ArCode: A Tool for Supporting Comprehension and Implementation of Architectural Concerns

Ali Shokri
Dept. of Software Engineering
Rochester Institute of Technology
Rochester, NY, USA
as8308@rit.edu

Mehdi Mirakhorli
Dept. of Software Engineering
Rochester Institute of Technology
Rochester, NY, USA
mxmvse@rit.edu

***Abstract*—Integrated development environments (IDE) play an important role in supporting developers during program comprehension and completion. Many of these supportive features focus on low-level programming and debugging activities. Unfortunately, there is less support in understanding and implementing architectural concerns in the form of patterns, tactics and/or other concerns. In this paper we present ArCode, a tool designed as a plugin for a popular IDE, IntelliJ IDEA. ArCode is able to learn correct ways of using frameworks' API to implement architectural concerns such as Authentication and Authorization. Analyzing the programmer's code, this tool is able to find deviations from correct implementation and provide fix recommendations alongside with graphical demonstrations to better communicate the recommendations with the developers. We showcase how programmers can benefit from ArCode by providing an API misuse detection and API recommendation scenario for a famous Java framework, Java Authentication and Authorization (JAAS) security framework.**

***Index Terms*—API Specification, Architectural Concerns, API Usage Model, API Recommendation, API Misuse Detection**

I. Introduction

Nowadays program developers increasingly rely on Integrated Development Environments (IDEs) to accelerate the development pace, achieve better control over their projects, and implement programs that correctly follow the programming language syntax [1], [2]. These IDEs are supported and enhanced by enormous plugins and tools developed by experts to facilitate code implementations. IntelliJ IDEA[1] is a leading IDE developed by JetBrains which comes with many features such as code completion, code navigation, and building projects [3]. Novice, as well as expert programmers receive many helpful fine-grained advice from these tools and plugins, mostly in a limited scope [4], [5]. For example, based on a partially written expression in a program, an auto-completion plugin would be able to leverage information obtained from the method context to provide some recommendations on how to complete the expression. These plugins make programmers' life much easier, such that many program developers cannot write a simple program without them. While programmers benefit from plugins and tools provided by IDEs, there are some important concerns that have not been addressed by them. For instance, despite the advancements in programming languages and development environments, junior program developers still suffer from lack of proper comprehension of the program under development [6]–[9] including its architectural aspects [10]. Furthermore, they find it difficult to implement complicated usecases and architectural tactics from scratch [11]. These tactics are mostly implemented by incorporating APIs of frameworks. Studies show that framework documentations are not well-prepared for junior programmers to learn how to implement different usecases by incorporating that framework in their code [12]. Moreover, some empirical studies show that programmers cannot rely on tutorials nor Q&A posts to learn how to correctly incorporate these APIs for a tactic or usecase implementation [13], [14]. In these cases, even if a programmer receives fine-grained recommendations from the IDE, it would not be enough towards building a real-world functioning application. Achieving this level of knowledge and experience and applying it in designing and implementing a program is not a trivial task. Therefore, to support programmers in the mentioned scenarios, empowering IDEs with more sophisticated tools and plugins is crucial.

In this paper, we introduce ArCode, a tool that helps programmers understand implemented architectural concerns in their programs, find possible incorrect implementations, and fixing them. To that extent, Arcode learns how to correctly leverage API of frameworks from two sources: **(i)** by analyzing frameworks' bytecode, and **(ii)** from a sample of programs incorporating API of frameworks of interest. It then performs static analysis on the under development program and creates a graph-based model of API usage in that program. In an interaction with the programmer, ArCode is capable of providing recommendations on how to fix possible API misuses.

II. ArCode

ArCode is a tool for analyzing programs written in Java, creating their API usage models, identifying deviations in the code from a correct implementation, and providing recommendations on how to fix the found deviations. It is implemented as a plugin for IntelliJ IDEA development environment. There are three main inputs to ArCode:

- Information about the framework of interest (e.g. JAAS framework)

[1]https://www.jetbrains.com/idea

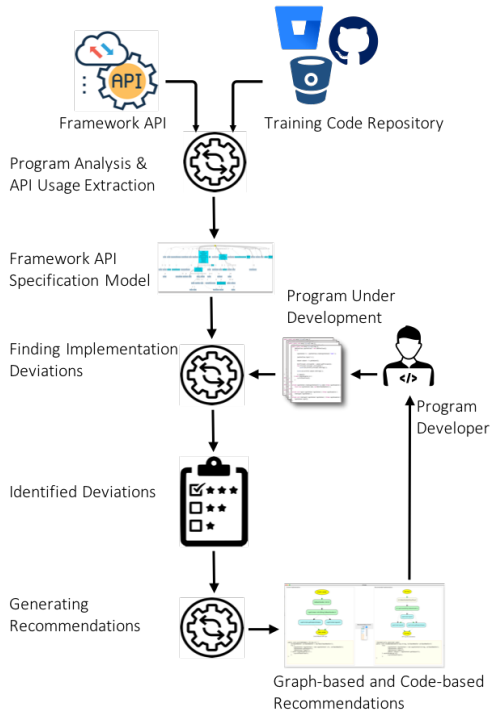


Fig. 1: ArCode main modules and their interactions.

- Path to training projects directory
- Path to the program under development

ArCode leverages static analysis techniques to extract some facts about API dependencies and their usage constraints from the framework binary code. It then analyzes programs in the training repository and filters out those that are violating API usage constraints. Mining the remained training programs, it builds a `Framework API Specification (FSpec)` model for the framework of interest. FSpec represents correct ways of leveraging a framework's API to implement usecases or architectural concerns. Next, ArCode analyzes the program under development and creates its graph-based API usage model. This model alongside with the created FSpec are used to compute the graph edit distance [15] between the current implementation and a correct implementation. This distance shows how far the current implementation is from a correct way of implementing an architectural concern or usecase. If the computed distance is greater than zero, it means that there is a deviation from a correct implementation. Therefore, ArCode provides a ranked list of recommendations on how to fix the found deviation. Each recommendation consists of a graph-based demonstration of the correct version of API usage and an automatically generated code snippet that implements that correct API usage. Detailed scientific aspects of ArCode is presented in its technical paper [16].

A. Design and Implementation

ArCode plugin is composed of three main modules which are responsible for **(i)** creating Framework API Specification (FSpec) model for a framework of interest, **(ii)** analyzing the program under development and identifying implementation deviations, and **(iii)** generating graph-based as well as code snippet recommendations. Figure 1 provides a high level overview of these modules and their interactions.

B. Features

ArCode plugin helps programmers with three main features.

1) Architectural Concern Summerization: ArCode analyzes the program under development and creates API usage model for it. Then, a graph-based representation of this model is rendered and returned to the programmer. Moreover, a translation of this model to an executable java code snippet is automatically generated and provided for more comprehension of API usages in the program.

2) Implementation Deviation Detection: After creating the API usage model of the program under development, ArCode searches through the created FSpec to identify incorrect API usages (i.e. API miuses) in the program. Then, it generates a list of change recommendations to be applied to the program. Each recommendation has a score value between 0 and 1 assigned to it. The higher the score is, the less changes needed to be applied to the program to make it a correct implementation. If score equals 1, it means that the program is following a correct implementation and the corresponding recommendation follows the same implementation. However, if there is no recommendation with score equal to 1, it means that the program deviates from a correct implementation and needs to be fixed.

3) Fix Recommendation: In case that there is a deviation from correct implementation, programmer would be able to navigate between provided recommendations. Recommendations with higher scores are most likely to be what programmer intended to implement. Please not that a higher score indicates a higher similarity between the current implementation and a recommendation. However, ArCode also provides recommendations with lower scores to give the programmer insights into what other correct API usages could be followed to make the code a correct implementation. Recommendation scores are colored green, blue, orange, and red to make programmers aware that the recommendation is very similar, somewhat similar, not similar, very different than the current implementation respectively.

C. How to use ArCode

The source code as well as released versions of ArCode plugin are available at this repository on GitHub: https://github.com/SoftwareDesignLab/ArCode-Plugin. Once the plugin is installed on Intellij IDEA, ArCode's menu will appear as a sub-menu of *tools* menu in this IDE (Figure 2). Programmer needs to fill out the fields in the *settings* menu as shown in Figure 3. That is the way to configure ArCode plugin. More details on ArCode settings could be found from ArCode plugin repository. Then, by clicking on *ArCode Runner* menu, ArCode starts analyzing the program and generating recommendations.

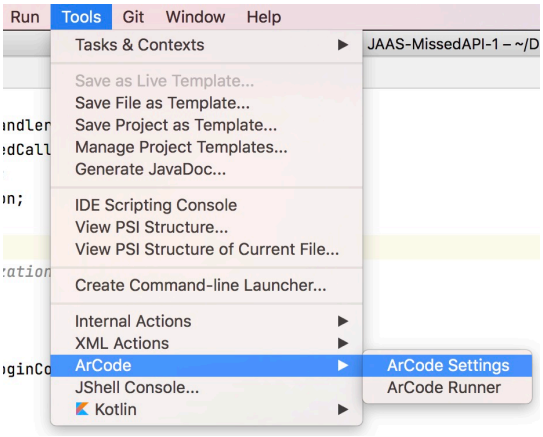


Fig. 2: ArCode main menu

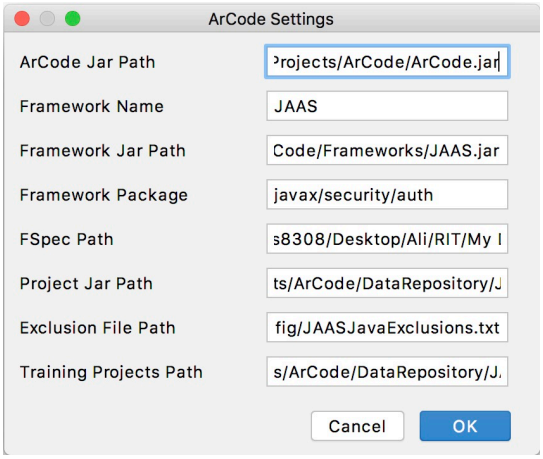


Fig. 3: ArCode Settings menu

```
1.  public class JaasUser {
2.
3.     public CallbackHandler createCallBackHandler(){
4.        return new MyCallbackHandler();
5.     }
6.
7.     public LoginContext createLoginContext( CallbackHandler callbackHandler )
8.           throws LoginException(){
9.        return new LoginContext("", callbackHandler);
10.    }
11.
12.    public void login(LoginContext loginContext) throws LoginException(){
13.       // Programmer forgets to implement this method!
14.    }
15.
16.    public void printSubject( LoginContext loginContext ){
17.       System.out.println( loginContext.getSubject() );
18.    }
19.
20.    public void logout( LoginContext loginContext ) throws LoginException(){
21.       loginContext.logout();
22.    }
23.
24.    public static void main(String[] args) {
25.       JaasUser jaasUser = new JaasUser();
26.       CallbackHandler callbackHandler = jaasUser.createCallBackHandler();
27.       try {
28.          LoginContext loginContext = jaasUser.createLoginContext(
29.             callbackHandler );
30.          jaasUser.login();
31.          jaasUser.printSubject( loginContext );
32.          jaasUser.logout();
33.       } catch (LoginException e) {
34.          e.printStackTrace();
35.       }
36.    }
37. }
```

Fig. 4: A program that intends to implement authentication tactic.

III. ArCode In Practice

To showcase how ArCode works, we start with a program in which a programmer aims to implement Authentication tactic by incorporating APIs from JAAS framework. Figure 4 shows this program. The start point of the program is main(...) method. First, an object of JaasUser is created at line 25. This is a class that programmer has developed and is expected to correctly implement authentication tactic. Except for main(...) method, JaasUser has 5 methods each of which are responsible for calling an API of JAAS framework to support the authentication process. Following lines 26 to 32, programmer first instantiates an object of LoginContext at line 9 using an instantiated object of CallbackHandler from line 4. Then, it supposed to call login() method of instantiated LoginContext at line 13. However, the programmer mistakenly has not implemented this part. Next, the Subject that supposed to be populated by calling login() method is printed in line 17. Here, since login() method is not called, then loginContext.getSubject() at line 17 will return null and so it does not print the desired output (i.e. a semantical error). Finally, calling logout() method at line 21 concludes the authentication process.

This simple program demonstrates a possible case of API misuse when a programmer aims to implement architectural concerns. As APIs of JAAS framework are called in different methods, it would be hard for a programmer to correctly track API calls and their relationships. Specifically, tracking API usages in a program becomes a challenging and an error-prone task while working on a large-scale software. ArCode supports programmers with automatically tracing and summarizing these API usages and representing them as a graph-based API usage model. It also generates a summarized version of API usages as an executable Java code snippet. These summerizations help programmer better understand how architectural concerns are currently implemented in their program. Figure 6 depicts the summerization (w.r.t. JAAS API) of the program shown in Figure 4.

Next, ArCode leverages the built Framework API Specification (FSpec) model to identify deviations from a correct implementation. In case that such a deviation is found, a list of recommendations will be generated for the programmer.

Figure 5 shows how ArCode plugin analyzes programmer's code, finds deviations from correct implementation, and provides recommendations. On the left side of this figure, a summarized version of the current implementation is shown as a graph-based API usage model as well as its related code snippet. On the right side, a possible way of fixing the found deviation is demonstrated by means of a graph-based API usage model and its corresponding code snippet. The programmer can navigate between the recommendation list through the items in the combo box. By selecting a new value from the combo box, the right panel will re-render the corresponding recommendation graph and code snippet. Values in the combo box represent the score of each recommendation. Scores are float numbers between 0 and 1. Recommendations

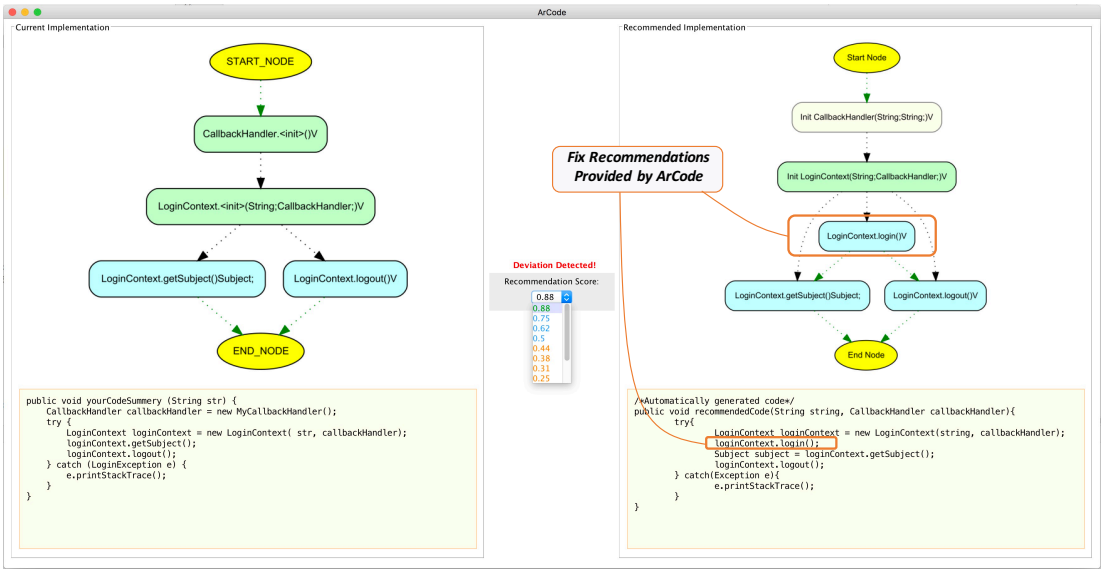
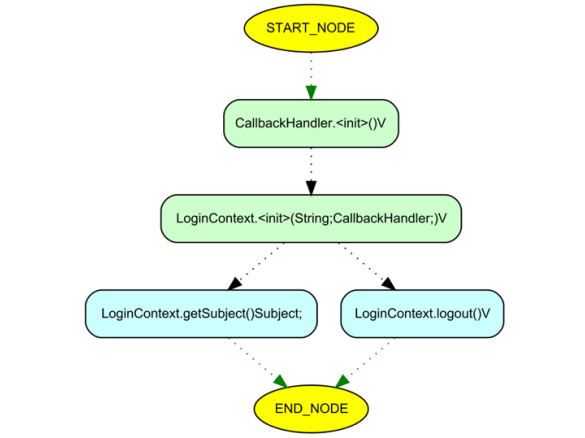


Fig. 5: Receiving recommendations from ArCode plugin in IntelliJ IDEA




```
public void programCodeSummery(String str) {
    CallbackHandler callbackHandler = new MyCallbackHandler();
    try {
        LoginContext loginContext = new LoginContext(str, callbackHandler);
        loginContext.getSubject();
        loginContext.logout();
    } catch (LoginException e) {
        e.printStackTrace();
    }
}
```

Fig. 6: Summerized version of code snippet shown in Figure 4 as a graph-based API usage model and a code snippet

with scores closer to 1 have more similarities to the current implementation. For programmers' convenient, scores in the range of [0.8, 1], [0.5, 0.8), [0.2, 0.5), and [0, 0.2) are colored as *green*, *blue*, *orange*, and *red* respectively. For instance, as shown in Figure 5, ArCode identifies that the programmer's current implementation deviates from a correct authentication implementation. Therefore, it provides a list of recommendations as a guidance on how to fix this problem. The highest score in the combo box of Figure 5 is 0.88. It means that many parts of the program complies with a correct API usage. Nevertheless, there are some API misuses needed to be fixed. For instance, the recommended graph-based API usage model has an additional node, LoginContext.login(), compared to API usage model of the current implementation. Black dotted arrows represent data-dependencies between APIs. It means that programmer needs to call login() method of the instantiated LoginContext in the code. Green dotted arrows represent API call sequence constraints. It means that calling login() method must be placed in the code before the location that either logout() or getSubject() methods are called. Furthermore, on the lower right side of Figure 5 an executable code snippet is displayed. Programmer can follow the code snippet to find a more comprehensive insight into the problem and its related fix recommendation.

IV. Conclusion & Future Work

In this paper we introduced ArCode plugin, which is a tool that helps programmers achieve a better understanding of the current state of architectural concerns implementation in their program. It also facilitates finding and fixing implementation deviations. Although this tool provides very insightful fix recommendations, it is the programmer's responsibility to apply the suggested changes to correct the implementation. There might be a few interactions between the programmer and ArCode to make sure that the final code complies with a correct API usage.

As of our future work, we aim to automate the process of performing fix recommendations in the program under development. In that regard, based on the selected recommendation, ArCode will automatically change the program. Then, it will re-analyze the code, re-generate program's API usage model and code summery, and populates a new list of recommendations. Also, future work direction includes automatically synthesizing the code from scratch based on the selected API usage by the programmer.

ACKNOWLEDGMENTS

This work was partially funded by the US National Science Foundation (NSF) under grant number CCF-1943300, CNS-1816845 and CNS-1823246.

REFERENCES

[1] A. Vihavainen, J. Helminen, and P. Ihantola, "How novices tackle their first lines of code in an ide: Analysis of programming session traces," in *Proceedings of the 14th Koli Calling International Conference on Computing Education Research*, 2014, pp. 109–116.

[2] L. Wu, F. Li, Y. Wu, and T. Zheng, "Ggf: A graph-based method for programming language syntax error correction," in *Proceedings of the 28th International Conference on Program Comprehension*, 2020, pp. 139–148.

[3] I. IntelliJ, "the most intelligent java ide," *JetBrains [online].[cit. 2016-02-23]. Dostupné z: https://www. jetbrains. com/idea/# chooseYourEdition*, 2011.

[4] I. Zayour and A. Hamdar, "A qualitative study on debugging under an enterprise ide," *Information and Software Technology*, vol. 70, pp. 130–139, 2016.

[5] F. Liu, G. Li, B. Wei, X. Xia, Z. Fu, and Z. Jin, "A self-attentional neural architecture for code completion with multi-task learning," in *Proceedings of the 28th International Conference on Program Comprehension*, 2020, pp. 37–47.

[6] I. Schröter, J. Krüger, J. Siegmund, and T. Leich, "Comprehending studies on program comprehension," in *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. IEEE, 2017, pp. 308–311.

[7] X. Xia, L. Bao, D. Lo, Z. Xing, A. E. Hassan, and S. Li, "Measuring program comprehension: A large-scale field study with professionals," *IEEE Transactions on Software Engineering*, vol. 44, no. 10, pp. 951–976, 2017.

[8] S. Stapleton, Y. Gambhir, A. LeClair, Z. Eberhart, W. Weimer, K. Leach, and Y. Huang, "A human study of comprehension and code summarization," in *Proceedings of the 28th International Conference on Program Comprehension*, 2020, pp. 2–13.

[9] M. Dias, D. Orellana, S. Vidal, L. Merino, and A. Bergel, "Evaluating a visual approach for understanding javascript source code," in *Proceedings of the 28th International Conference on Program Comprehension*, 2020, pp. 128–138.

[10] M. Galster and S. Angelov, "What makes teaching software architecture difficult?" in *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 2016, pp. 356–359.

[11] N. B. Harrison and P. Avgeriou, "Leveraging architecture patterns to satisfy quality attributes," in *European conference on software architecture*. Springer, 2007, pp. 263–270.

[12] M. P. Robillard and R. Deline, "A field study of api learning obstacles," *Empirical Software Engineering*, vol. 16, no. 6, 2011.

[13] S. Baltes and S. Diehl, "Usage and attribution of stack overflow code snippets in github projects," *Empirical Software Engineering*, vol. 24, no. 3, pp. 1259–1295, 2019.

[14] F. Fischer, K. Böttinger, H. Xiao, C. Stransky, Y. Acar, M. Backes, and S. Fahl, "Stack overflow considered harmful? the impact of copy&paste on android application security," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 121–136.

[15] H. Bunke, "On a relation between graph edit distance and maximum common subgraph," *Pattern Recognition Letters*, vol. 18, no. 8, pp. 689–694, 1997.

[16] A. Shokri, J. C. S. Santos, and M. Mirakhorli, "ArCode: Facilitating the Use of Application Frameworks to Implement Tactics and Patterns," *arXiv e-prints*, p. arXiv:2102.08372, Feb. 2021.