

Code2Algo: Eclipse Tool for code recommendation

Lucas Raggi, Hyago P. Brito, Arthur Bernardo, João M. L. Pereira,
Balduino Fonseca, Márcio Ribeiro, Jairo Souza

¹Instituto de Computação
Universidade Federal de Alagoas (UFAL) - Maceió - Brasil

{lrr, hpb, asbm, jmp}@ic.ufal.br

Abstract. *During software development, developers may implement specific algorithms that need a comprehension of the correct implementation of them. However, it is humanly impossible to remember the whole variety of algorithms. In this context, we propose Code2Algo, a integrated development environment tool based on the Eclipse platform that receives a snippet and provides the complete algorithm. Thus, this tool comes to help the developers, improving their performance and gaining time, since you will not need to search for it outside the platform.*

1. Introduction

An algorithm is a finite sequence of computational steps that transform the input into the output that aims to obtain a solution for a specific type of problem [Leiserson et al. 2002]. In this perspective, developers need to use algorithms to solve problems during software development activities, but not always they know how to implement these solutions because some algorithms could not be trivial. In addition, developers may have an effort in obtaining implementations of algorithms through external resources.

Several studies have presented recommender systems to assist software engineering tasks using data mining techniques or machine learning approaches. For example, they suggest recommendations for method invocations and constructors [Bruch et al. 2009], calls to the APIs [Raychev et al. 2014], method names [Alon et al. 2018], and generic codes through similar codes [Luan et al. 2018]. However, none of these studies provided recommendation systems with the possibility of recommending algorithms to assist developers during day-to-day activities.

To support developers during development tasks and provided an integrated development environment tool (IDE), we present in this paper Code2Algo¹, an Eclipse *plug-in* tool to recommend algorithms for developers. When using this tool, developers can receive up to three algorithms recommendations based on the incomplete code that they are writing at moment in the IDE.

To perform our study, we applied our script, Java Line Remover, in educational repositories containing algorithms to train our machine learning model with incomplete codes.

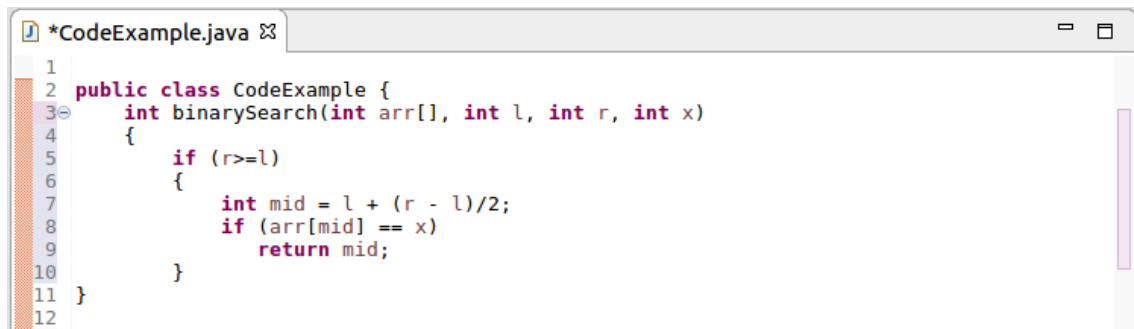
2. Motivational Example

To implement algorithms, developers may have a previous knowledge or experience to perform this. However, various algorithms was created with different purpose, and a

¹<http://www.code2algo.com>

higher effort is necessary to know all implementations of existing algorithm. This is aggravated for novice developers, which have high chances to get stuck when they do not know how to implement a specific algorithm during software development. To mitigate this, Code2Algo helps developers getting a piece of the code, and recommending a full code with the algorithm.

For example, Figure 1 presents incomplete source code of a *Binary Search* algorithm. During software development, developers may do not have the behaviour expected of our algorithm or he does not know how to finish it.

A screenshot of a Java IDE window titled '*CodeExample.java'. The code is as follows:

```
1
2 public class CodeExample {
3     int binarySearch(int arr[], int l, int r, int x)
4     {
5         if (r >= l)
6         {
7             int mid = l + (r - l) / 2;
8             if (arr[mid] == x)
9                 return mid;
10        }
11    }
12 }
```

The code is incomplete, missing the closing brace for the `binarySearch` method and the closing brace for the class.

Figure 1. Example of developer getting stuck

Therefore, with our plugin Code2Algo, instead of going outside of IDE to search for help, developers can use Code2Algo to get algorithms recommendation. As a consequence, they gain productivity because they do not need more go outside of IDE to search for external resources such as documentation or repositories with the implementations of algorithms. In some cases, just the name of the method is enough to produce results.

3. Tool

In this section, we present how our tool deals with the aforementioned problems. Then, we describe its architecture in Section 3.1.

Using a solution based on Code2Vec [Alon et al. 2018], we try to identify the name of the algorithm that the developer wants to implement. Using this structure, we have good accuracy for a complete algorithm, but we wanted to identify with an incomplete snippet or even just the name of the method in some cases. To do it, we trained a Code2Vec model with incomplete algorithms. To get the incomplete algorithms, we passed some repositories in our script called Java Line Remover, more detailed in section 3.1, with this, we got a good identifier for incomplete codes.

To recommend the more similar algorithm for the users code, we applied a ranking with three features, detailed in section 3.1, in the search result in our database, after the identification, this way we got good recommendations in each case. Every code retrieved is shown in a perspective view with expand bars to alternate between the three recommendations, very simple and easy usable where the user can copy the code retrieved from our API to use in his code. For example, Figure 2 presents our plugin recommending an algorithm to the developer. First part of the Figure 2 shows the incomplete code of a bubble sort algorithm made by a developer, while the second part presents the recommendation of a complete bubble sort provided by our tool.

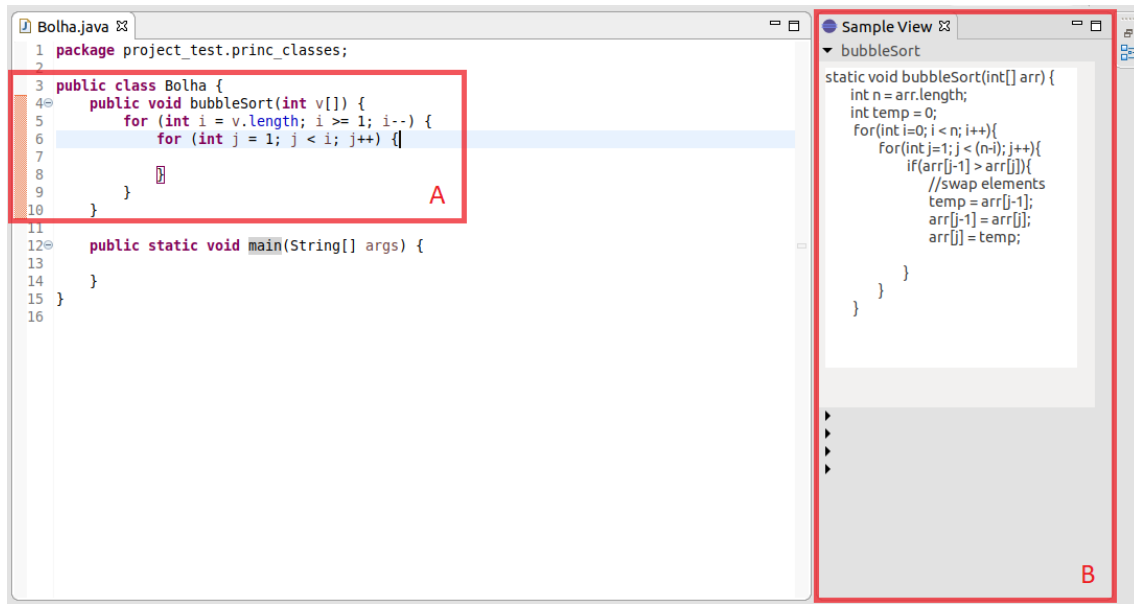


Figure 2. Integration of our tool into Eclipse IDE

In addition, our plug-in have three ways to operate, the first is based on a complete algorithm developed, can use it to review if its code is the most similar to his code and if its implementation is right. Second, if the developer do not know the implementation of certain algorithm, but knows its name, can try naming a method to retrieve the full algorithm in our tool. Last, if the developer started the algorithm, but got stuck in the middle, he can use our plug-in to finish his code.

As shown in the Figure 2, the in development *Bolha* shown in area A was identified by our API as a *BubbleSort* and returned a full code which is exhibited in area B.

3.1. Architecture

Here we will describe the architecture of our tool. It is an Eclipse plug-in that uses an interface based on a perspective view. We can see in Figure 1 that the user only have access to the plug-in interface, which communicates with our API, specified in Figure 2. The principal modules are described below:

- **Code :Identifier:** Implementation of Code2Vec [Alon et al. 2018] trained with our created database with incomplete code as shown in Figure 3 that demonstrate all steps for database creation. For all codes selected, we passed it through a script called Java Line Remover, where we get a full algorithm code, and start removing a number of lines from it in a way where it is still compilable. This way we can get correct pieces of code to train our Code2Vec. The code identifier receives from the plug-in a JSON object containing the developer's full code and the name of the method he is typing. The Identifier finds the method in the object and split it to use in our model of Code2Vec, as output we have the name of the algorithm being developed.
- **Code Extractor:** Module of our API where we pick a method and extract the most relevant information from it, which are the return type, number and type of parameters. It is used in the code received from user and in the returned codes from our database.

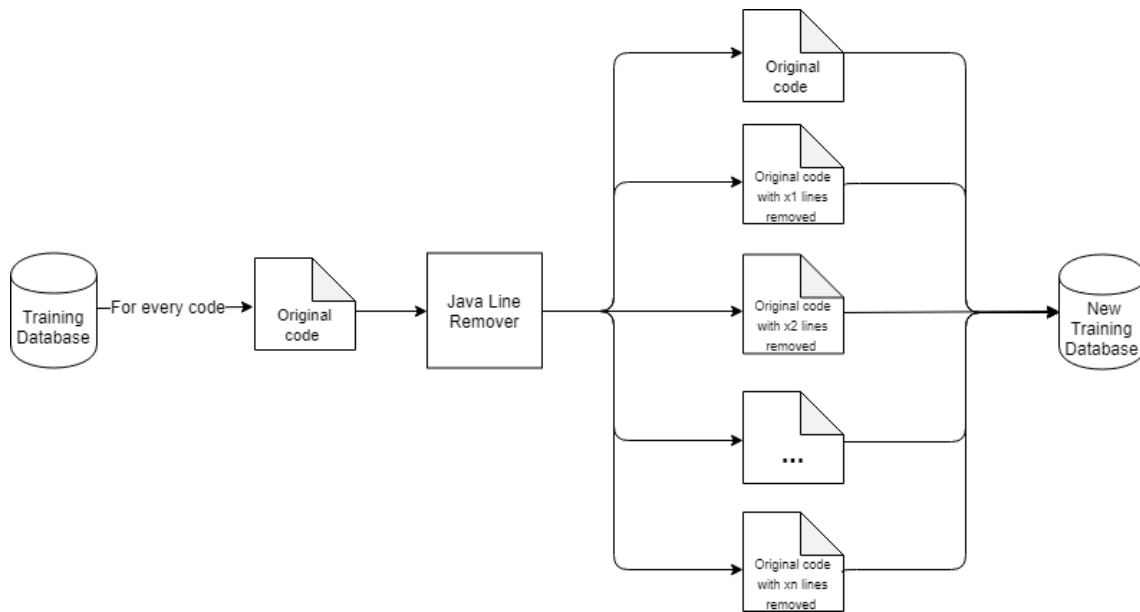


Figure 3. Java Line Remover. Used to create our Training Database.

- **Code Recommender:** Part of API where the ranking is made and based on it, return the code for user. The ranking is made based on three parameters which are extracted in Code Extractor, they are: number of parameters, type of return and parameters. Each of this metrics receives a score from 0 to 1 based on the similarity between the user code and the candidates, which are algorithms found in our database. Then from the candidates, the best three are selected and returned for user.
- **Interface:** The view for user, it is a perspective view, developed to be easy to understand and use. As developer types in the editor, our plug-in is listening in background. When the return comes from our API, we show the recommendations as demonstrated in Figure 2 Area B.
- **Recommender Database:** A manually selected database composed by algorithms from educational repositories made by experienced developers.

Code2Algo works in the following way. First, the user opens our plug-in and start typing the code. Then, the tool, recognizing that a method is being typed, sends its snippet to API. In the next step, the Code Identifier module identifies if the method is an algorithm and which algorithm is being developed. After that, the Code Recommender searches for identified code in the Recommender Database and ranks them based on the similarity with the incomplete snippet made by user. Finally, the plug-in shows the return of API for user, which decides if he wants to use it or not.

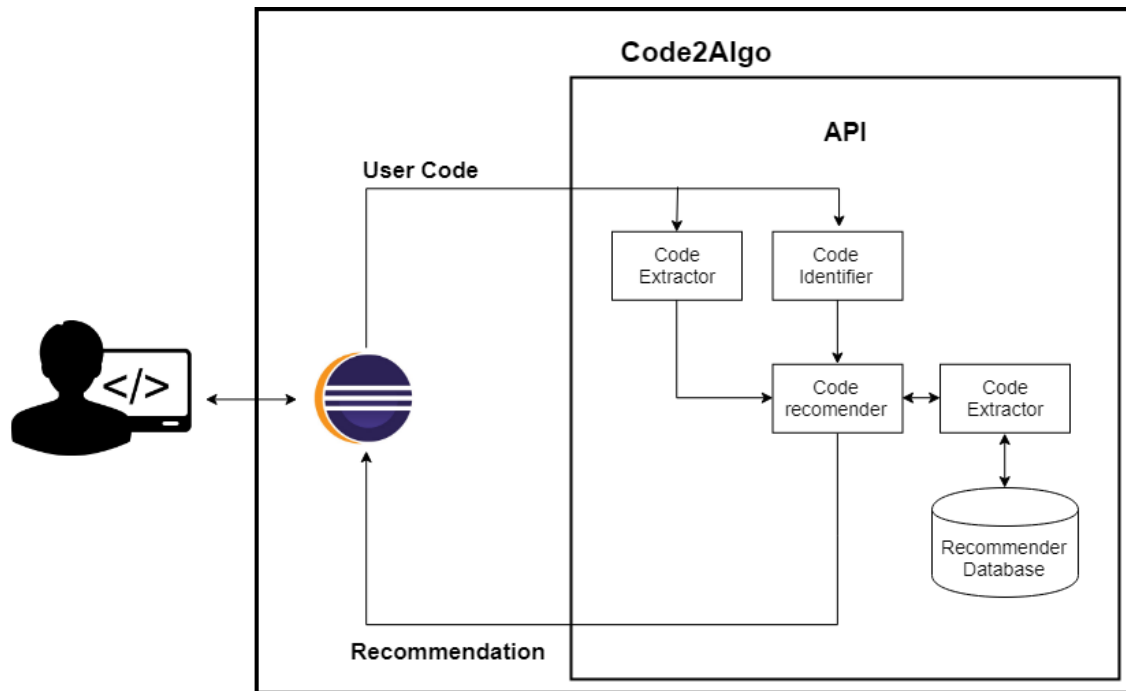


Figure 4. Flow Diagram

4. Related Work

Our tool relates directly with Code2Vec [Alon et al. 2018] that identify the algorithm based on a snippet, but with incomplete codes, it might not be accurate. Our Code Identifier is focused on, given an incomplete snippet of code, identify the algorithm.

Aroma [Luan et al. 2018], is a tool that do almost the same job, identify the algorithm using a similar version of Code2Vec [Alon et al. 2018], do a light search on its database, select candidates, cluster them, and intercept for a good recommendation. But, it is not focused on algorithms, our tool focus on algorithms, like sort, search and others, which implementation can stuck developers, professionals and novices.

[Bruch et al. 2009] wanted to improve the system of code completions by learning from examples. Basically his algorithm extract the context of the variable, search for variables used in similar situations and then synthesize method recommendations out of these nearest snippets. But it focus on recommendations by relevance for user instead of alphabetical order, which is the recommendation used in Eclipse. Our tool wants to recommend based on similarity with user code, but we recommend a full algorithm, not only methods.

Futhermore, [Murali et al. 2017] generates code to complete the function that developer wants to create, it does not use the lines of code that user is typing, but "draft program" instead, a method with holes and "queries" that are tips you give about the code, what functions it will call, variables to create, etc. Code2Algo needs only the lines of code that developer is typing and is restricted to algorithms.

5. Conclusion

This paper presented Code2Algo, a tool for Eclipse, which support the developers writing the correct version of the algorithm. The tool consists mainly of an Eclipse plug-in, that identify if the developer is writing an algorithm, and shows to him the complete code recommendation of it, and an API, which, given the snippet sent by the plug-in, identify the algorithm, search for similar codes, and returns three most similar to developer code recommendations. We applied this tool for the most used types of algorithms, like search, order, and others. As a future work, we want to add more options to our interface, so, the plug-in adapts for each user. Besides that, improve our database, this way we can identify and recommend more detailed and specific algorithms.

References

- Alon, U., Zilberstein, M., Levy, O., and Yahav, E. (2018). code2vec: Learning distributed representations of code. *CoRR*, abs/1803.09473.
- Bruch, M., Monperrus, M., and Mezini, M. (2009). Learning from examples to improve code completion systems. page 213.
- Leiserson, C., Cormen, T., Rivest, R., and Stein, C. (2002). *Algoritmos: teoria e prática*. ELSEVIER.
- Luan, S., Yang, D., Sen, K., and Chandra, S. (2018). Aroma: Code recommendation via structural code search. *CoRR*, abs/1812.01158.
- Murali, V., Chaudhuri, S., and Jermaine, C. (2017). Bayesian sketch learning for program synthesis. *CoRR*, abs/1703.05698.
- Raychev, V., Vechev, M., and Yahav, E. (2014). Code completion with statistical language models. *ACM SIGPLAN Notices*, 49(6):419–428.