Towards Generating Templates of Method Body Based on Method Name and Related Identifiers

Yuya Onizuka* Yasuhiro Hayase† Tetsuo Yamamoto‡
Yuki Kashiwabara* Takashi Ishio* Katsuro Inoue*

*Graduate School of Information Science and Technology, Osaka University {y-onizuk,k-yuki,ishio,inoue}@ist.osaka-u.ac.jp
†Faculty of Engineering, Information and Systems, University of Tsukuba hayase@cs.tsukuba.ac.jp
‡College of Engineering, Department of Computer Science, Nihon University tetsuo@cs.ce.nihon-u.ac.jp

Abstract

In modern software development, developers have to select and combine appropriate APIs from software libraries to implement any features. This paper proposes an approach that takes as input a method name which a developer is attempting to create, and suggests APIs that are likely used as a template of method body. By using the template as a reference and/or editing the template, the developer can write the method body. Our approach generates templates from association rules that associate APIs with identifiers such as method names, class names, and field types/names included in a large set of source files.

Categories and Subject Descriptors D.2.6 [*Software Engineering*]: Programming Environments

Keywords API, Code completion, Association rule mining

1. Introduction

In modern software development, developers have to select and combine appropriate APIs from software libraries to implement features. However, learning APIs is difficult and time consuming because of numerous APIs and their combinations.

Some approaches which suggest candidates of method body from the context of the source code under editing are proposed to develop software efficiently [2–4]. They suggest code snippets by leveraging the context of the method body, therefore, they can suggest better candidates than code completion of standard IDEs. However, they cannot suggest appropriate candidates without a part of a method body.

In this paper, we propose to suggest sets of APIs as templates of a method to developers attempting to create a new method. The approach suggests APIs which developers probably write in the method body after they decide the method name. The idea which suggests templates of a method body is implemented as one of the code completions in Eclipse [1]. Eclipse suggests creating method *getFoo* and *setFoo* in a class which has field *foo*, because developers are likely to create a getter/setter to get the value or set a value in a class declared a field. We generalize the idea for other kinds of methods so that IDE can suggest templates of method body to developers. APIs which developers are likely used in method body are learned by association rule mining. We associate APIs in each method with its method name and identifiers around the method such as class names, field names and field types.

2. Approach

This paper proposes an approach which suggests templates of method body. Developers complete the method body by using the template as a reference and editing it. Our approach focuses on relations between a method name and the method body to generate templates of method name.

A method usually has a short and self-describing name, whose *verb* and *object* respectively express the operation and its target performed in the method. In some cases, a method name comprises only a verb, whose object is implicitly indicated by identifiers around the method, e.g. the class name or fields.

Ideally, it is expected that generated templates of method body are complete source code, however, generating complete method bodies to general methods is very difficult, since method bodies have variable definitions, method calls, and complex control structures.

In this paper, a template of a method body is a set of methods which are likely called in the method. Our approach helps developers to choose APIs from a lot of APIs; developers can look up various documents that explain the usage of the APIs included in a template.

Templates of method body are generated from association rules which association rule mining find from a large set of source files. Association rule mining use method names, methods called by the methods, and identifiers around the methods such as class names and field types/names. Method names are divided into verb and object because the verb of a method name represents the behavior of the method.

One example is a Data Access Object (DAO) class. Classes using databases have connection fields and is defined a method which executes SQL statements. The method is named *updateUserInfo* and so forth and processed the following steps: 1) create a SQL statement (the method name is *createStatement*), 2) execute the SQL statement (*executeUpdate*), 3) close the SQL statement (*close*). It expects that association rules found from source files using a database have a pattern which methods called by the method are *createStatement*, *executeUpdate*, and *close* when the field type is *Connection* and the verb of the method is *update*. Our approach suggests a set of methods {*createStatement*, *executeUpdate*, *close*}

AOAsia'13, March 25, 2013, Fukuoka, Japan.
Copyright © 2013 ACM 978-1-4503-1861-7/13/03...$15.00

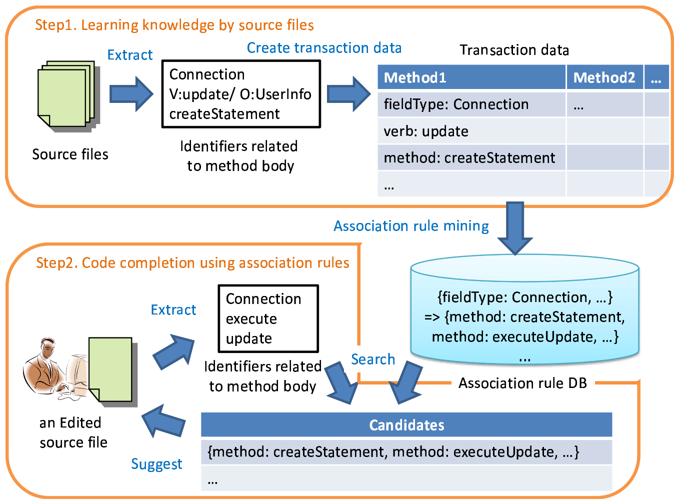


Figure 1. AN OVERVIEW OF THE IMPLEMENTATION OF OUR APPROACH

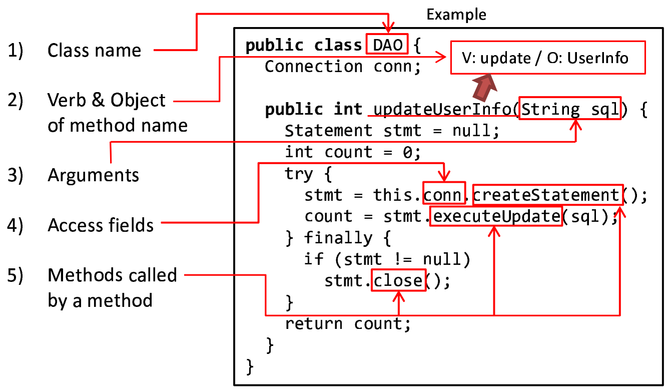
by using this pattern when a developer creates a new method the verb of which is *update* in a class defined a *Connection* field.

3. Implementation

An overview of the implementation of our approach is shown in Figure 1. The implementation consists of two steps: 1) learning knowledge by source files, and 2) code completion using association rule DB. Step 1 is performed in advance of Step 2.

Step 1 creates a database that contains support, confidence, and association rules that each antecedent is a set of identifiers of source files and each consequent is a set of methods called by a method. In terms of association rule mining, a transaction is a set of identifiers related to a method. For each method in source files (a training data set), we extract the verb of the method name, the object of the method name and related identifiers including the class name, arguments of the method, fields accessed by the methods and method names called in the method as shown in Figure 2. Each name is associated with its origin; for example, the class name “DAO” is represented as an item “class:DAO” in a transaction. After all methods are translated into transactions, we execute association rule mining to extract association rules whose consequent is a set of methods called in method bodies. For example, a rule $\{$verb:update, fieldType:Connection$\} \Rightarrow \{$method:createStatement, method:executeUpdate, method:close$\}$ is extracted from source files. This rule indicates that a method likely calls a set of methods *createStatement*, *executeUpdate* and *close* if the verb of the method is *update* and the method can access a field whose type is *Connection*.

Step 2 suggests templates of method body as sets of method calls to a developer just after input a method name. We extract verbs, objects, and related identifiers from a source file edited by a developer, and then look for association rules whose antecedents are included in the source file. Finally, consequents of searched rules are suggested as templates of method body to the developer. The templates are sorted or narrowed down by support, confidence, and so on. The developer chooses a template and inserts it in the source file. The developer writes the method body by using the template as a reference and/or editing the template.



```
public class DAO {
  Connection conn;

  public int updateUserInfo(String sql) {
    Statement stmt = null;
    int count = 0;
    try {
      stmt = this.conn.createStatement();
      count = stmt.executeUpdate(sql);
    } finally {
      if (stmt != null)
        stmt.close();
    }
    return count;
  }
}
```

Figure 2. VERB AND OBJECT OF METHOD NAME AND IDENTIFIERS RELATED TO METHOD BODY

4. Current Status and Future Work

Currently, association rules are obtained from a small set of source files, and we are evaluating the rules. We are going to analyze the results and create association rule DB from a larger set of source files. In parallel, we are implementing our approach as an eclipse plugin.

We are planning two evaluation experiments. One experiment evaluates the usefulness of suggested templates. In the experiment, we compare templates of a method body generated from methods in source files by our approach with the original method body to see that they match. The other experiment asks subjective opinions. Some developers worked on the tasks and answers questionnaire about generated templates and usability of the tool.

Even though a method can have two or more functions, its name tends to expresses only one of these functions. Those multi-function methods could obstruct the knowledge extraction from source file set, since method calls implementing a function are bound for a method name expressing a different function. Aspect-Oriented programming techniques can improve the effectinveness of our approach by improving cohesion of multi-function methods.

We are also planning to provide not a set of method calls but different type of data to a developer. For example, developer can obtain the appropriate order of API usage if a sequence of method calls is used as a method template.

Acknowledgments

This research was supported by KAKENHI No.21700031, No.21240002, and No.23680001.

References

[1] Eclipse. http://www.eclipse.org/.

[2] M. Bruch, M. Monperrus, and M. Mezini. Learning from examples to improve code completion systems. In *Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM Symposium on the Foundations of Software Engineering*, 2009.

[3] A. T. Nguyen, T. T. Nguyen, H. A. Nguyen, A. Tamrawi, H. V. Nguyen, J. Al-Kofahi, and T. N. Nguyen. Graph-based pattern-oriented, context-sensitive source code completion. In *Proceedings of the 2012 International Conference on Software Engineering*, pages 69–79, 2012.

[4] Y. Tetsuo, Y. Norihiro, and H. Yoshiki. Seamless souce code reuse using source code corpus (in japanese). *IPSJ Journal*, 53(2):644–652, feb 2012.