Türk Mühendislik Araştırma ve Eğitimi Dergisi
Turkish Journal of Engineering Research and Education

1(1), 1-7, (2022)

RESEARCH ARTICLE

Code Quality Analysis Engine with Codes Written in Typescript

TypeScript ile Yazılmış Kodlar için Kalite Analiz Motoru

Mehmet Göktürk*, Fatih Koç
Gebze Technical University, Faculty of Engineering, Mechanical Engineering, Kocaeli

Geliş / Received: 13.04.2022
Kabul / Accepted: 11.05.2022
***Corresponding Author:** **Mehmet Göktürk,** gokturk@gtu.edu.tr

ABSTRACT: In this study, an automatic software based on basic known software quality parameters, which can apply static code analysis on code files written with the new generation TypeScript programming language, which is made available by Google instead of Javascript, measure compliance with the defined standards and as a result calculate a quality index value for the code file. quality analysis engine has been developed.
The software developed as a web application, where users can perform the analysis process through the application interface by making the necessary configuration changes, access the analysis results, the calculated quality index value and the list of situations that cause this value to decrease, has been arranged.
In the study, it is aimed to help software developers using TypeScript language to work in accordance with quality code development standards and to enable these standards to be followed more easily by embodying them with the help of a measurable metric. For this purpose, a web-based software was developed using node.js and Angular libraries, rules were defined for the TypeScript language in this software, and it was possible to calculate a quality index value according to the results obtained by running these rules on the analyzed file with the help of the configuration determined by the users.
As a result of the study, different scenarios of the rules were created and the TypeScript files of these scenarios were tested with the developed web application and the performance criteria were evaluated. A success of 62% was achieved in the first results obtained, and the developers evaluated that the system could be used.

Keywords: Code Quality Analysis, Typescript, Angular, Software Development, Software Quality

ÖZ: Bu çalışmada Google tarafından Javascript yerine kullanıma sunulan yeni nesil TypeScript programlama dili ile yazılmış kod dosyaları üzerinde statik kod analizi uygulayabilen, tanımlanmış olan standartlara uygunluğun ölçülmesi ve bunun sonucunda kod dosyası için bir kalite indeks değeri hesaplayan, temel bilinen yazılım kalite parametrelerini temel alan otomatik bir yazılım kalite analiz motoru geliştirilmiştir.
Kullanıcıların gerekli konfigürasyon değişikliklerini yaparak uygulama arayüzü üzerinden analiz işlemini gerçekleştirebildiği, analiz sonuçlarına, hesaplanan kalite indeks değerine ve bu değerin düşmesine yol açan durumların listesine erişebildiği bir web uygulaması olarak geliştirilen yazılım düzenlenmiştir.
Çalışmada TypeScript dilini kullanan yazılım geliştiricilerinin kaliteli kod geliştirme standartlarına uygun şekilde çalışmalarına yardımcı olmak ve bu standartların ölçülebilir bir metrik yardımı ile somutlaştırılarak daha kolay şekilde takip edilmesini sağlamak amaçlanmıştır. Bu amaç doğrultusunda node.js ve Angular kütüphaneleri kullanılarak hazırlanan bir web tabanlı yazılım geliştirilmiş, bu yazılımda TypeScript dili için kurallar tanımlanmış ve kullanıcıların belirlediği konfigürasyon yardımı ile analiz edilen dosya üzerinde bu kurallar çalıştırılarak elde edilen sonuçlara göre bir kalite indeks değeri hesaplanması mümkün olmuştur.
Çalışma sonucunda gerçekleştirilen kurallara ait farklı senaryolar oluşturulmuş ve bu senaryolara ait TypeScript dosyaları geliştirilen web uygulaması ile test edilerek başarım kriterleri değerlendirilmiştir. Elde edilen ilk sonuçlarda %62 düzeyinde bir başarı elde edilmiş, sistemin kullanılabilir olduğu geliştiriciler tarafından değerlendirilmiştir.

Anahtar Kelimeler: Kod Kalite Analizi, Typescript, Angular, Yazılım Geliştirme, Yazılım Kalitesi

1. INTRODUCTION

The Angular software development framework system, popularized by Google, is an open source project that can be freely used and modified by anyone [1]. A more qualified language than Javascript called TypeScript is used in this popular framework, which is mostly created to develop web applications. In this structure, frontend developers work using frameworks such as Angular or React to present and manipulate data efficiently.

The Typescript programming language offers a work environment that can be easily managed by large teams. Compared to the Javascript language used with AngularJS, which was used in previous software development framework, significant advanced features have been provided to the developers.

However, automatic examination of the developed code is one of the important mechanisms that can be put forward in the process of quality software development, in order to ensure that software developers, -who are observed to be able to maintain old and bad habits and continue to make various basic principle mistakes-, write quality code.

In this study, a static code analysis was applied on the code files written in TypeScript, which is the programming language of the Angular framework structure, to measure the compliance with the predefined standards and quality guidelines, and as a result, a system was developed that calculates a quality index value for the code file entered into the system.

Within the software development process, the complexity of software projects increases as the development process progresses, personnel changes in software teams, the growth of project scales and the growth in the source code base make it difficult to maintain the code quality at acceptable levels, and the cost of software development and maintenance gradually increases due to the decrease in quality [2].

With the system developed within the scope of the study, it is aimed to help software developers using TypeScript language to write code in accordance with quality code development standards and to follow these standards more easily by embodying them with the help of measurable metrics. For this purpose, a web-based static code analysis software was developed using node.js and Angular libraries.

In the developed software, the number of rules for the TypeScript language were defined and the configuration parameters determined by the users were created. With the help of this system, a final quality index value was calculated according to the results obtained by running the rules entered in the system on the analyzed file. Furthermore new rules can be added in a flexible structure with a parametric approach.

In the following sections of the article, there are literature review on the subject, the method and material chosen for problem solving, the technologies and tools used, the findings, conclusion and discussion, and references sections.

2. LITERATURE REVIEW

Traditional code analysis and problem finding has been a popular subject for software engineering discipline. Several tools were emerged for thei purpose. Eslint for example statically analyzes written code to quickly find problems. It is built into most text editors and you can run ESLint as part of the continuous integration pipeline [3]. The primary reason ESLint was created was to allow developers to create their own linting rules. ESLint was designed to have all rules completely pluggable where customization was priority. ESLint, especially working with IDE and text editors, applies static code analysis on the codes written in JavaScript language and aims to detect errors while writing the code [4]. Since TypeScript language is built on JavaScript, ESLint can also analyze TypeScript files.

SonarQube, on the other hand, provides quality and safety analysis for code written in more than 20 languages, and is added as an extra step to the software development process, helping to analyze the entire project before distribution [5,6]. SonarQube can perform analyzes with the help of an extra module specially designed for TypeScript language.

Apart from the popular applications mentioned, there are also smaller scale examples of similar

static code analysis software distributed as NPM packages.

3. METHOD AND MATERIAL

3.1 General Approach

Given the vast set of existing code quality analysis tools, this study limited its focus to static code analysis on TypeScript language only with high speed generic evaluation as target pperformance parameter.

Static code analysis approach has been followed in this study. As described by Stefanovic et. al The static code analysis process is useful both for optimizing the operation of the compiler and detecting irregularities and possible defects [7,8] which can be used to evaluate quality as well. Systems can be developed helping developers to understand the behaviour of programs and to identify various possible defects in those without their execution performed. Static code analysis is therefore performed by programs that explain the behaviour of developed code.

3.2 Design

The project is designed as a web application. It is divided into two main parts, the back-end and the front-end. The back-end will share the results in JSON format with the generated API by applying static code analysis on the given TypeScript file and checking the defined rules. The front-end part will receive the result data via this API and visualize it with a user-friendly interface. The backend part is divided into 4 main parts:

- Data Processing Module: This module is responsible from loading code file containing TypeScript program and processes using a typescript compiler.
- Rule Engine: It is the module where each rule to be checked by evaluating the data of the decoded file is applied.
- Configuration Module: It is used to create configuration web application backend and API services running on Node.js. It is the module that enables the transfer of user preferences such as file name and information about the rules to the application.
- API: It is the module that is responsible from calculating the Code Quality Index using the results of the rules and allows these results to be shared with the front-end of the system.

The system architecture showing dataflow that is composed of these modules is shown in Figure 1.

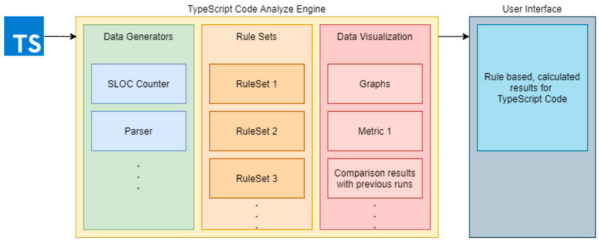

Figure 1: System architecture

The analysis process on the back-end was triggered after data was obtained from the user with the help of front-end services.

Visualization and processing of this data and making it user-friendly and intuitive are also done in front-end section. A structure suitable for MVC architecture has been utilized to develop entire system.

3.3 Technology and Tools Used

Implementation of software, that works on developer code requires combination of several technologies. Significal amount of parsing and string matching is required. In this section, information about the purposes of using each technology and tools in the design, development and testing process of the system are given.

3.3.1 Technologies

As the subject language Typescript is focus of this study, same environment is used in order to develop the system. Therefore, along with the Typescript language, Angular framework is used for the development of the front-end application. For certain sections, Javascript is also utilized for some parts of back-end and front end development phases. As a framework, Node.js has been used to establish server structure of the back-end application where Express.js is used to create web application backend and API services running on Node.js. HTML/CSS code processed and

Highcharts library was accomodated to draw graphic results on user interface. Therefore a complete multilayer web framework was established to data processing and presentation.

For the data processing module, a typescript parser (NPM Package) needed and therefore utilized. Furthermore, "ts-file-parser" module was also used to decode Typescript source code files successfully. Figure 2 shows main user interface for uploading the codefiles that is used in the study using ts-file-parser module.

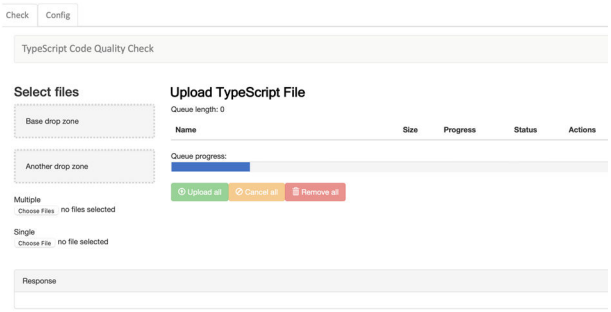

Figure 2: Code file upload interface

Usable interaction was one of the key challenges of the study since developers are usually hesitant in using such systems. In order to create user friendly front-end user interface, Angular Material Design npm package was installed along with the system, where professional look and feel can be obtained with relatively smaller amount of effort.

3.3.2 Tools

The system has been developed in Visual Studio Code development environment. In order to draw diagrams during design pahase, draw.io package was used to draw diagrams where the functionality in the main canvas can be controlled by tha main application. Draw io libraries, the menus, the toolbar, the default colours, the storage location, can all be changed by the programmer. Furthermore, tabulation and word processing tools were also used to evaluate data obtained from the system.

3.4 General Structure

Once the typescript file becomes uploaded, the back-end part of the system performs analysis of source code file using the user preferences that are stored in a 'config.json' file in directory. The file path to be analyzed, the directory path where the result files will be created, the defined rules, the categories of the rules, the weights of the rules in the code quality index and the preferences of these rules are included in this configuration file along with other necessary parameters. We have observed similar approaches were used in previous studies literature [9-11]. These quantity of these parameters can later be extended for practical purposes. The user is expected to make any changes and modification before running the backend of the application and restart the backend process after any changes are made. For the study, there are only certain rules that are tested. Since the main purpose of the study is to test viability and workability of such a technique, limited number of rules were implemented. The rules that are included in preferences file are given in Table 1. In Table 1, rule names, categories and content of the rule is summarized.

The initialization of system requires a backend server to be started manually. After the backend server is started, it will constantly listen to the specified port and respond to incoming HTTP requests for source file analysis processing task. The analysis process is then started with the request from the front end client per uploaded file. As the analysis process starts, the file to be analyzed will be read/parsed and this file will be analyzed in different ways with the help of the 'typescript-parser' and 'ts-file-parser' NPM packages used within the system. In this way, asynhronous use of parser and source code evaluator becomes possible by multiple clients. It is expected that in large software development teams, code quality check service will be used frequently by the developer IDE systems.

Through the decomposing process of input files, data obtained as a result will be analyzed using the implemented rules. In order to accomplish this task, "rule checks" of this input source file are necessary. The system then makes these rule checks. Each rule is evaluated and an individual rank value is obtained from the execution of each rule. Based on the ranks obtained from each rule, a code quality index is then calculated by the results from the rules. Then all of the results will be collected in an object in JSON format and sent as response to the incoming request coming from the client computer user interface.

Table 1: Defined rule set.

Rule Name	Category	Content
duplicatingLibraryNames	Code Quality	Checks for duplicate libraries.
duplicatingSpecifiers	Code Quality	Translation results Checks for duplicate library modules.
namingConvention	Code Standards	Controls the naming of defined variables and function parameters.
methodReturnType	Code Quality	Controls the return data types of functions.
methodLOC	Code Quality	Controls the line count of functions.
methodArgumentType	Code Quality	Controls the data types of the parameters of the functions.
fieldType	Code Quality	Checks the data types of the variables belonging to the classes.
commentsForMethods	Code Standards	Checks if there is a descriptive comment at the beginning of the functions.

The response obtained from back end and transferred to the the front-end side is interpreted in various ways and visualization of the data is provided using a graphics library. Charts were created using the open source Highcharts library, and the incoming data is plotted using these charts and visualized [12]. Then it becomes possible to access the code quality index obtained as a result of the analysis and the existing exceptions via the user interface. This will give a stimulating and criticizing index value and graphics explanation for the developer user interface.

4. RESULTS

Although the study is only to test the viability of the methodology, it has been observed that the source code analysis and quality assesment process works successfully in general. It has been observed that file parsing performance is a key factor in order to get acceptable results. It has also been observed that the defined rules can detect undesirable situations in the file as expected. Therefore special exception handlers are necessary in order to achieve more clean and meaningful output. This, in turn is similar to some other developer who has no domain knowledge on subject of code, checking the source code quality visually. It has been observed that the "Code Quality Index" is calculated correctly, taking into account the weights and error numbers given by the user. This index has been manually checked, and verified by the developer team and research team.

It has been observed that the backend application architecture prepared with Node.js and Express.js works in relatively good performance for multiuser application to be installed on a web server. With multiuser client availability, a future system might collect requests from IDE applications which developers use for development. Then, quality assesment tasks can be handled at the background with consent of the developer.

In the study performed, some data is seemed missing in several graphics drawings in the front-end application prepared with Angular, therefore it has been observed that improvements and bug fixes were made in the front-end application to overcome these issues. In Figure 3, parsed structures are displayed with respect to their categories.

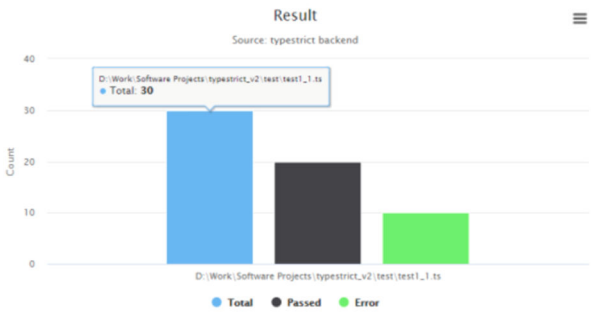

Figure 3: Analysis Display Graph 1 from interface

As for the results, for the test run, under 2 different categories, 8 rules were implemented, the success rate of 4 of checking these rules was observed to be 100%, and for 4 rules, it was observed that it remained slightly below the success target of 85%.

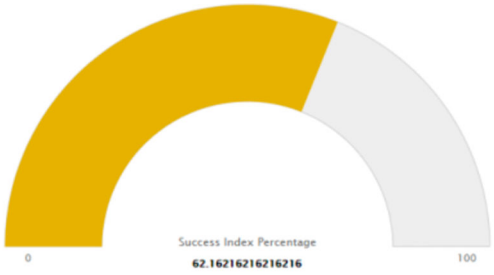

Figure 4: Analysis Display Graph 2 from interface

Speed characteristics of the system performance is also measured. It has been observed that the analysis process of a 200-line TypeScript file took less than 1 second, and it took around 500 milliseconds on average in the tests performed. Code Quality Index values were given in Figure 4 along with individual analysis results. This index was calculated as a mean value for all individual indices. In Figure 5, each rule is displayed and matching rules and violations were reported to compute resulting scores for each parameter.

It was also observed that it was very hard to pull/import a single file from the GIT system, and GIT integration was therefore abandoned temporarily because it was an efficient way to download the entire project and analyze a single file through it. Further elaborations are required in order to address this problem for full integration with GIT system to be used seamlessly as a GIT plug-in.

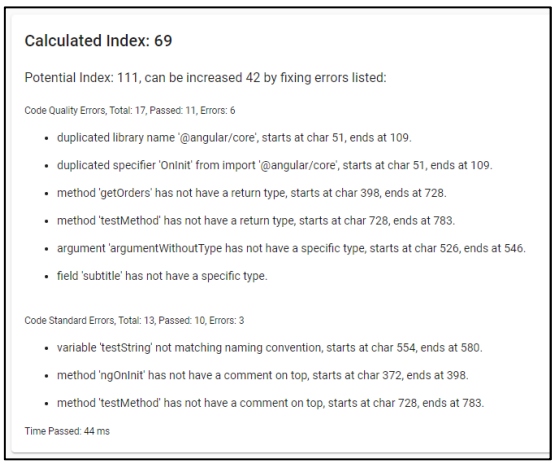

Figure 5: Analysis Display List: Index Value

5. CONCLUSION

In this study, it was aimed to measure and assess source code compliance with predefined standards by applying static code analysis on code files written in TypeScript programming language. Then as a result, it was aimed to calculate a "Code Quality Index" value for the file that was supplied.

With this study, the goal was to create a tool to help software developers using TypeScript language to work in accordance with quality code development standards and to make these standards easier to follow by embodying them with the help of a measurable metric. This objective is demonstrated in this study that it can be achieved with success.

The basic requirements within the scope of the initial goals of study were fulfilled, and a Code Quality Index was calculated for each file as a result of this analysis by applying static code analysis.

There are further improvements that can be done in various aspects of the study and these are listed as follows:

- Creating as many different test scenarios as possible and continuing to develop existing rules to improve success rates.
- Implementation of more rules, and extendible rule library for better code quality management
- Making user interface improvements to enhance interaction.
- Having multiple uploads working for large projects.
- Creating different user accounts for the web application and keeping their individual configurations in the database and managing them through the interface.
- Creating more realistic results using sophisticated methods and heuristic additional methods for calculating the Code Quality Index.
- Using machine learning algorithms to create a second quality index using tagged codebase as learning set.

Through the interviews, it has been seen that developer personnel performance and motivastion can be increased with the structure obtained as a result of this study. Further gamification and ranking approaches can also enhance this increase.

Having a server based code quality checking system will enable development of a future self learning code quality checking system which may extract rules using uploaded and developed code by the developers, slowly adapting to enterprise coding behaviour and standards. This will be studied in further research.

Today, developer satisfaction and performance are of great importance for companies. Software developers do not want to stay in institutions where old approaches are preferred. For this reason, the effect of increasing the motivation of software developers to stay in the institution will be significant.

6. REFERENCES

[1] Y. Fain and A. Moiseev, "Angular Development with TypeScript," 2nd. Manning, NY, 2018.

[2] R. Bellairs, "What Is Code Quality? And How to Improve Code Quality." perforce.com. https://www.perforce.com/blog/sca/what-code-quality-and-how-improve-code-quality (accessed Apr 4 2022).

[3] V. Raychev, "Learning to Find Bugs and Code Quality Problems-What Worked and What not?." in 2021 International Conference on Code Quality (ICCQ). IEEE, 2021, pp.1-5.

[4] "ESLint About." eslint.org. https://eslint.org/docs/about/ (accessed Apr 4 2022)

[5] "SonarQube Documentation." sonarqube.org. https://docs.sonarqube.org/latest/ (accessed Apr 10. 2022)

[6] "First Line Outsourcing, Static Analysis of JavaScript applications with SonarQube" medium.com. https://medium.com/firstlineoutsourcing/static-analysis-of-javascript-applications-with-sonarqube-1aacdf11d4ac (accessed Apr 5 2022)

[7] D. Stefanović, D. Nikolić, S. Havzi, T. Lolić and D. Dakić, "Static Code Analysıs Tools: A Systematic Literature Review." in Proc. of the 31st DAAAM International Symposium, B. Katalinic(ed.) Published by DAAAM International, 2020, pp. 565-573.

[8] E. Thoren and F. Brännlund Stål, "Usage of Angular from developer's perspective: Based on a literature and empirical study." B.S. Dissertation, Faculty of Computing Blekinge Institute of Technology, Karlskrona, Sweden, 2017.

[9] S.H.Jensen, A. Møller and P. Thiemann, "Type analysis for JavaScript," in Proc. International Static Analysis Symposium, Springer, Berlin, Heidelberg, 2009, pp. 238-255.

[10] Aseem Rastogi, N. Swamy, C. Fournet, G. Bierman and P. Vekris, "Safe & efficient gradual typing for TypeScript," in Proc. of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages 2015, pp. 167-180.

[11] J. Bogner, and M. Merkel, "To Type or Not to Type? A Systematic Comparison of the Software Quality of JavaScript and TypeScript Applications on GitHub," Preprint, in Proc. Of 19th International Conference on Mining Software Repositories MSR2022, Pittshburgh PA, 2022.

[12] "High Charts Library Documentation," highcharts.com. https://www.highcharts.com/docs/index (accessed Apr 10 2022).