Search for Structurally Similar Projects of Software Systems

Aleksey Filippov[1][0000−0003−0008−5035] and Julia Stroeva[1][0009−0003−8026−235X]

Department of Information Systems, Ulyanovsk State Technical University, 32
Severny Venetz Street, 432027 Ulyanovsk, Russia

Abstract. The authors have developed an approach to the search for structurally similar projects of software systems. Teachers can use the proposed approach to search for borrowings in the works of students. The concept behind this proposal is that it can to locate projects that students have used as parts of a current project.

The authors propose a new algorithm for determining the similarity between the structures of software projects. The proposed algorithm is based on finding similar structural elements in the source code of the program in an abstract syntax trees analyzing.

The authors developed a software system to evaluate the proposed algorithm. The current version of the system only supports Java programs. However, the system operates with its own representation of the abstract syntax tree, which allows you to add support for new programming languages.

Keywords: source code · structure analysis · structurally similar projects · hashing.

1 Introduction

Currently, the most practical work of students in information technology includes laboratory and coursework. These classes help students understand the theoretical information from their lectures.

Typically, student work is a small program that solves typical problems. In most cases, these works contain few files or a few lines of code. The architecture and algorithms of such programs are also simple.

The teacher needs to spend many time to check all the work. The teacher usually notices when the student has borrowed a program source code. Students in such cases do not change the structure of the borrowed source code, but rename variables or change types of loops (from *for* to *while*), etc.

The software system proposed in this article allows you to analyze the structure of projects and provide information about their structural similarity. The indicator of the uniqueness of the current project structure is used to evaluate the uniqueness of the project in comparison with other projects.

2 State of the art

There are no universal methods for analyzing the source code of software systems at the moment. Certain methods of analysis are used to solve various problems.

There is a group of methods for analyzing source code, which is based on obtaining and analyzing abstract syntax trees (AST). An AST is an abstract representation of the grammatical structure of a source code. It expresses the structure of a program in some programming language as a tree structure. Each AST node is an operator or a set of operators of the analyzed source code. The compiler generates an AST because of the parsing step. Unlike a parse tree, an AST does not have nodes or edges for syntax rules that do not affect the semantics of the program (for example, grouping brackets).

It can also analyze projects using call graph generation tools, such as *CodeViz* or *Egypt*. It is possible to use some functions of reverse engineering tools, such as IDA Pro.

AST-based approaches can find structurally similar projects. However, such approaches have high computational complexity [6]. Many existing approaches analyze a larger number of parameters than is necessary to solve the problem of this study [1, 3, 5–7]: project dependencies, the number of stars in the repository, the contents of the documentation, etc.

The paper [2] presents an analysis of approaches and software tools for searching for borrowings in the text and source code. However, there is no mention of software tools for searching for borrowings in the source code.

In the article [4], the authors analyze borrowings in the source code according to the sequences of using external programming interfaces and the frequency of such calls. This method is not suitable for solving the problem of this study because of the educational orientation of the analyzed source code.

Thus, it is necessary to develop an approach to the search for structurally similar projects, which are focused on working with simple software systems and with a high speed of analysis.

3 The Proposed Algorithm for Analyzing the Structure of the Source Code

The source code of the software system in the proposed algorithm is the main source of data for identifying structural features.

We formed an AST to analyze the source code. There are various libraries and tools for all existing programming languages for the formation of AST. Thus, using your own representation of the AST allows you to add support for new programming languages to the system without changing the analysis algorithms.

We will use the following AST model:

$$AST = \langle N, R \rangle,$$

where $N = \{N_1, N_2, \ldots, N_n\}$ is the set of nodes AST;

$N_i = \langle name, data \rangle$ is an i-th AST node containing the node name, node data;
R is the set of relations between AST nodes.

We developed an algorithm to highlight the structure of the project in analyzing the source code, which comprises the following steps:

1. Form an ASD for the project.
2. Select nodes with type "Class":
$$N^{Class} = \{N_i \in N | F\left(N_i.data\right) = \text{'Class'}\},$$
3. Find nodes with the "Class field" type in the found classes:
$$N^{Vars} = \{N_i^{Class} \in N^{Class} | F\left(N_i^{Class}.data\right) = \text{'Field'}\},$$
4. Find nodes with the "Methods" type in the found classes:
$$N^{Methods} = \{N_i^{Class} \in N^{Class} | F\left(N_i^{Class}.data\right) = \text{'Method'}\},$$
5. Find nodes with type "Method Argument" in the found methods:
$$N^{MethodsArgs} = \{N_i^{Methods} \in N^{Methods} | F\left(N_i^{Methods}.data\right) = \text{'Arg'}\},$$
6. Find nodes with the type "Operator" in the found methods:
$$N^{MethodsOps} = \{N_i^{Methods} \in N^{Methods} | F\left(N_i^{Methods}.data\right) = \text{'Operator'}\},$$
7. Create based on previously got sets of AST for the analyzed source code, considering the set of relations R.
8. Save the resulting AST in a graph database (GDB) to facilitate data handling.

Figure 1 shows an example of the source code and the ASD got for it.

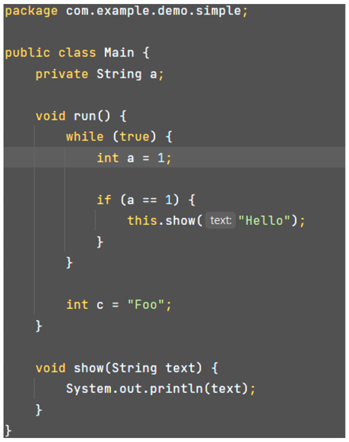
GDB is a non-relational type of database based on the topographic structure of the network. Graphs represent sets of data as nodes, edges, and properties. Relational databases provide a structured approach to data. GDBs are more flexible and focused on fast data acquisition, considering various types of links between them.

We used Neo4j as the GDB, since this system has a high speed of operation even with a large amount of stored data.

4 The Proposed Algorithm for Determining the Structural Similarity of Software Projects

The determination of the structural similarity of projects is based on the use of a hashing algorithm. We use a hash function to collapse an input array of any size into string.

It is necessary to get the values of the AST hash function of the analyzed project:



```
package com.example.demo.simple;

public class Main {
    private String a;

    void run() {
        while (true) {
            int a = 1;

            if (a == 1) {
                this.show( text: "Hello");
            }
        }

        int c = "Foo";
    }

    void show(String text) {
        System.out.println(text);
    }
}
```

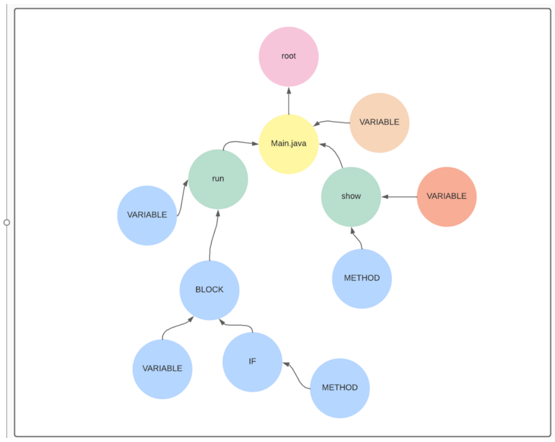


Fig. 1. Sample source code and its AST.

1. Get fragments (paths) of the AST graph from the root node of the graph tree to each node of the graph.
2. We extract the distinguishing property of each node (type) in the current fragment.
3. We passed the type of the node to the hash function. The result is an output string of a certain length.
4. If fragment A contains hash H and fragment B contains hash H, then we can say that fragments A and B have similarity in one node.
5. If fragment A contains hash H, and fragment B does not contain hash H, then fragments A and B do not have similar nodes.

We calculate the hash function values using the Neo4j GDB using the md5 algorithm.

Thus, the number of matching hash values affects the uniqueness and borrowing rates of code in a project. The following expression is used to calculate project originality:

$$O = \frac{H^C \notin H}{H^C},$$

where H^C is the set of values of the hash function of the current project;
H is the set of hash values of other projects in the system.

5 Description of the Developed Software System

Figure 2 shows the developed software system. The system comprises three nodes, which are on different nodes of the computer network.

Users interact with the web client on the *Frontend* node. The *Backend* node performs the main business logic for implementing the proposed approach in

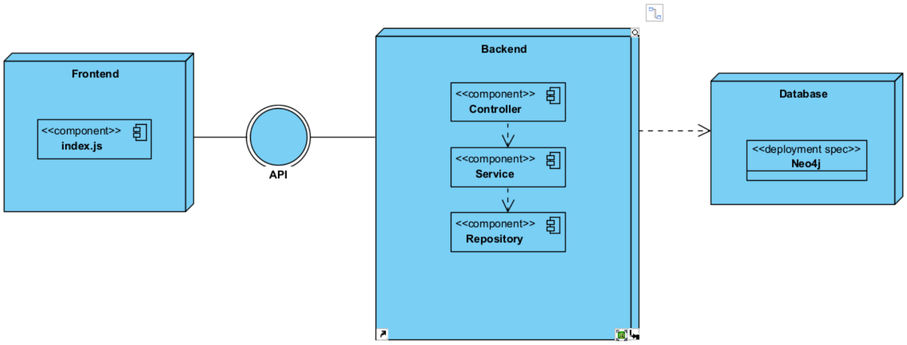


Fig. 2. Deployment diagram.

searching for structurally similar projects. *Backend* and *Frontend* nodes communicate through an API. The GDB is on the *Database* node and provides a saving of project data.

The developed software system represents a web application using a client-server architecture. The web client sends requests to the server, the server processes the requests and returns responses to the web client.

The web client is an application written in JavaScript using the Vue js framework. Vue.js is a framework for developing single page applications and web interfaces. The main advantages of this framework are its lightness (small size of the library in lines of code), performance, flexibility, and excellent documentation.

We implement the server part of the application in Java using the Spring Boot framework. The Spring framework is a whole ecosystem for developing applications in the Java language, which includes a huge number of ready-made modules. Spring Boot extends the Spring framework. The main advantages of this framework include speed and ease of development, auto-configuration of all components, easy access to databases and network capabilities.

The current version of the software system only supports source code analysis in the Java. The JavaParser library is used to form an AST in the process of Java code analysis. This library allows you to build an internal representation of the AST, which is then translated into the proposed AST model using the previously discussed algorithm.

Figure 3 shows an example of the internal representation of the AST in the JavaParser library.

We used neo4j GDB for data storage. Possibly redundant expression GDBs over relational ones is the ability to change the data model.

The GDB data model allows you to store the following nodes:

- nodes with type “Package” (Java-specific);
- nodes with type “Class”;
- nodes with the “Class field” type;

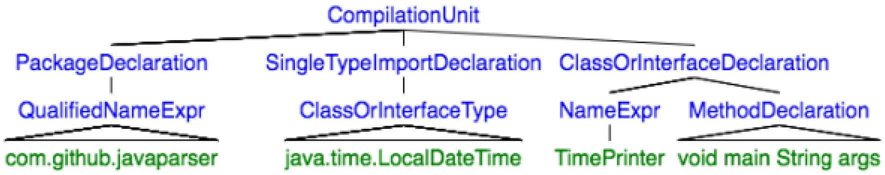


Fig. 3. An example of the internal representation of the JavaParser library AST.

- nodes with type “Method”;
- nodes with type “Method argument”;
- nodes with type “Operator”.

We arrange the nodes in the GDB hierarchically. For example, a class is in a package, but a method is in a class. The data model allows you to form the following relationships between graph nodes:

- HAS_CLASS is a relationship between a package and a class;
- HAS_FIELD is a relationship between a class and a class field;
- HAS_METHOD is a is a link between a class and a method;
- HAS_ARG is a relationship between method and method argument;
- HAS_BLOCK is a link between a method and a statement.

The main idea of searching for structurally similar projects is to use hashing of graph fragments (paths) based on the md5 function. We formed the path from the root node to each node of the graph. We take nesting into account when forming the path (package -> class -> field / method -> method argument / operator). The hash function takes the string representation of the node type as input.

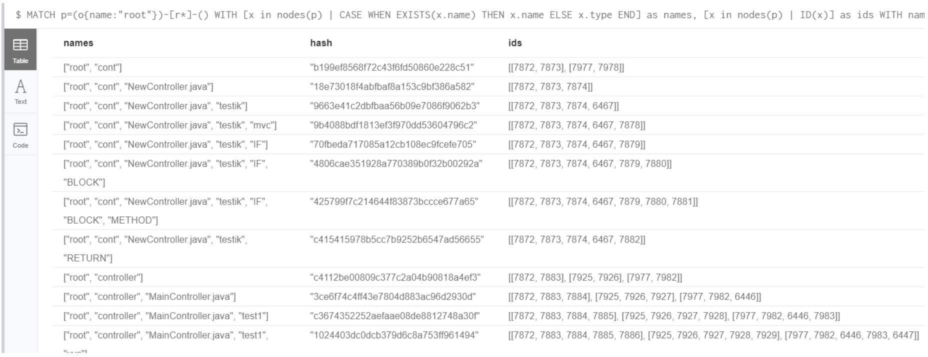
Example of a request to get paths and a hash that matches this path:

```
MATCH p = (o{name:"root"})-[r*]- ()
WHERE ID(o)={0}
WITH [x in nodes(p) | CASE WHEN EXISTS(x.name)
THEN x.name ELSE x.type END] as names,
     [x in nodes(p) | ID(x)] as ids
WITH names, apoc.util.md5(names) as hash, ids
RETURN names, hash, ids
```

Figure 4 shows the result of the query execution.

Figure 4 shows that the hash *"b199ef8568f72c43f6fd50860e228c51"* matches the path *[“root”, “cont”]*. Two graphs contain this hash. The primary keys of the root nodes of these graphs are 7872 and 7977.

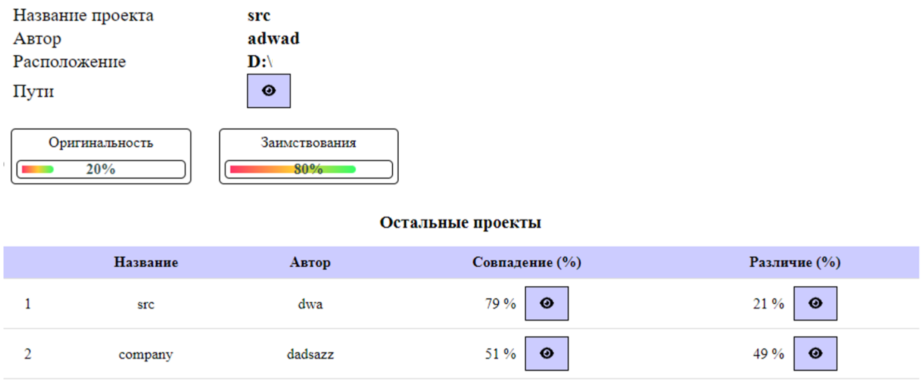
Thus, we can discover the number of matching and different paths in the analyzed projects by obtaining hashes for all AST fragments. Figure 5 shows an example of the developed system.



```
$ MATCH p=(o{name:"root"})-[r*]-() WITH [x in nodes(p) | CASE WHEN EXISTS(x.name) THEN x.name ELSE x.type END] as names, [x in nodes(p) | ID(x)] as ids WITH nam
```

names	hash	ids
["root", "cont"]	"b199ef8568f72c43f6fd50860e228c51"	[[7872, 7873], [7977, 7978]]
["root", "cont", "NewController.java"]	"18e73018f4abfbaf8a153c9bf386a582"	[[7872, 7873, 7874]]
["root", "cont", "NewController.java", "testik"]	"9663e41c2dbfbaa56b09e7086f9062b3"	[[7872, 7873, 7874, 6467]]
["root", "cont", "NewController.java", "testik", "mvc"]	"9b4088bdf1813ef3f970dd53604796c2"	[[7872, 7873, 7874, 6467, 7878]]
["root", "cont", "NewController.java", "testik", "IF"]	"70fbeda717085a12cb108ec9fcefe705"	[[7872, 7873, 7874, 6467, 7879]]
["root", "cont", "NewController.java", "testik", "IF", "BLOCK"]	"4806cae351928a770389b0f32b00292a"	[[7872, 7873, 7874, 6467, 7879, 7880]]
["root", "cont", "NewController.java", "testik", "IF", "BLOCK", "METHOD"]	"425799f7c214644f83873bccce677a65"	[[7872, 7873, 7874, 6467, 7879, 7880, 7881]]
["root", "cont", "NewController.java", "testik", "RETURN"]	"c415415978b5cc7b9252b6547ad56655"	[[7872, 7873, 7874, 6467, 7882]]
["root", "controller"]	"c4112be00809c377c2a04b90818a4ef3"	[[7872, 7883], [7925, 7926], [7977, 7982]]
["root", "controller", "MainController.java"]	"3ce6f74c4ff43e7804d883ac96d2930d"	[[7872, 7883, 7884], [7925, 7926, 7927], [7977, 7982, 6446]]
["root", "controller", "MainController.java", "test1"]	"c3674352252aefaae08de8812748a30f"	[[7872, 7883, 7884, 7885], [7925, 7926, 7927, 7928], [7977, 7982, 6446, 7983]]
["root", "controller", "MainController.java", "test1",	"1024403dc0dcb379d6c8a753ff961494"	[[7872, 7883, 7884, 7885, 7886], [7925, 7926, 7927, 7928, 7929], [7977, 7982, 6446, 7983, 6447]]

Fig. 4. An example of the result of a request to Neo4j.



Название проекта	**src**
Автор	**adwad**
Расположение	**D:\**
Пути	👁

Остальные проекты

	Название	Автор	Совпадение (%)	Различие (%)
1	src	dwa	79 % 👁	21 % 👁
2	company	dadsazz	51 % 👁	49 % 👁

Fig. 5. System operation example.

6 Experiments

7 State of the art

7.1 A Subsection Sample

Please note that the first paragraph of a section or subsection is not indented. The first paragraph that follows a table, figure, equation etc. does not need an indent, either.

Subsequent paragraphs, however, are indented.

Sample Heading (Third Level) Only two levels of headings should be numbered. Lower level headings remain unnumbered; they are formatted as run-in headings.

Sample Heading (Fourth Level) The contribution should contain no more than four levels of headings. Table 1 gives a summary of all heading levels.

Table 1. Table captions should be placed above the tables.

Heading level	Example	Font size and style
Title (centered)	**Lecture Notes**	14 point, bold
1st-level heading	**1 Introduction**	12 point, bold
2nd-level heading	**2.1 Printing Area**	10 point, bold
3rd-level heading	**Run-in Heading in Bold.** Text follows	10 point, bold
4th-level heading	*Lowest Level Heading.* Text follows	10 point, italic

–

1.

Displayed equations are centered and set on a separate line.

$$N^{Class} = \{N_i \in N | F\left(N_i.data\right) = \text{'Class'}\},$$

Please try to avoid rasterized images for line-art diagrams and schemas. Whenever possible, use vector graphics instead (see Fig. ??).

Theorem 1. *This is a sample theorem. The run-in heading is set in bold, while the following text appears in italics. Definitions, lemmas, propositions, and corollaries are styled the same way.*

Proof. Proofs, examples, and remarks have the initial word in italics, while the following text appears in normal font.

For citations of references, we prefer the use of square brackets and consecutive numbers. Citations using labels or the author/year convention are also acceptable. The following bibliography provides a sample reference list with entries for journal articles [?], an LNCS chapter [?], a book [?], proceedings without editors [?], and a homepage [?]. Multiple citations are grouped [?,?,?], [?,?,?,?].

Acknowledgements Please place your acknowledgments at the end of the paper, preceded by an unnumbered run-in heading (i.e. 3rd-level heading).

References

1. Aleksey Alekundrovich, F., Yurevich, G.G., Aleksey Michailovich, N., Nudezhda Glebovna, Y.: Approach to the search for software projects similar in structure and semantics based on the knowledge extracted from existed projects. In: Computational Science and Its Applications–ICCSA 2020: 20th International Conference, Cagliari, Italy, July 1–4, 2020, Proceedings, Part I. pp. 718–733. Springer (2020)
2. Ali, A.M.E.T., Abdulla, H.M.D., Snasel, V.: Overview and comparison of plagiarism detection tools. In: Dateso. pp. 161–172 (2011)
3. Beniwal, R., Dahiya, S., Kumar, D., Yadav, D., Pal, D.: Npmrec: Npm packages and similar projects recommendation system. In: Data Analytics and Management: Proceedings of ICDAM. pp. 701–710. Springer (2021)
4. Chae, D.K., Ha, J., Kim, S.W., Kang, B., Im, E.G.: Software plagiarism detection: a graph-based approach. In: Proceedings of the 22nd ACM international conference on Information & Knowledge Management. pp. 1577–1580 (2013)
5. Nadezhda, Y., Gleb, G., Pavel, D., Vladimir, S.: An approach to similar software projects searching and architecture analysis based on artificial intelligence methods. In: Proceedings of the Third International Scientific Conference “Intelligent Information Technologies for Industry”(IITI’18) Volume 1 3. pp. 341–352. Springer (2019)
6. Nguyen, P.T., Di Rocco, J., Rubei, R., Di Ruscio, D.: Crosssim: exploiting mutual relationships to detect similar oss projects. In: 2018 44th Euromicro conference on software engineering and advanced applications (SEAA). pp. 388–395. IEEE (2018)
7. Nguyen, P.T., Di Rocco, J., Rubei, R., Di Ruscio, D.: An automated approach to assess the similarity of github repositories. Software Quality Journal **28**, 595–631 (2020)