Search for Structurally Similar Projects of Software Systems

Aleksey Filippov$^{1[0000-0003-0008-5035]}$, Anton Romanov$^{1[0000-0001-5275-7628]}$, 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 works. 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 each other.

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.

We can analyze projects using call graph generation tools, such as *CodeViz* or *Egypt*. Or we can use of reverse engineering tools, such as IDA Pro. The call graph based approaches allow developers to solve the program comprehension task for better program maintenance or to reduce security issues [7, 10–12].

Another group of methods 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 as a structured tree and rarely depends on programming language. Each AST node is an operator or a set of operators of the analyzed source code. The compiler generates an AST on the parsing step. Unlike a parse tree, an AST does not contain nodes or edges that do not define the semantics of the program (for example, grouping brackets).

AST-based approaches allow us to find structurally similar projects. However, such approaches have high computational complexity [8]. Many existing approaches analyze a larger number of parameters than is necessary to solve the problem of this study [4, 6, 8, 9, 13]: project dependencies, the number of stars in the repository, the contents of the documentation, etc.

The paper [3] presents an review of approaches and software tools for borrowings searching in the text and source code. However, there is no mention of existing software tools for borrowings searching in the source code.

In the article [5], the authors analyze borrowings in the source code according to the sequences of using external programming interfaces (external dependencies) and the frequency of such calls. This method is not suitable for solving the problem of this study because of the educational orientation. Some student projects can not use external dependencies.

Thus, it is necessary to develop an approach to the search for structurally similar projects, which are focused on simple software systems and a high speed of analysis. A set of source code files will be considered as projects.

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

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

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. We use own representation of the AST to add support for new programming languages without changing the analysis algorithms.

We define the proposed AST model as follows:

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

where $N = \{N_1, N_2, \ldots, N_n\}$ is the set of AST nodes;
$N_i = \langle name, data \rangle$ is an i-th AST node containing the node name and data;
R is the set of relations between AST nodes.

We developed an algorithm to extract the structure of the project in the source code analyzing. The proposed algorithm contains the following steps:

1. Select nodes with the 'Class' type as the N^{Class} set:
$$N^{Class} = \{N_i \in N | F\left(N_i.data\right) = \text{`Class'}\},$$
2. Select nodes with the 'Class field' as the N^{Vars} set from the N^{Class} set:
$$N^{Vars} = \{N_i^{Class} \in N^{Class} | F\left(N_i^{Class}.data\right) = \text{`Field'}\},$$
3. Select nodes with the 'Methods' type as the $N^{Methods}$ set from the N^{Class} set:
$$N^{Methods} = \{N_i^{Class} \in N^{Class} | F\left(N_i^{Class}.data\right) = \text{`Method'}\},$$
4. Select nodes with the 'Method Argument' type as the $N^{MethodsArgs}$ set from the $N^{Methods}$ set:
$$N^{MethodsArgs} = \{N_i^{Methods} \in N^{Methods} | F\left(N_i^{Methods}.data\right) = \text{`Arg'}\},$$
5. Select nodes with the 'Operator' type as the $N^{MethodsOps}$ from the $N^{Methods}$ set:
$$N^{MethodsOps} = \{N_i^{Methods} \in N^{Methods} | F\left(N_i^{Methods}.data\right) = \text{`Operator'}\},$$
6. Create the set of ties R between the nodes from sets obtained in previous steps.
7. Save the resulting AST in a graph database (GDB).

In this algorithm, F (*) is a search function that finds nested nodes. The function input is a node or subtree, and the output is a node of the desired type.

Figure 1 shows the resulting AST for the following source code:

```
package com.example.demo.simple;
public class Main {
    private String a;
    void run() {
        while(true) {
            int a = 1;
            if (a == 1) {
                this.show("Hello")
            }
        }
        int c = "Foo";
    }
    void show(String text) {
        System.out.println(text);
    }
}
```

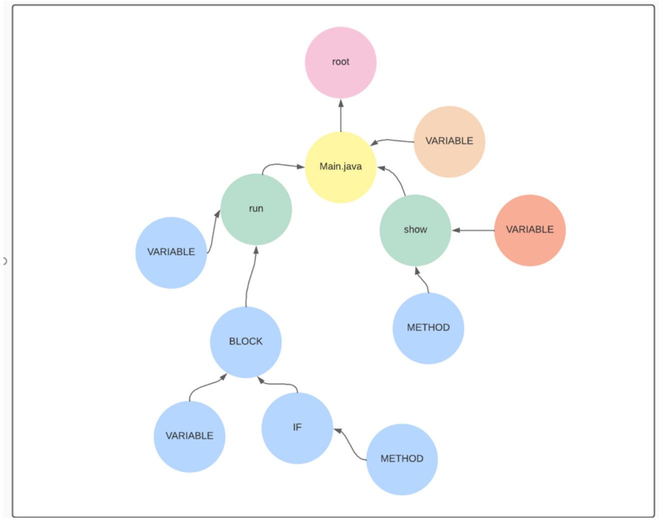


Fig. 1. Sample source code and its AST.

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. GDBs are more flexible than relational databases. GDBs are more flexible than relational databases and allow you to fast obtain data of various types, considering numerous relations.

We use the Neo4j [1] GDB as the data storage. Neo4j is a graph database management system. Neo4j stores nodes, edges connecting them, and attributes of nodes and edges. Neo4j has a high speed of operation even with a large amount of stored data.

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

The detection of the projects structural similarity is based on the hashing algorithm. We use a hash function to minimize the size of an input data.

The proposed AST hashing algorithm is contains from the following steps:

1. Select all paths of the AST graph from the root node to each other node.
2. Get a value of the 'type' property for each node of the current path.
3. Calculate an MD5 hash function for the current path. We calculate the hash functions using the apoc.util.md5 plugin for the Neo4j. As a result of this step, formed a set that contains a tuple of the following values:
   - a path,
   - a path md5 hash.

   For example:

- <‘root->class->method->if’, ‘820b...9c4b’>,
- <‘root->class->field’, ‘6161...eab3’>.

The following expression is used to calculate project originality:

$$O = \frac{H^C \notin H}{H^C}, \tag{1}$$

where H^C is a set of hash functions of the analyzed project;
H is a set of hash values of other projects in the system.

5 Results

5.1 Architecture of the developed system

Figure 2 shows the deployment diagram in the UML notation of the developed software system. The developed system has the three-tier architecture.

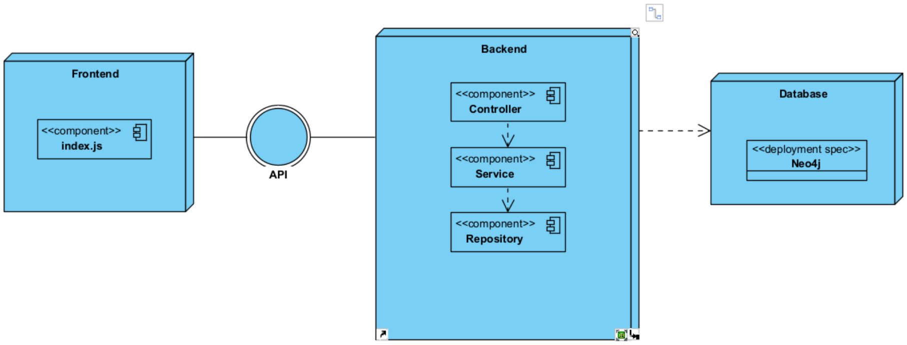


Fig. 2. Deployment diagram.

Users interact with the web client on the *Frontend* node. The *Backend* node performs the main business logic for searching of the structurally similar projects. *Backend* and *Frontend* nodes communicate through an API. The *Database* provides data storage functions.

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

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

The current version of the software system supports only Java-based software projects. The JavaParser library is used to form an AST in the Java source code analysis. This library allows you to extract the AST using the previously discussed algorithm.

5.2 Data model for representing AST as a GDB fragment

In this subsection, we discussed the proposed data model for representing AST as a GDB fragment.

The GDB data model contains the nodes with the following type:

- 'Package' (Java-specific),
- 'Class',
- 'Class field',
- 'Method',
- 'Method argument',
- 'Statement' (declaration, expression and control statements).

We arrange the nodes in the GDB hierarchically. For example, a class-node is a part of a package-node, a method-node is a part of a class-node. The data model allows you to form the following ties between data model nodes:

- 'HAS_CLASS' is a relationship between a 'Package' and a 'Class' nodes,
- 'HAS_FIELD' is a relationship between a 'Class' and a 'Class field' nodes,
- 'HAS_METHOD' is a relationship between a 'Class' and a 'Method' nodes,
- 'HAS_ARG' is a relationship between 'Method' and 'Method argument' nodes,
- 'HAS_BLOCK' is a link between a 'Method' and a 'Statement'.

The proposed algorithm for searching for structurally similar projects is to use hashing of graph paths based on the md5 function. We describe the hashing algorithm in the previous section. The searching algorithm can be represented as the following Cypher-query:

```
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
```

Table 1 shows the result of the Cypher-query.

Table 1 shows that:

- the hash '346f...a463' matches the path 'root->package',
- the hash '840b...7f9a' matches the path 'root->package->class',

Table 1. An example of the result of the searching Cypher-query.

names	hash	ids
root->package	**"346f...a463"**	[[7872, 7873], [7977, 7978]]
root->package->class	**"840b...7f9a"**	[[7872, 7873, 7874], [7977, 7978, 7979]]
root->package->class ->method	**"7151...0f3d"**	[[7872, 7873, 7874, 7875], [7977, 7978, 7979, 7980]]
root->package->class ->method ->statement.control	"5810...f0c9"	[[7872, 7873, 7874, 7875, 7879]]
root->package->class ->method ->statement.control ->statement.expression	"fd3c...5a3c"	[[7872, 7873, 7874, 7875, 7879, 7880]]
	...	
root->package	**"346f...a463"**	[[7872, 7873], [7977, 7978]]
root->package->class	**"840b...7f9a"**	[[7872, 7873, 7874], [7977, 7978, 7979]]
root->package->class ->method	**"7151...0f3d"**	[[7872, 7873, 7874, 7875], [7977, 7978, 7979, 7980]]

- the hash ‘7151...0f3d’ matches the path ‘root->package->class->method’,
- two projects with identifiers 7872 and 7977 contain this structural patterns (paths).

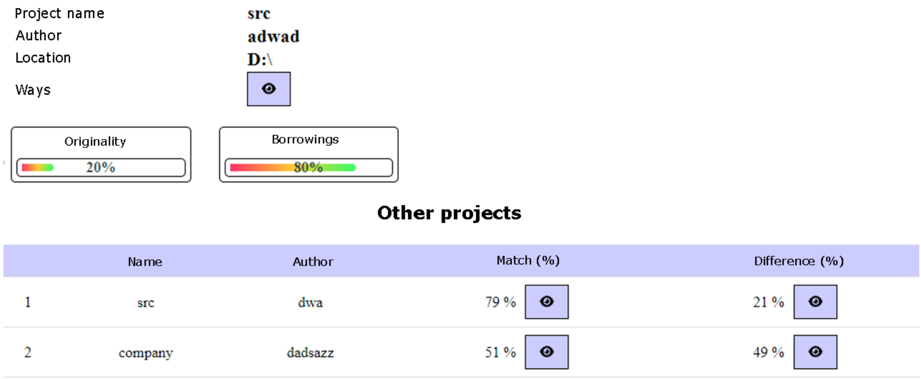
Thus, we can calculate the number of matching and not matching paths (see eq. 1) in the analyzed project compare with other projects in data storage. Figure 3 shows the main form of the developed system.

6 Experiments

We conducted experiments to evaluate the speed of source code analysis. We calculated the results relative to the number of lines of code and the number of files in the analyzing project. The main aim of the experiment is to determine the speed of the algorithm, considering the average number of lines of code processed per minute. We used the IntelliJ IDEA Statistic plugin [2] to get the data for the experiment. The plugin allows you to calculate the number, size, number of lines, average value and other information for each file in the project. You can also find out the total number of rows, the number of lines of code, the proportion of lines of code, the number of comment lines, the proportion of comment lines, etc.

We selected 10 random Java projects for this experiment. Table 2 presents the results of experiments for analyzing the speed of the proposed algorithm.

Table 3 presents the results of experiments to determine the total time of projects analyzing and the number of nodes in resulting graphs.



Project name	src
Author	adwad
Location	D:\
Ways	👁

Originality 20%

Borrowings 80%

Other projects

	Name	Author	Match (%)	Difference (%)
1	src	dwa	79 % 👁	21 % 👁
2	company	dadsazz	51 % 👁	49 % 👁

Fig. 3. The main form of the developed system.

Table 2. Results of experiments for analyzing the speed of the proposed algorithm.

#	Project name	Lines of code	Java files	Lines of code per minute
1	BaseRecycler	3 896	92	2 491
2	AlamazDev	15 776	103	2 658
3	SnakeBoom	20 534	158	3 255
4	retrofit	32 119	227	2 718
5	Glide	37 508	203	2 576
6	ZXing	51 857	310	2 533
7	RxJava	64 101	339	2 814
8	VisualProjectCore	71 303	450	2 969
9	mc-dev	85 267	877	2 746
10	xRayJavaTool	97 249	937	2 730
Average value				2 749

Table 3. Results of experiments to determine the total time of projects analyzing and the number of nodes in resulting graphs.

#	Project name	Total time (min)	Number of graph nodes
1	BaseRecycler	1.6	844
2	AlamazDev	6.0	1 837
3	SnakeBoom	6.3	2 197
4	retrofit	11.8	7 118
5	Glide	14.6	8 496
6	ZXing	20.5	10 560
7	RxJava	22.7	11 972
8	VisualProjectCore	24.1	13 334
9	mc-dev	31.1	14 444
10	xRayJavaTool	35.6	23 946

The experiment revealed that we processed an average of 2 750 lines of code per minute. Student projects contains average 500-3000 lines of code. Thus, the analysis of one project takes on average less than one minute.

7 Conclusion

This article presents the results of developing an approach and a system for searching for structurally similar projects.

We solved the following tasks:

- we analyzed existing methods of source code analysis, including the methods for borrowings searching in a text and source code;
- we developed the algorithm for extracting the AST in analyzing a project source code;
- we developed the algorithm for determining originality of a project based on the the AST structure hashing;
- we implemented the software system to determine originality of a project;
- we conducted experiments to determine the speed of the proposed algorithm.

Thus, the developed system makes it possible to find borrowings in student projects in less than a minute on average.

Acknowledgements The authors acknowledge that the work was supported by the framework of the state task No. 075-03-2023-143 "Research of intelligent predictive analytics based on the integration of methods for constructing features of heterogeneous dynamic data for machine learning and methods of predictive multimodal data analysis".

References

1. Neo4j graph database & analytics | graph database management system, https://neo4j.com/, [Online; accessed 11-June-2023]
2. Statistic - intellij ides plugin, https://plugins.jetbrains.com/plugin/4509-statistic, [Online; accessed 11-June-2023]
3. Ali, A.M.E.T., Abdulla, H.M.D., Snasel, V.: Overview and comparison of plagiarism detection tools. In: Dateso. pp. 161–172 (2011)
4. 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)
5. 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)
6. Filippov, A., Guskov, G., Namestnikov, A., Yarushkina, N.: 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)

7. Ghavamnia, S., Palit, T., Mishra, S., Polychronakis, M.: Temporal system call specialization for attack surface reduction. In: Proceedings of the 29th USENIX Conference on Security Symposium. pp. 1749–1766 (2020)
8. 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)
9. 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)
10. Soares, D., Pereira, M.J.V., Henriques, P.R.: Integrating a graph builder into python tutor. In: Second International Computer Programming Education Conference (ICPEC 2021). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2021)
11. Tang, F., Østvold, B.M.: Assessing software privacy using the privacy flow-graph. In: Proceedings of the 1st International Workshop on Mining Software Repositories Applications for Privacy and Security. pp. 7–15 (2022)
12. Vinayaka, K., Jaidhar, C.: Android malware detection using function call graph with graph convolutional networks. In: 2021 2nd International Conference on Secure Cyber Computing and Communications (ICSCCC). pp. 279–287. IEEE (2021)
13. Yarushkina, N., Guskov, G., Dudarin, P., Stuchebnikov, V.: 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)