the redux - global - state. The structural composition of the redux state (also called store) can be found in the file `client/src/redux/store.ts` and is illustrated in Figure 4.3.

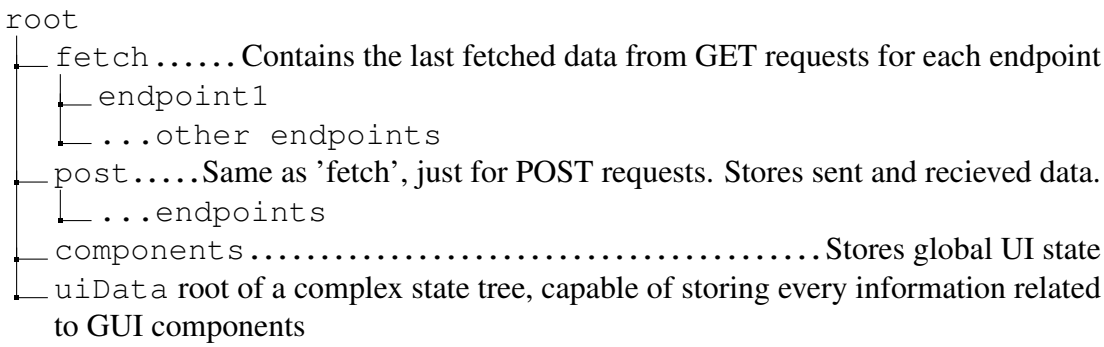

Figure 4.3: Data structure of the Redux store

4.3.1 Managing server data

There is an issue with fetching data from a server, as only plain text data can be sent, in our case it is in JSON format. This is a JavaScript markup format, so it is easy to parse it into a run-time JavaScript object. But the TypeScript type cannot be inferred and it is lost in this procedure. Having accurate types is essential to fluid programming, since the IDE relies on types for suggestions. The developer cant be expected to remember every property name, function name and also their types for every variable in the project. Luckily types can be hinted to the TypeScript compiler.

GET requests are automatically handled in a way that makes linking types to api endpoints straight forward and readable using the generic `FetchHandler` and `PostHandler` classes from `client/src/redux/generic/`. These heavily utilized classes are used to create a Redux reducer and an action creator function. They link an API endpoint with a typescript Type and the created reducer and action creator are both respecting this type. Later the reducer is used to add an entry to the store, where the

fetched data can be accessed. The action creator is used by the components to initiate API requests with the proper parameters. The progress of the request can be monitored with built in variables in the store called:

- `fetching` - true if API call was started, but it is not yet completed
- `fetched` - true if API call was started and completed (with or without errors)
- `isError` - true if API call completed with an error
- `error` - an error object describing the error or null
- `data` - the received data object ready for use

POST requests are handled similarly, only that in this case there is two data types that have to be linked to the same endpoint: the type for the sent data and another one for the received data.

As an example of how simple it is to use these see the following code snippets:

```
const RequestHandler = new PostHandler<
  Params,
  Response,
  PostState<Params, Response>
>("code")
```

This creates the `RequestHandler` object which will handle POST requests where the data that is being sent is of type `Params` and the received data is of type `Response`. After instantiating the object an action creator can be instantiated:

```
const sendRequest = RequestHandler.getAction("/something/save")
```

But in order to access the result in the store, the related reducer has to be registered first in the store like so:

```
const rootReducer = combineReducers({
  fetch: combineReducers({
    // fetch reducers...
  })
  post: combineReducers({
    theRequest: RequestHandler.getReducer()
    // other post reducers...
  })
})

// create and initialize the redux store like normally:
export const store = createStore(
  rootReducer,
  composeEnhancers(applyMiddleware(thunk))
)
```

From this point on the function sendRequest can be used to send POST requests to the "api/something/save" API endpoint on the server like so:

```
const par: Params = { value: 1 }
sendRequest(par);
```

The result will be accessible under in the store after the request completed:

```
const recievedData: Response = store.post.theReducer.data;
```

All the interfaces - aka the types - used to communicate with the server are stored in the common project inside of the interfaces directory. Theese types are accessible from both the server and the client, which ensures that the types used by both are always consistent. The `common/interfaces` folders underlying directory structure mirrors the server web-API in such a way, that for every request the used interface can be found explicitly. For example the `GET /data/project` requests response type can be found in the `common/interfaces/get/data/projects.ts` file. And the `common/interfaces/post/data/something/save.ts` file contains the Response and Params types for the `POST /data/something/save`. So this folder can also be interpreted as an api documentation.

4.4 Routing

The GUI client application is a so called single page application. This means that there is only a simple html file that is being served. All content, every user interaction is created using JavaScript, even links that seem to point to another page are actually handeled by the same page, the JavaScript code recognizes which part of the application should be made visible. That is, routing is handeled by JavaScript, by the react-router package. All routes are defined in the `client/src/components/App.tsx` file, a route mapping consists of a url matching expression and a Component. When the url entered in the browser matches a route mapping, the associated component is automatically loaded and some parameters can be parsed from the url string. The mappings for the project can be seen here:

```
<BrowserRouter basename="/">
  <Header>
    <Switch>
      <Route exact path={"/projects"} component={ProjectList} />
      <Route exact path={"/projects/:name"} component={ProjectDashboard} />
      <Route exact path={"/projects/:name/codebrowser/:path?"} component={CodeBrowser}
          />
      <Route exact path={"/projects/:name/metrics"} component={ProjectMetrics} />
```

```
8       <Route exact path={"/projects/:name/:lang/:id/metrics"} component={Metrics} />
9       <Route exact path={"/projects/:name/:lang/:id/rules"} component={Rules} />
10      <Route exact path={"/projects/:name/:lang/:id/clones"} component={Clones} />
11      <Route exact path={"/projects/:name/:lang/:id"} component={AnalysisDashboard} />
12      <Route exact path={"/projects/:name/:lang/:id/clonesviewer"} component={
            ClonesViewer} />
13      <Route exact path={"/ug/:lang/:params"} component={UsersGuide} />
14      <Route exact path={"/settings"} component={Settings} />
15      <Route path={"/home"} component={Home} />
16      <Route exact component={NoMatch} />
17    </Switch>
18  </Header>
19 </BrowserRouter>
```

According to the input data structure described in subsection 3.3.7 the input data can be structured the following way: there are analyses, programming languages and projects. Each analyses belongs to exactly one language and exactly one project. Every project can have multiple languages, to which multiple analyses may belong. Following this structure, to identify an analysis you need to provide

1. the name of the project
2. the target language
3. the number/index of the analyses in the date-time ordered list of all analyses

And in order to identify a specific project it suffices to provide only the project name. Following this structure, two different levels of data are distinguished on the client, there is:

- project level data, which is relevant for the whole project
- analysis level data, which is relevant for one specific analyses

In order to get any kind of data from the back end, sending the project name that the requested data belongs to is always nescessary, that is why the default `Route` component that ships with the `"react-router"` package has been extended to automatically put the currently active project into a special place inside of the redux store, that can be accessed from any component. This extended component can be found in the file `client/src/components/routes/AbsRouteComponent.tsx`. It is used transparently in `App.tsx` under the name of `Route` (as if it was the default one shipping with the `"react-router"` package). The extension procedure is explained by in the following code snippet:

```
render() {
  const Component = this.props.component
  const rest = { ...this.props }
  delete rest.component
  delete rest.projectList
  delete rest.selectedProject
  delete rest.setCompState
  delete rest.fetchProjectList
  return (
    <Route
      {...rest}
      render={props => {
        let selectedProject = getProjectByName(
          props.match.params.name,
          this.props.projectList
        )
        if (selectedProject) {
          this.props.setSelectedProject(selectedProject)
        }
        this.props.setRouteParams({
          project: props.match.params.name
        })
        return <Component {...props} />
      }}
    ></Route>
  )
}
```

4.5 Code Browser

The application has an integrated code browser that is capable of displaying the source code of an analyzed project, with syntax highlighting, supporting custom text highlights (changed background and foreground for any section of text), displaying source code violations inline and also marking duplicated code lines. The component can be parameterized with following parameters:

```
code: string[]                      // source code to display
startingLineNum?: number            // owerride the starting number of the line numbering
    . default 0
violations?: IndexMap<Violation[]> // object containing violations to display in the
    code line margin
clones?: CloneInstance[]            // object containing clones to display in the code
    line margin
highlights?: Highlight[]            // array describing highlighted sections of text
scrollLine?: number                 // line number, scroll here initially default = 0
path?: string                       // if supplied, this filepath will be displayed on
    the top of the browser
```

The component uses an external package called highlightjs to initially syntax-highlight the source code. This package takes the raw code as a string and creates an html markup out of it. This markup is then post-processed character for character using parse5, another external tool. Parse5 is a streaming parser for html, this means that it is extreamly efficient. Additional html tags are injected using parse5, theese tags provide the additional

custom highlighting. The Source-code is then split up into lines and each line gets rendered on its own separate component. The Line component manages the layout of the code line using css grid.

4.6 Clone Browser

Duplicated lines of source code can be found using the code broowser, since theese lines get highlighted, but browsing explicitly for clones is not efficient in this way. There is a special tool for this, we call it the `CloneBrowser`. This component lists every Clone Class in the analyzed project. A clone class is simply a set of related Clone Instances. The user can choose a clone class from the list and after that specific clone instances can be selected as well. The Clone browser then shows all relevant information for the selected instances: the Metrics, the source-code and it also highlights the parts, where the two instances differ. This highlight is useful for a developer, because it helps during the reorganization of code, when elliminating Clone Instances by moving code to more abstract functions/classes. The highlighted, differing parts are the things that have to be parameterized.

As was already mentioned, the `Clone Browser` also displays the source-code snippet of the selected Clone Instances. This is of course done reusing the `Code` component I introduced in section 4.5. The `Code` component, as most components that i wrote, is written in a way that allows for easy reuse.

4.7 Metrics Table

Displaying various metrics about components of the source code is handeled by the `MetricTable` component. This component is used multiple times in the project, since tables of code metrics are displayed on multiple places. As the name suggests, the component displayes the metrics in a two dimensional grid, much like an excel sheet. Whats special is that the component uses virtualization technology and on top of that, the first row and first two columns are both frozen in place. This is accomplished by displaying 4 different tables and keeping the scroll positions manually synced up in just the right way. Virtualization means that when the page is loaded the table does not get rendered

completely, only those rows are rendered, which are actually visible at the time. This way rendering is blazing fast and not slowed down by the sometimes tremenduous amount of rows that would be generated. The remaining rows will be rendered when the user scrolls the table, always removing the rows that are not visible anyways and inserting the ones that will shortly scroll into view. Virtualization is handeled by the `"react-virtualized-auto-sizer"` external package.

Chapter 5

Description of the Finished Product

In this chapter the finished product is characterized, accompanied with screenshots of important parts that I have worked on.

After setting up and launching the application and after copying analyses of some projects to the configured “projects” folder, the user has to use its favorite web-browser and navigate to localhost:4000. The user is then presented with the list of projects available, as seen in Figure 5.1

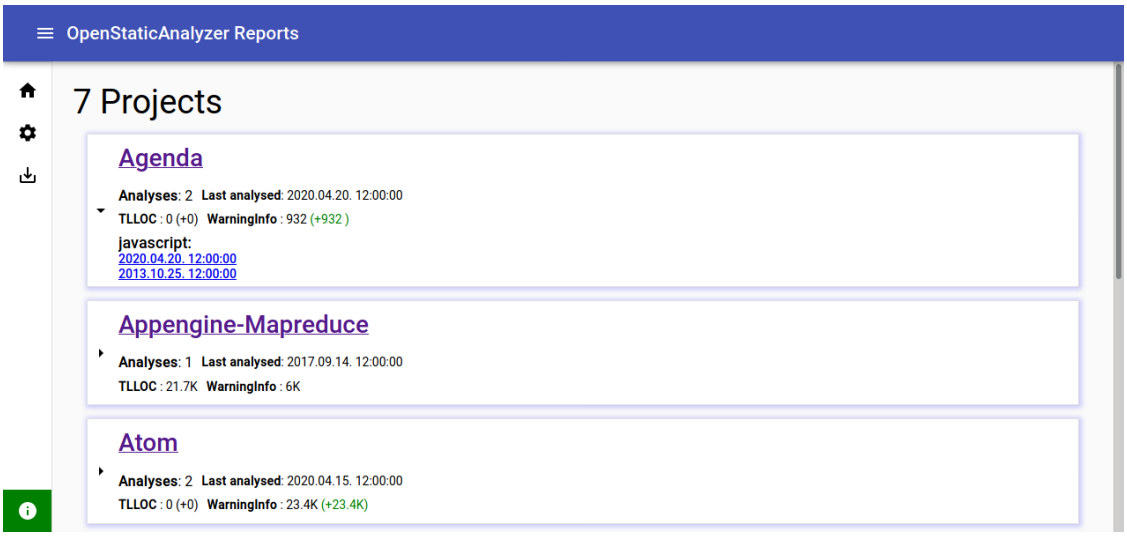

Figure 5.1: The main page showing the list of projects

Some descriptive metrics are already shown here, such as TLLOC (Total Lines Of Logical Code) which gives an idea of the projects size. When the user clicks on the name of any project, the project history page for that project will be displayed.

5.1 Project History page

Following the page flow from top-to-bottom, see Figure 5.2 for this, we first see the name of the current project, as well as its current size and the languages in the project. Next, there is a Code composition bar chart, showing the distribution of simple code lines (blue), duplicated code (in red) and documentation lines (green). Then we see various tables with a few customizable metrics and the change in these metrics relative to the previous version. The list of metrics displayed and layout can be configured in the `client_config.json` file. There are two line charts: "Rule violations history" and "Metrics history".

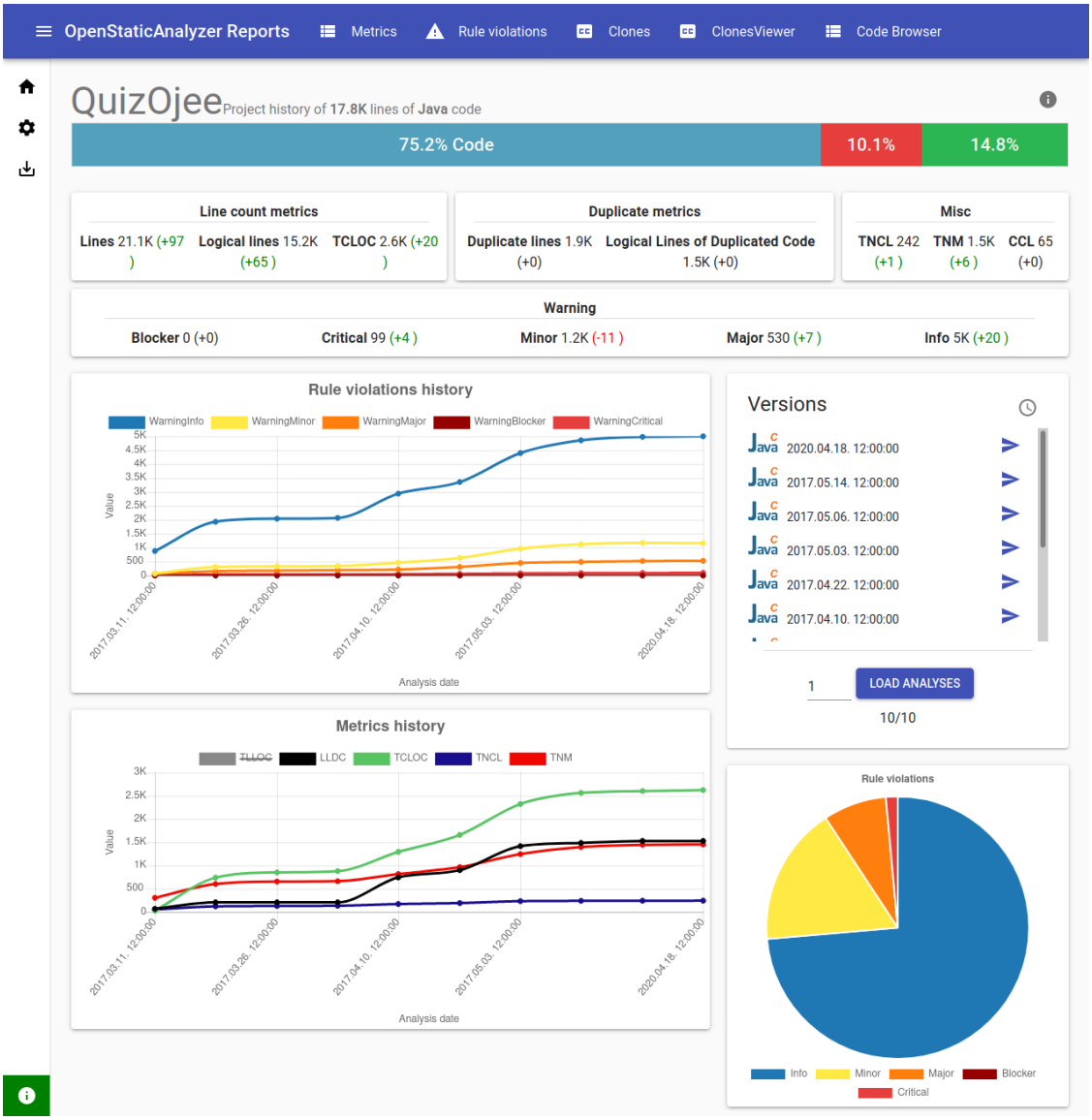

Figure 5.2: The project history page showing general information about the projects quality in the present, as well as the change of quality with time in the past.

As the name suggests, the “Rule violations history” chart shows the number of violations by violation severity throughout the lifetime of the project. Below, on the Metrics history chart the change of some key metrics can be tracked. In addition, on the pie chart the current distribution of violations can be seen, again, grouped by violation severity. Every analyzed version (which corresponds to single analyses) can be individually opened when clicking on the arrow next to it in the Versions block. Notice, that the content of the appbar on the very top of the page has now changed and new pages are available. Also note that navigating back to the list of projects can be done from anywhere by using the navigation bar on the left side and clicking on the home button (the house icon). Let us now discuss what the new pages are that just became available through te top-side navigation bar. Since the “Rule violations” and “Clones” pages are not my work I will not describe these in this thesis.

5.2 Metrics

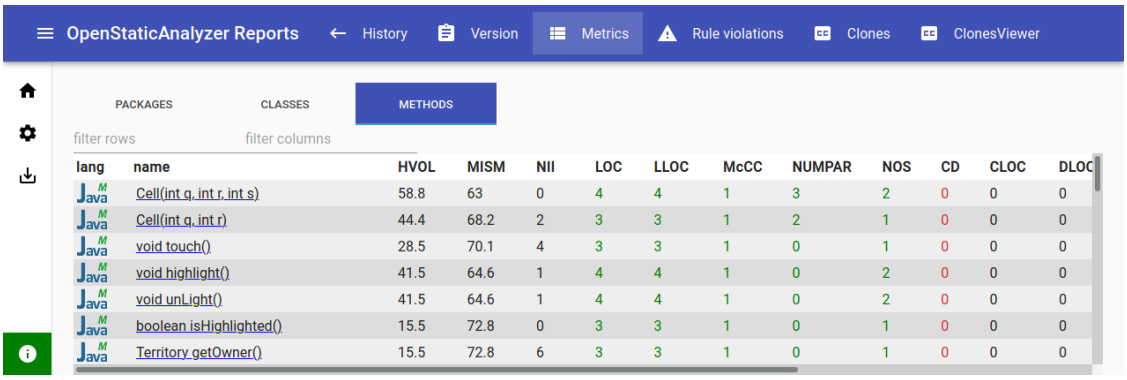

Figure 5.3: The metrics page listing metrics for various software components: packages, classes, methods

The user can select the desired type of software component, whose metrics are of interest. The table can be filtered by the name column using the “filter rows” input above the header row. Using the filter columns input, the list of columns can be filtered keeping only the ones that are of interest. When a column header is clicked, the table content is ordered according to the columns content in decreasing order, when clicked again it changes to ascending order. When hovering over the column header cell a tooltip appears displaying the full name and description of the metric in that column. A screenshot of the metrics page can be seen in Figure 5.3.

5.3 Code Browser

Figure 5.4 shows the code browser page. The screen is split into two parts. On the left

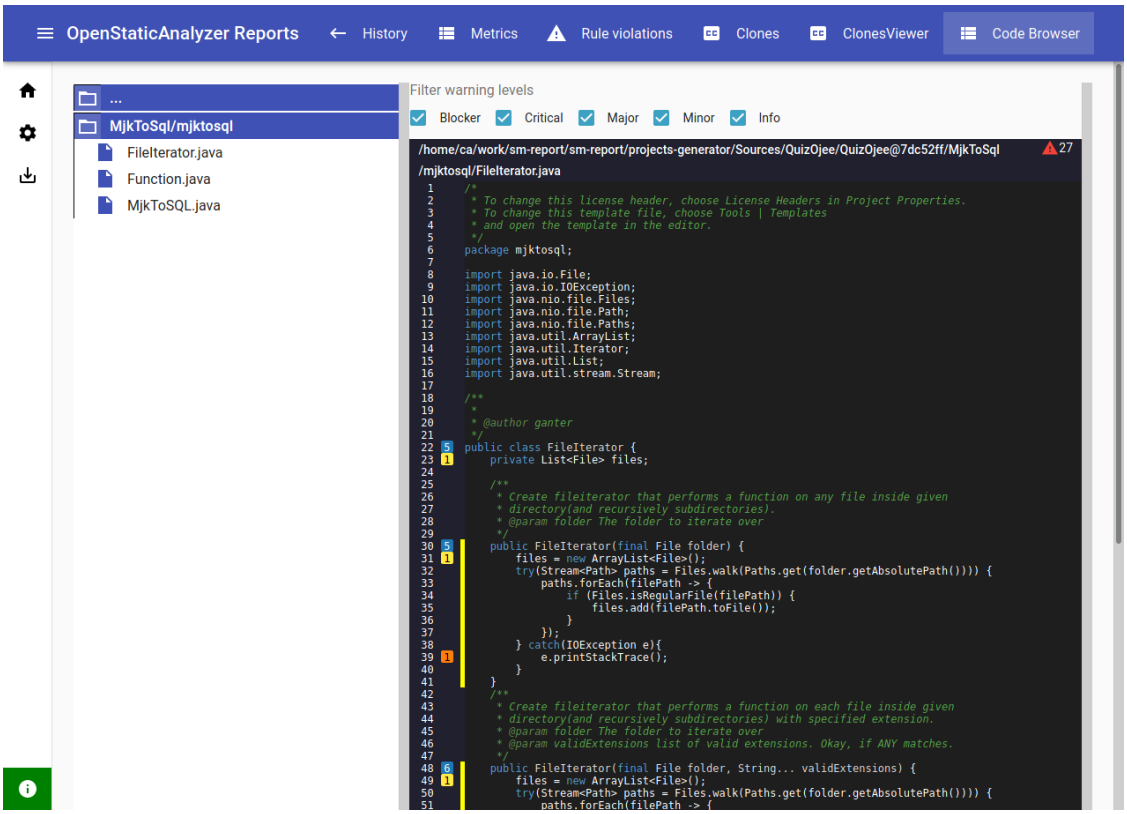

Figure 5.4: The code browser with an opened source file

side there is a file browser you can browse through the project source files and select one of them and the right side displays the source-code of the currently selected source file. The source-code browser supports proper syntax highlighting. Additionally the warnings and coding rule violations found by OpenStaticAnalyzer during the analysis are placed in small rectangular boxes next to the line number. The number in the box counts the number of warnings that have been found for that line. There are 5 levels of severity: Info, Minor, Major, Blocker, Critical - from the least to the most severe in order. Each level has its own color, which can be configured in the config file for the client. The color of the boxes is always the color of the most severe violation on that particular line. The boxes can be clicked on, to toggle display of the violation data (see on Figure 5.5). Additional information about the violation type can be accessed by clicking on the "?" icon next to the violation name in the colored box. Right above the source code, there are five tick-boxes, where the user can specify what kind of violations should be displayed and which should be hidden. It is for example possible to hide the less relevant violations of the

Figure 5.5: In this screenshot the violation boxes for two violations are displayed. The first one has a severity of “major” the second one is only of “minor” severity.

lowest “info” severity and only focus on the more important, more severe violations. In some places there are yellow stripes on the margin, these striped lines have been identified as duplicated lines of code by the OpenStaticAnalyzer. Clicking on one of these yellow stripes will navigate to the Clone Viewer page, automatically selecting the clone instance that the that line belongs to.

5.4 Clone Viewer

On the Clone Viewer page, seen in Figure 5.6 everything about clones is displayed. At first the clone classes can be selected by using the clone class dropdown menu, after selecting it, the desired clone instances can be choosen, two at once, for comparison. Most clone classes only have two instances, so once the class has been choosen, its first two instances are selected automatically. Relevant metrics for both the clone class and the clone instance are displayed, as well as the relevant snippets of source code that are duplicated. The differences in between the two instances of code are highlighted like a diff.

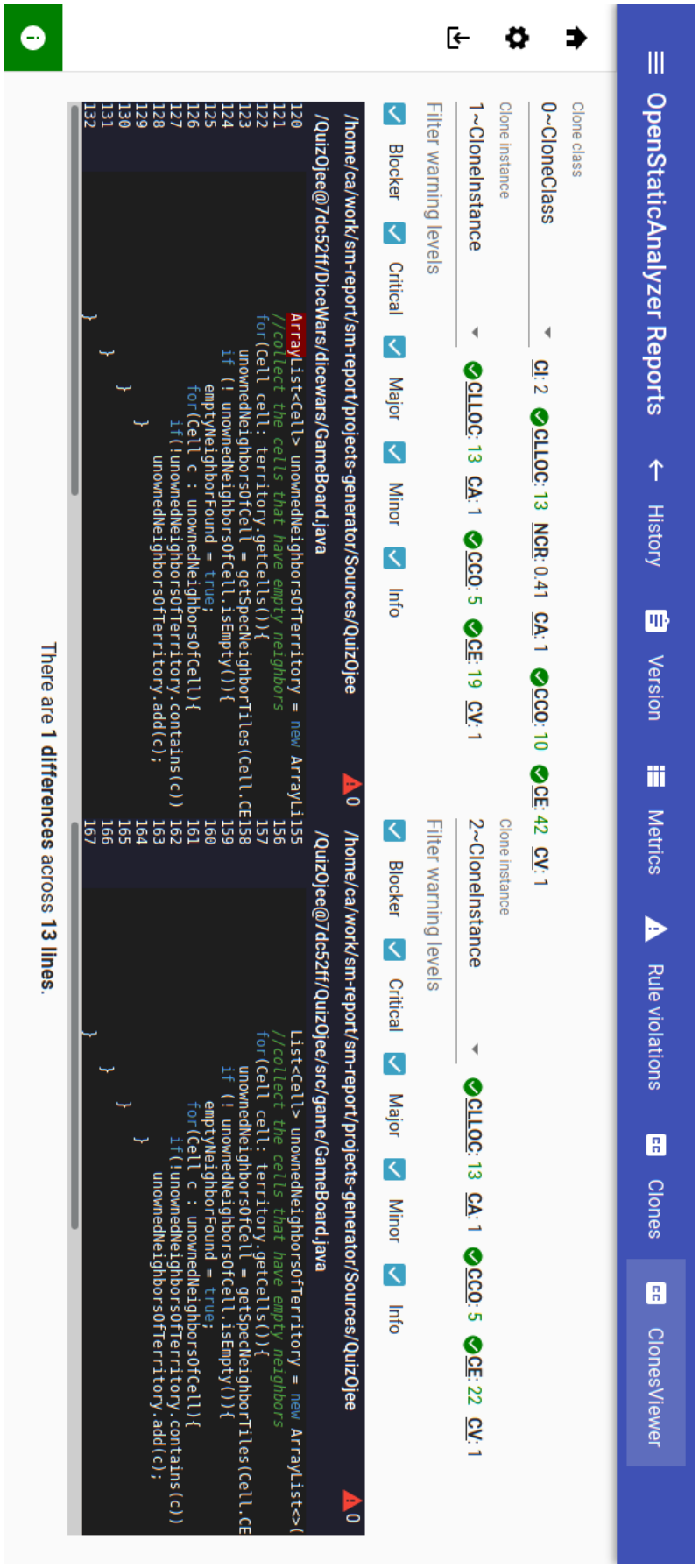

Figure 5.6: The Clone Viewer page.

5.5 Quality analysis

I talked a lot about how important it is to analyze the quality of a software, therefore I will analyze and evaluate the software of this project objectively. The analysis was performed using SonarQube and the sonar-scanner. I would have liked to use OpenStaticAnalyzer,however unfortunately does not support TypeScript, so I chose another tool to do the job.

5.5.1 Logical Lines Of Code

According to SonarQube the software is composed of 43K lines of code, however out of these there are 26K lines of html code, which actually is not even real code as it is just the UsersGuide files generated from markdown. Substracting the 26K fake lines of code leaves us with:

- 17K logical lines of **TypeScript** code
- 114 logical lines of **Python** code
- and 99 logical lines of **JavaScript** code

There is a total of 177 source files, which results in an average of 97 lines of code per file. That is considered to be a lot and files should be broken down into smaller parts.

5.5.2 Duplicated lines

SonarQube found 3287 duplicated code lines, so only around 2% of code is duplicated. Which is considered to be very good.

5.5.3 Cyclomatic Complexity

This project has a summed up Cyclomatic Complexity of 2165. The cyclomatically most complex file is the ProjectManager on the server with a complexity score of 156. This indicates, that it might make sense to split that file into multiple smaller modules, which is entirely feasible since it is just a collection of methods for processing data. There is a total of 177 source files, which results in an average complexity of 12.2, which is just slightly above the recommended value of 10.

5.5.4 Other metrics

SonarQube gives the code an “A” on Security and Maintainability, however only a “D” on Reliability. D means there is room for improvementand there is definitely some smaller cleanup jobs still to be done. The software is currently being released, however after the release process has finished, this will improve to an “A”. SonarQube itself estimates the Remediaton Effort to only 5 hours.

Chapter 6

Comparable Solutions and Conclusion

6.1 Comparing SM-Riport with other alternatives

In chapter 2 the “SourceMeter plugin for SonarQube” and the project of Deniel Dzadik Zozémusz was introduced. In this section I will compare SM-Riport with those solutions.

6.1.1 Functionality

Table 6.1 shows a comparison of the key features of SourceMeter. From the functionality point of view Dzadik Deniels solution gets the last place. It barely does more than just displaying the results as an excel sheet would do. The SonarQube plugin has more features overall, because it inherits a lot from the SonarQube platform, however when focusing only on the displed data that OpenStaticAnalyzer prduces, SM-Riport performs actually slightly better than the SonarQube plugin, as it is respecting more features.

6.1.2 Performance

SonarQube is a highly tested and optimized system and therefore it has very good performance. Due to the amount of all the features we implemented, SM-Riport is a bit on the slower side, sometimes it takes a few seconds to load larger projects or to display the source-code of larger files. SonarQube is very responsive compared to SM-Riport. Dzadik’s solution is also lightning fast, probably due to the fact that it is a very lightweight application with just a few and not very complex features.

Features	SonarQube plugin	Dzadik-solution	SM-Riport
metric support	yes	yes	yes
code duplication support	partial	yes	yes
violation support	yes	yes	yes
project history support	yes	no	yes
multiple projects	yes	no	yes
display source code	yes	no	yes
diff clone instances	yes	no	yes
display violations in the code	yes	no	yes
mark duplicated lines of code	no	no	yes
works on any OS	yes	yes	yes
providing feedback	yes	no	partial
scalable	no	yes	yes

Table 6.1: Comparison of various features

6.1.3 Setup and configuration

Setting up SonarQube used to be more of a difficult task, taking about half a day to get everything right. It has come a long way in the last 2 years. Since it is possible to launch a SonarQube server with docker using the command `sudo docker run -d --name sonarqube -p 9000:9000 sonarqube:8.3-community`, this reduces the setup time and effort to installing docker and after that the SonarQube platform can be used. But the SourceMeter plugin still has to be copied to the plugins directory of the SonarQube installation manually.

With SM-Riport and Deniel's solution the user has to install NodeJS. This is just a tiny bit simpler than installing and using docker. After that SM-Riport can be started using a script and Dzadicks solution by launching it with the `npm start` command while being in the right directory.

To sum it up, it is easy to set up any of theese solutions, SM-Riport is in my opinion slightly easyer than Deniel's solution. Setting up SonarQube needs a bit more time until the server runs and the plugin is installed.

Configuration

Deniel's solution has no configuration settings available. SonarQube can be configured using graphical means and some settings are available via config files. SM-riport can be configured mostly using config files, but also has very few configuration settings that can be changed inside the GUI.

6.2 Future plans

As always, there is still room for some improvements, let me list a few plans we have for the future:

- improve the overall performance and responsiveness of the application by location performance hotspots in the code and fixing them
- improving the performance of the input processing, with the help of the OpenStaticAnalyzer development team, by modifying the OpenStaticAnalyzer so that it also generates an additional JSON file containing only the data we need to visualize using SM-Riport. The visualized data is just a small fraction of the complete dataset generated by OpenStaticAnalyzer, therefore this is going to have a large impact on performance.
- adding new features, like adding support for metrics of all the component levels supported by OpenStaticAnalyzer (with respect to every supported language). Currently only the three most important ones are displayed: namely package, class and method metrics.
- provide a more user friendly configuration using the GUI replacing the configuration files.

6.3 Closing thoughts

I have enjoyed working on this project very much, I feel that I have improved a lot when it comes to using React, Redux, TypeScript, Git and all technologies that have been used. It has been a very useful experience to make important architectural decisions at the beginning of the project and see how they work out over the lifespan of the project. This project has been started in the summer of 2018 and has been actively developed ever since, meaning it has been developed actively for about two years by now. I have learned the importance of documenting every step of the development, not only using source-code comments, but also using Git and the features that GitLab provides for project management. I have also learned to work together with a small team of developers. I feel that

the experiences I gathered while working on this project and the knowlegde about source-code quality assurance, which I have picked up along the way, will be of big help in my future career as a Software Engineer.

Bibliography

[1] Tibor Bakota. Evaluating the effect of code duplications on software maintainability. PhD thesis, University of Szeged, Ph.D. School in Computer Science, 2012.

[2] Bence Barta and Günter Manz. Nyílt forráskódú minőségbiztosító keretrendszer api változásainak vizsgálata. TDK dolgozat, Szegedi Tudományegyetem, szoftverfejlesztés tanszék, maj 2018.

[3] CoolerVoid. Code warrior on github. https://github.com/CoolerVoid/codewarrior.

[4] Shaumik Daityari. Angular vs react vs vue: Which framework to choose in 2020. https://www.codeinwp.com/blog/angular-vs-vue-vs-react/.

[5] Vincent Driessen. A successful git branching model. https://nvie.com/posts/a-successful-git-branching-model/.

[6] Ben Frederickson. Ranking programming languages by github users. https://www.benfrederickson.com/ranking-programming-languages-by-

[7] Google. Most used programming languages. https://codeburst.io/10-top-programming-languages-in-2019-for-dev

[8] Maurice H Halstead. Elements of software science. Elsevier computer science library : operational programming systems series. North-Holland, New York, NY, 1977.

[9] SonarQube Honlapja. https://www.sonarqube.org/.

[10] J. Loeliger and M. McCullough. Version Control with Git: Powerful tools and techniques for collaborative software development. O'Reilly Media, 2012.

[11] T. J. McCabe. A complexity measure. IEEE Transactions on Software Engineering, SE-2(4):308–320, Dec 1976.

[12] Lukáš Mešťan. Git quick statistics tool. https://github.com/arzzen/git-quick-stats.

[13] Manz Günter Peter. Gui felület fejlesztése nyílt forráskódelemző platformhoz. BSc szakdolgzat, Szegedi Tudományegyetem, szoftverfejlesztés tanszék, 2018.

[14] SourceMeter plug-in for SONARQUBE platform. https://github.com/FrontEndART/SonarQube-plug-in/graphs/contributors.

[15] Kengo TODA. Findbugs. https://github.com/spotbugs/spotbugs.

[16] Hungary University of Szeged. Openstaticanalyzer source code. https://github.com/sed-inf-u-szeged/OpenStaticAnalyzer.

[17] www.breachlock.com. Top 3 open source tools for sast. https://www.breachlock.com/top-3-open-source-tools-for-sast/.

[18] Dzadik Deniel Zozémusz. Forráskód elemző szoftver kimenetének feldolgozását és vizualizálását megvalósító modern webalkalmazás. BSc szakdolgzat, Szegedi Tudományegyetem, szoftverfejlesztés tanszék, 2016.

Nyilatkozat

Alulírott Programtervező Informatikus Msc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztés Tanszékén készítettem, Msc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy diplomamunkámat a Szegedi Tudományegyetem Informatikai Intézet könyvtárában, a helyben olvasható könyvek között helyezik el.

Szeged, May 15, 2020 .....

aláírás

Acknowledgments

I would like to thank István Siket for his support, the continuous guidance regarding any issues arising during either the project or my studies and for his help in the preparation of my Master's thesis.

Furthermore, I am very grateful to Árpád Hohl, my high school math teacher, for teaching me the basics of programming in his spare time with his full heart and soul in it - accepting not a dime for his efforts - and thus helping me jump-start my IT-programming career.

This work was partially supported by grant *2018-1.2.1-NKP-2018-00004* "Security Enhancing Technologies for the IoT" funded by the Hungarian National Research, Development and Innovation Office.

Appendix A

My Contribution to the project

I have been working on and contributing to the following files.

```
.prettierignore
.prettierrc
client/.env.default
client/index.html
client/mocha-webpack.opts
client/package-lock.json
client/package.json
client/src/components/BooleanHide.tsx
client/src/components/CloneClassSelector.tsx
client/src/components/CloneInstanceSelector.tsx
client/src/components/Code/Code.tsx
client/src/components/Code/Header.tsx
client/src/components/Code/Line.tsx
client/src/components/Code/Violation.tsx
client/src/components/Code/ViolationRating.tsx
client/src/components/CustomMenuItem.tsx
client/src/components/FSNode.tsx
client/src/components/Header.tsx
client/src/components/Loading.tsx
client/src/components/Login.tsx
client/src/components/MetricsTable.tsx
client/src/components/ProjectListItem.tsx
client/src/components/ProjectStatusBar.tsx
client/src/components/TemplateReduxComponent.tsx
client/src/components/TooltipStyledOnClick.tsx
client/src/components/VirtualizedSelect.tsx
client/src/components/routes/AbsRouteComponent.tsx
client/src/components/routes/Analysis/ClonesViewer.tsx
client/src/components/routes/Analysis/Dashboard.tsx
client/src/components/routes/Analysis/Metrics.tsx
client/src/components/routes/CodeBrowser.tsx
client/src/components/routes/FileBrowser.tsx
client/src/components/routes/Metrics.tsx
client/src/components/routes/ProjectList.tsx
client/src/components/routes/UsersGuide.tsx
client/src/components/smIcons.tsx
client/src/redux/action-types.ts
client/src/redux/actions/code.ts
client/src/redux/actions/login.ts
client/src/redux/generic/Cache.ts
client/src/redux/generic/FetchHandler.ts
client/src/redux/generic/PostHandler.ts
client/src/redux/generic/defaults.ts
client/src/redux/reducers/code.ts
```

```
client/src/redux/store.ts
client/src/redux/uiData/codeBrowser/filterSettings.ts
client/src/redux/uiData/codeBrowser/index.ts
client/src/redux/uiData/index.ts
client/src/redux/uiData/login.ts
client/src/redux/uiData/projects/data/index.ts
client/src/redux/uiData/projects/data/projectData/analyses.ts
client/src/redux/uiData/projects/data/projectData/codeBrowser.ts
client/src/redux/uiData/projects/data/projectData/dashboard.ts
client/src/redux/uiData/projects/data/projectData/fileSystem.ts
client/src/redux/uiData/projects/data/projectData/index.ts
client/src/redux/uiData/projects/data/projectData/metricsPage.ts
client/src/redux/uiData/projects/data/projectData/violations.ts
client/src/redux/uiData/projects/index.ts
client/src/redux/uiData/projects/projectList.ts
client/src/redux/uiData/projects/selectedProject.ts
client/src/redux/uiData/router.ts
client/src/redux/uiData/submenu.ts
client/src/util/colors.ts
client/src/util/consts.ts
client/src/util/request.ts
client/src/util/typed-connect.ts
client/test/Cache.spec.ts
client/test/utils.spec.ts
client/tsconfig.json
client/webpack.config.js
common/consts.ts
common/index.ts
common/interfaces/codeMetrics.ts
common/interfaces/get/data/aggregate/projectName/metrics.ts
common/interfaces/get/data/aggregate/projectName/tools.ts
common/interfaces/get/data/path_overwrite_get.ts
common/interfaces/get/data/projects/index.ts
common/interfaces/get/data/projects/metrics/index.ts
common/interfaces/get/data/projects/projectName/lang/ID/code_duplications.ts
common/interfaces/get/data/projects/projectName/lang/ID/coding_rule_violations.ts
common/interfaces/get/data/projects/projectName/lang/ID/dashboard.ts
common/interfaces/get/data/projects/projectName/lang/ID/filesystem.ts
common/interfaces/get/data/projects/projectName/lang/ID/metrics.ts
common/interfaces/post/data/aggregate/projectName/index.ts
common/interfaces/post/data/delete_cache.ts
common/interfaces/post/data/file/index.ts
common/interfaces/post/data/path_overwrite_save.ts
common/interfaces/post/metricHelp.ts
common/interfaces/post/review/new.ts
common/interfaces/post/review/save.ts
common/libs/graph/columbus/io.ts
common/libs/graph/columbus/strtable.ts
common/libs/graph/graphlib/attribute.ts
common/libs/graph/graphlib/attribute_composite.ts
common/libs/graph/graphlib/attribute_float.ts
common/libs/graph/graphlib/attribute_int.ts
common/libs/graph/graphlib/attribute_owner.ts
common/libs/graph/graphlib/attribute_string.ts
common/libs/graph/graphlib/edge.ts
common/libs/graph/graphlib/edge_type.ts
common/libs/graph/graphlib/graph.ts
common/libs/graph/graphlib/node.ts
common/libs/graph/graphlib/node_type.ts
common/libs/graph/graphsupportlib/metric.ts
common/model/Analysis.ts
common/model/CloneClass.ts
common/model/CloneInstance.ts
common/model/FileSystem.ts
common/model/Metric.ts
common/model/Project.ts
common/model/Review.ts
```

```
common/model/SourceCodeElement.ts
common/model/Stats.ts
common/model/ToolFunctionality.ts
common/model/Violation.ts
common/model/Warning.ts
common/model/config/serverConfig.ts
common/package-lock.json
common/package.json
common/tsconfig.json
common/util.ts
package-lock.json
package.json
projects-generator/.gitignore
projects-generator/README.md
projects-generator/config.ts
projects-generator/consts.ts
projects-generator/helpers/analysisLogHelper.ts
projects-generator/index.ts
projects-generator/package-lock.json
projects-generator/package.json
projects-generator/projects/Agenda/process.ts
projects-generator/projects/Appengine-Mapreduce/process.ts
projects-generator/projects/Atom/process.ts
projects-generator/projects/Ceilometer/filter.txt
projects-generator/projects/Ceilometer/process.ts
projects-generator/projects/JFreeChart/process.ts
projects-generator/projects/Log4CPP/process.ts
projects-generator/projects/Log4J/SoftFilter.txt
projects-generator/projects/Log4J/fbfilelist.txt
projects-generator/projects/Log4J/process.ts
projects-generator/projects/QuizOjee/SoftFilter.txt
projects-generator/projects/QuizOjee/process.ts
projects-generator/util.ts
scripts/prepare-commit-message.sh
server/.env.default
server/package-lock.json
server/package.json
server/resources/sm-riport.sh
server/src/components/UGManager.ts
server/src/components/handlers/aggregators/aggregateFileTree.ts
server/src/components/handlers/codeDuplications.ts
server/src/components/handlers/fileTree.ts
server/src/components/handlers/metrics.ts
server/src/components/logger.ts
server/static/.gitkeep
server/tsconfig.json
server/webpack.config.js
```

B. függelék

Magyar összefoglalás - Summary in Hungarian

B.1. Témakiírás

A feladat az OpenStaticAnalyzer parancssori statikus forráskód-elemző grafikus felületének létrehozása. A grafikus felhasználói felület megjeleníti könnyen értelmezhető és informatív módon az elemző által generált adatokat, beleértve a különféle metrika értékeket és szabálysértéseket, valamint a kódmásolatokat. A projekt legyen jól karbantartható és új funkciókkal könnyen bővíthető. A megoldás működjön ugyanúgy bármelyik széles körben elterjedt desktop operációs rendszeren (Windows, Linux, IOS), valamint legyen könnyen telepíthető és felhasználóbarát.

A feladat az OpenStaticAnalyzer statikus forráskódelemző grafikus felhasználói felületének kialakítása és fejlesztése. Az OpenStaticAnalyzer elemző megismerése annak kimeneti adatainak feldolgozása érdekében. Kapcsolódó irodalom kutatása és szoftverek keresése, amelyek hasonló feladatokat látnak el. A megszerzett ismeretek alapján kidolgozni és implementálni egy grafikus felületet, amely képes az OpenStaticAnalyzer által kiszámított metrikák és kódmásolások transzparens megjelenítésére. Az alkalmazás fejlesztéséhez megfelelő programozási nyelv és ökoszisztéma kiválasztása. A szoftver architektúrájának megtervezése, fejlesztése és kiadása.

B.2. Bevezetés és motiváció

Olyan korban élünk, amelyben az élet minden területén találkozunk különböző szoftverekkel, ezek közül néhányan kritikus rendszereket is üzemeltetnek, pl. banki vagy biztonsági rendszerek esetében. Minden szoftverre jellemző, hogy azt várjuk el tőle, hogy használható legyen az adott feladatra, legyen naprakész, és soha ne történjenek benne hibák. Ahhoz hogy ezt elérjük, fontos a szoftver minőségének folyamatos monitorozása, és javítása. Ezt különböző eszközökkel érhetjük el, első sorban statikus forráskód elemzők segítségével, mint pl. a SonarQube [9], OpenStaticAnalyzer [16], FindBugs, CodeWarrior [3].

B.2.1. Hogyan lehet egy szoftver minőségét számszerűsíteni?

A szoftver minőségét különböző metrikákkal jellemezhetjük. Ezek olyan számszerű értékek, amelyeket a forráskód alapján lehet meghatározni, az egyik legegyszerűbb pl.: a LOC vagyis a „Lines Of Code”, melynek a kiszámítása egyszerűen úgy történik, hogy megszámoljuk összesen hány sornyi forráskódunk van. A metrikákat csoportosítani is lehet: megkülönböztethetünk méret-, dokumentáció-, komplexitás- kohéziós-, kapcsolódottság- és öröklődéses metrikákat. A forráskód alapján ezen felül azonosíthatunk „rossz szokásokat”, vagy „szabálysértéseket”, melyek helytelen működéshez vezethetnek a végtermékben. Valamint kereshetünk duplikált kódrészleteket is, melyek létezése szintén negatívan befolyásolja a kód minőségét [1]. Arra a kérdésre, hogy miért éri meg egy cégnek az általa fejlesztett termékek forráskódjának a minőség-biztosítása, azt a választ találtam, hogy úgy kell tekinteni a forráskódra, mint egy befektetésre. Ha jó a forráskód, akkor azt a jövőben újra fel lehet használni, és ezért sokszorosan meghozza az árát. Dolgozatomban bemutatásra kerülnek különböző forráskód elemzők. Szerepel a SonarQube, amely egy teljes rendszer, szinte bármely közismert programozási nyelven íródott kódot képes elemezni, vagy natívan, vagy valamilyen plugin segítségével, és a kinyert adatokat magas szinten, felületesen szolgáltatja. A Code Warrior [3] - mely szintén egy többnyelvű elemző - arról híres, hogy kevés fals pozitív szabály-sértést talál a kódban [17]. A SpotBugs, mely a FindBugs utódja [15], és kifejezetten Java nyelvű projektek elemzésére alkalmas, az elterjedt java fejlesztői környezetekhez - Eclipse, Netbeans, IntelliJ IDEA - beépülő modulokkal rendelkezik. És végül bemutatásra kerül a Szegedi Tudomány Egyetem ál-

tal fejlesztett OpenStaticAnalyzer, amely különösen fontos jelen dolgozat szempontjából, hiszen ehhez készült grafikus megjelenítő a projekt során. Az OpenStaticAnalyzer különböző metrikákat képes kiszámolni, nem csak fájl, hanem a forráskód szemantikailag értelmezhető valamennyi komponens szintjén, valamint képes szabály-sértések és kódmásolatok beazonosítására is. Kimenetét szöveges és bináris fájl formájában állítja elő.

B.3. Kapcsolódó munkák és előzmények

A 2. fejezetben kettő kapcsolódó projektmunkát mutatok be: a „SourceMeter plugin for SonarQube”-t, és a Dazdik Deniel Zozémus szakdolgozatának alapját képző megjelenítőt, mely a SourceMeterhez készült. A SourceMeter nagyon hasonló az OpenStaticAnalyzerhez, a két elemző alapjaiban megegyezik és kimeneti formátuma azonos. Fontos különbség, hogy az OpenStaticAnalyzer teljesen szabadon felhasználható, nyílt forráskódú ingyenes szoftver, míg a SourceMeter fizetett megoldás, mely extra funkciókkal egészíti ki az OpenStaticAnalyzer-t. Ezért hozható kapcsolatba jelen feladattal ez említett két projekt. A SonarQube plugin olyan módon nyújt megoldást a SourceMeter kimenetének megjelenítésére, hogy integrálja a SourceMeter elemzési eredményeit a SonarQube platformba, így a SourceMeter elemzési eredményei a SonarQube platform webes felületén válnak megtekinthetővé. A plugin megjelenítésért felelős részt jómagam fejlesztettem [14]. Sajnos a SonarQube pluginAPI-jának folyamatos és nehezen követhető változásai ellehetetlenítik ennek a projektnek a továbbfejlesztését [2]. A Dzadik Deniel által fejlesztett megjelenítő egyszerű és gyors, viszont csak kis mértékben rendelkezik több funkcióval, mintha egy excel táblázatban tekintenénk meg az elemzési eredményeket. A kitázött cél, hogy e két megoldásnál az általunk fejlesztett SM-Riport nevű alkalmazás jobban sikerüljön.

B.4. Tervezés

A tervezés során szinte teljesen szabad kezet kaptam, egészen a programozási nyelvtől kezdve. Több nyelv is szóba jött, mint a Java és a JavaScript, végül a döntés a TypeScriptre esett, ami tulajdonképpen rendelkezik mindkettő nyelv sajátosságaival, a JavaScript kiterjesztése, a benne írt kód nagyon hasonlít Java kódra, viszont JavaScriptre fordul. Vé-

leményem szerint előnye a JavaScript felett hogy statikus típusokat képes kezelni. Architekturálisan egy a SonarQube-hoz hasonló szerver-kliens alkalmazásban gondolkodtam, ugyanis ez a legrugalmasabb megoldás. A NodeJS szerver fogja feldolgozni az input fájlokat, neki lesz hozzáférése a merevlemezhez, a kliens pedig egy böngészőben futtatott web-applikáció, mely a szervertől kapott adatokat megjeleníti. A front end oldalán valamilyen modern, komponens alapú keretrendszert szerettem volna alkalmazni, így az Angular, a React, és a Vue keretrendszerek jöttek szóba, melyek közül a legnépszerűbb, és könnyen megtanulható React lett a nyertes. A React mellett szükség volt egy állapotkezelő megoldásra, is, amelynek segítségével el lehet menteni az állapot-adatokat. Erre a célra a Redux kínálta magát, mely arról híres hogy nagyon jól használható React-al együtt.

Manapság nem készül szoftver verzió-kezelő használata nélkül, én is alkalmaztam a jelenkor legnépszerűbb verzió kezelő programját a Git-et. Ez az eszköz használható jól és jobban (pl. rossz az, amikor egy branchen dolgoznak sokan), annak érdekében hogy helyesen használjuk, a Git-Flow [5] „branching model” alkalmazása mellett döntöttem. Ez a modell különböző osztályokba sorolja a brancheket (master, release, develop, feature, hotfix és bugfix). A modell helyes kivitelezésével lehetségessé válik több funkció párhuzamos fejlesztése, úgy hogy a változtatások összefésülésénél ne jelentkezzenek különösebb problémák. Emellett lehetőség adódik a fejlesztésnek bármely pontján elindítani a release folyamatát, szintén az aktuális fejlesztésekkel párhuzamosan, ezzel biztosítva a folyamatos releaselés lehetőségét. A projekt menedzselése GitLabon keresztül történt.

A GitLab egy központi Git repozitory-t biztosít, és ezt a funkcionalitását kiegészíti projekt menedzselő funkciókkal. A fejlesztési döntéseket és lépéseket GitLab-on belül issuekkal dokumentáltuk. Minden feladathoz/problémához készült egy issue, melyben pontosan le volt írva minden tudnivaló azzal kapcsolatban, és különböző tagek jelölték az issuek állapotát a „todo”-tól a „closed” állapotig.

A forráskód minőségének része, hogy a kód teljes része konzisztensen, ugyanolyan stílusban legyen megírva. Ennek garantálására a Prettier automatikus forráskód formázót vetettem be, olyan módon hogy a git verzió-kezelő rendszer minden egyes kommit előtt lefuttatta ezt a formázót, és a kommitot csak abban az esetben hagyta jóvá, ha a kommittálásra kerülő kód a Prettiernek megfelelően van formázva. Ezt a fajta funcionalitást a git-hooks nevű git feature kihasználásával értem el.

A projekt kiadása és fejlesztése során is WebPack csomagoló vezényelte a fordítást. Ez a csomagoló („bundler”) program a megfelelő konfiguráció mellett képes lefordítani a TypeScript fáljokat JavaScript fájlokká, azokat összelinkelni egyetlen fájlba, majd tömöríteni azt. Így a kész termékben a sok forrásfájl helyett már csak egyetlen minimális méretű fájllal kell foglalkoznia a felhasználónak.

A 3.2. ábrán látható a SourceMeter/OpenStaticAnalyzer által előállított output fájlszerkezete. Ilyen alakú könyvtárakat kap inputnak a készítendő program, minden elemzett projektenként egyet, ezeket kell tudni feldolgozni. Az input szerkezetéből kinyerhetőek a projektek nevei, hogy milyen nyelven készültek a projektek, hány elemzés készült és mikor, illetve elemzésenként a `projektnev.graph` bináris fájl tartalmából kiolvasható minden elemzési adat.

B.5. Megvalósítás

A 4. fejezetben a megvalósított forráskódot mutatom be, annak is csak azon részeit, melyeket aktívan saját magam fejlesztettem. A fejlesztési munkák közül felvállaltam a tervezés feladatát, meghoztam a fontosabb megvalósításbeli döntéseket és a projekt szerkezeti felépítését is én alakítottam ki. Az A melléklet azon fájlok neveit tartalmazza, amelyek fejlesztése hozzám tartozik, illetve a 4.1 ábrán látható az egyes fejlesztők által a teljes projekt eddigi futása során hozzáadott/törölt/módosított sorok száma. Látható, hogy a legtöbb fejlesztést én végeztem a projekten.

A 4.2 ábra ismerteti a projekt fájlszerkezeti felépítését, és rövid leírást nyújt a mappák tartalmáról is. A projekt 3 jól elkülöníthető részre bontható: a szerver a kliens és a „közös”, a common projekt, amely a szerver és kliens által is használt adatszerkezeteket tartalmazza. Létezik még egy negyedik alprojekt is, a projects-generator, mely a tesztelést segíti elő azzal, hogy segítségével operációs rendszertől függetlenül, minden fejlesztőnél elő lehet állítani egy tartalmilag ekvivalens elemzésekből álló inputot a megjelenítőnk számára.

Bemutatásra kerül, hogy a kliens milyen módon kéri le az adatokat a szervertől. Itt fontos szempont volt, hogy az adatoknak állítsuk vissza a TypeScript-es típusait, mert ezek egyébként elvesznek a csatornán, amelyen a szerver-kliens kommunikáció zajlik. Ezeknek a típusoknak a kényelmes helyreállítását szolgálják a FetchHandler és a Post-

Handler osztályok, melyek használatát a 4.3.1 fejezetben mutatom be kód-részleteken keresztül.

A Reactnak köszönhetően az egész applikáció egyetlen HTML fájlból áll csupán, mégis annak látszatát kelti, hogy több weboldalból áll az applikáció. Ezt a funkcionalitást a react-router csomag segítségével értem el. Ennek a konfigurálása található az `App.tsx` fájlban, azzal az apró trükkel kombinálva, hogy felüldefiniáltam a standard `Route` komponenst egy sajáttal, ami kimenti, hogy az aktuális URL alapján éppen melyik projekt van megnyitva. Ennek a projektnek a nevét kimenti a Redux által menedzselt store-ba, ahonnan bármelyik komponens eléri. Ez az információ szinte minden API híváshoz szükséges, ezért nagyon praktikus hogy a felüldefiniált komponens a háttérben folyamatosan naprakészen tartja. Ezután bemutatom a `CodeBrowser` komponens funkcionalitását, az interfészét és az implementáció különleges részeit. Ezt követi a `CloneBrowser` és a `MetricTable` komponens belső működésének rövid leírása, és az általam különlegesnek tartott részek kiemelése.

B.6. A megvalósított applikáció bemutatása

Az 5. fejezetben bemutatom az elkészült applikáció fontosabb részeit, amelyeket én fejlesztettem. A leírtakat képernyőképekkel egészítettem ki. Az 5.1 képen a kezdőoldalt láthatjuk, melyen fel vannak sorolva az elérhető projektek néhány metrikával együtt. Az 5.2 ábra a „Projekt History”, vagyis a projekt fontosabb általános információit ábrázolja. Különböző metrikák, a szabálysértések mennyisége kétféle diagrammon, illetve ezek változása az idővel, valamint néhány metrikának a változása az idővel. Látható még a forráskód összetételében a kódsorok, a dokumentációs sorok és a kódmásolatok aránya. Az 5.3 képernyőképen látható a Metrikák megjelenítése. Itt három különböző komponens szint („packages”, „classes” vagy „methods”) közül választhatunk, hogy a táblázatban melyek metrikáit szeretnénk látni. A táblázat oszlopai rendezhetőek, tartalma szűrhető. Az egeret az oszlop fejlécre véve megjelenik egy rövid leírás az adott oszlopban található metrikáról. A komponens nevére kattintva elnavigál az oldal a CodeBrowserbe, és megmutatja a komponens forráskódját. A CodeBrowser-t láthatjuk az 5.4 képen. A projekt forrásfájlai megjeleníthetők, a forráskód margóján kis négyzetalakú dobozok jelzik a forráskód adott sorában hány, és milyen súlyos (ezt a szín jelzi) szabálysértést talált

az elemzőnk. Ha ezekre a jelölőkre kattintunk lenyílnak a szabálysértések pontos leírásai is, mint ahogy azt az 5.5 képen láthatjuk. Az 5.6 képernyőképen pedig a CloneBrowser-t láthatjuk, amely egymás mellett megjeleníti a kódmásolatok forráskódját, kiemelve a kettő közti különbségeket. A megjelenített kódmásolatokat a 3 lenyíló menü segítségével állíthatjuk be. Valamint a CloneClass és CloneInstance metrikák is megtekinthetőek itt.

B.7. A megvalósított applikáció minőségelemzése

Az 5.5. fejezetben elemzem az elkészült applikáció minőségét a SonarQube segítségével. Egy összessen több mint 17.000 sornyi végrehajtható forráskódból álló program készült el, melynek alig 2%-a tartalmaz kódduplikációt, ami nagyon jó. Komplexitását tekintve fájlonkénti átlagban kicsivel $(2,2$-vel) a McCabe által javasolt 10 felett van, ami azt jelenti hogy kis mértékben komplexebb lett az ajánlottnál. Karbantarthatóságból teljesen megfelelő, valamint biztonsági szempontoknak is teljesen megfelelő „A" osztályozást adott a SonarQube. A megbízhatóság terén lehet még javítani, „D" osztályozást ért el a forráskód, de ezzel tisztában van a csapat, és ezek a hibákat a release folyamat végére kijavításra kerülnek.

B.8. Konklúzió

Ebben a fejezetben (6. fejezet) összehasonlításra kerül az elkészült SM-Riport termék a másik kettő alternatívával, a Dzadik Deniel Zozémusz féle megoldással, illetve a „SourceMeter plugin for SonarQube platform"-mal. Funkcionalitás terén, ahogyan azt a 6.1 ábra szemlélteti, a Dzadik-féle megoldás messze hátramarad. Az OpenStaticAnalyzer kapcsán Sm-Riport kis mértékben több feature-t támogat, mint a SonarQube-plugin. Ez véleményem szerint sikeres eredmény. A telepítés és első futtatás is az SM-Riportnál a legegyszerűbb, szinte ugyanilyen könnyű a Dzadik-féle megoldást először futtatni. A SonarQube plugin esetében az bonyolítja meg a telepítést, hogy nem csak a SonarQube platformot, hanem a sonar-scannert, illetve a SourceMeter plugin fájlait is telepíteni kell egymás után, így a másik kettőhöz képest ezt a legnehezebb feltenni. A konfiguráció terén a SonarQube vezet, hiszen rengeteg mindent lehet grafikus környezetben állítani, nem csak konfigurációs fájlok átírásával konfigurálható. Az SM-Riport főleg a konfigurációs

fájlokra támaszkodik, de néhány beállítás elvégezhető a grafikus felületen is.

B.8.1. Jövőbeli tervek

Mint minden projektnél, itt is fennáll a projekt továbbfejlesztésének lehetősége, hadd soroljak fel néhány a projekttel kapcsolatos jövőbeli tervet:

- Javítani szeretnénk az általános hatékonyságát és responzibilitását az alkalmazásnak olyan módon, hogy beazonosítjuk a forráskódban az u.n. „hotspot" területeket. Ezek azok a kis részek amelyek nagy mértékben befolyásolják a teljesítményt, ezeken szeretnénk optimalizálást végrahajtani.

- Az input feldolgozásának hatékonyságát is javítani szeretnénk. Ennek érdekében az OpenStaticAnalyzert fejlesztő csapat segítségével új JSON formátummal szeretnénk kiegészíteni az OpenStaticAnalyzer kimenetét. Ebben az új kimeneti fájlban csak az SM-Riport számára lényeges elemzési adatok lesznek tárolva, így töredékére csökkenhető az adat-importálásra és feldolgozásra elhasznált idő, és jelentősen gyorsabban fognak az adatok a kliensoldalra elérkezni.

- Új, apróbb jellegű kiegészítő tulajdonságokkal szeretnénk bővíteni az eszközt, pl. megjeleníteni az OpenStaticAnalyzer által kiértékelt minden komponens metrikáit, ugyanis jelenleg csak a három legfontosabb jelenik meg - a csomag, osztály és metódus szintű komponensek metrikái.

- felhasználó barátabb módon szeretnénk elérhetővé tenni a fontosabb configurációs beállításokat, a konfigurációs fájlok helyett inkább a könnyebben kezelhető grafikus felületen keresztül.

B.8.2. Személyes tapasztalatok

Nagyon örülök annak, hogy két évvel ezelőtt ezt a projektfeladatot megkaptam és fontos architekturális döntéseket hozhattam, majd követhettem, hogyan befolyásolják ezek a projekt fejlődését. Megtanultam a fejlesztés minden lépésének dokumentálásának fontosságát. Megtanultam együtt dolgozni egy kis fejlesztői csapattal is. A projekt során nagyon sok hasznos tapasztalatra tettem szert, sőt forráskód minőségbiztosításról szóló ismereteket is szereztem, amelyek megalapozzák szoftverfejlesztői karrieremet.