Integrating Context-Free Code Examples into Local Code Context

Xi Ge and Emerson Murphy-Hill
Department of Computer Science, NC State University
Raleigh, NC, USA
xge@ncsu.edu emerson@csc.ncsu.edu

***Abstract*—Code examples effectively communicate the proper usage of software libraries. More than following code examples, developers may directly integrate them into their local codebase. However, the integration usually requires updating part of a given code example, which activity is tedious and time-consuming. Existing techniques that automate integration always assume that the code example comes from an available context. However, due to the increasing popularity of online forums such as Stack Overflow, developers often share short code snippets without any specific context, leading to the lower usefulness of the existing techniques. To address this problem, we propose a novel approach called SmartCopy that automatically generates a list of updated code examples as integration candidates; ranks the candidates by using a heuristic-based algorithm; and finally asks the developer to select one of the candidates to be integrated into her codebase. To evaluate SmartCopy. . .**

I. Introduction

When developing software, developers frequently use available libraries to save the effort of implementing every feature from scratch. To correctly interact with these libraries, developers need to spend effort in learning the usage of the APIs associated with them. However, the learning of API usage is both time-consuming and labor-intensive, partially because libraries are designed for generalized purposes, however the developer intends to leverage them to implement a specific feature. Therefore, to bridge this gap, the developer needs to sufficiently understand libraries before she can use them correctly; failing to do so may lead to disastrous outcomes [1].

Developers learn API usage through multiple channels such as library documentation, online forums, and code examples. Among them, code examples are essential for API learning because of three reasons: (1) comparing to explanatory text, code examples are self-explaining and concise; (2) aiming at demonstrating the best practice, code examples are safe to follow [2]; and (3) in the form of source code themselves, code examples can be easily integrated to the software under developers' implementation [3].

As summarized by Holmes and colleagues, reusing code examples takes three independent steps: locating, selecting and integrating [4]. For each step, researchers proposed multiple techniques to assist developers with. For instance, Thummalapenta and Tao proposed a tool called PARSEWeb that automatically locates online code examples to recommend API call sequences [5]; Cottrell and colleagues presented Guido that visually presents the difference between two code examples to help developers select the desirable one [6]; also proposed by Cottrell and colleagues, Jigsaw exploits the contextual similarity between the codebase containing the desired functionality and the codebase in the need of the functionality to semi-automate the integration of the desired functionality into the latter [7].

Although the existing semi-automated integrating techniques demonstrate their effectiveness, one common assumption of these techniques is the availability of the rich context from where the code examples can be borrowed [7]. However, the assumption does not hold if developers copy code examples from online forums, where code examples are usually short snippets without any context. Given the increasing popularity of websites like Stack Overflow, more code examples have to be integrated manually [8], [9]. Therefore, to address this problem, the present paper investigates the semi-automation of context-free code examples into a developer's local codebase. In summary, we make the following contributions:

- A heuristic-based technique called SmartCopy that helps a developer integrate code examples copied from context-free sources, such as online forums, into the local codebase under implementation.
- An open source implementation of SmartCopy as a plugin to the Eclipse IDE.
- A filed study of the heuristics adopted by SmartCopy. Our study suggests that . . .

II. Motivation

The section motivates the need of automatic integration through a real-world example. Suppose Jane is a Java developer adopting the process of test-driven development. On day, when Jane is implementing a module for file interactions, she needs to fill the skeleton illustrated in Code 6, which method is supposed to read a given input file and to return all text in the file. However, Jane is insufficiently familiar with the Java APIs to implement this feature. Thus, she asked this question on Stack Overflow. Shortly, another developer responded her with a short code example illustrated in Code 2. After reading the response, Jane decided to integrate the code example into the skeleton; hence, she copied and pasted Code 2 to Code 6, leading to the method illustrated in Code 3.

However, Code 3 still needs Jane's manual updates to properly function. First, Jane needs to rename the input of the copied snippet, which is a file instance called `file`, to the name of the method parameter, which is `inputfile`. Second, the copied snippet uses the library of `java.io`; thus Jane needs to import the library. Finally, the output after reading file,

```java
public String readFile(File inputFile) {
        return "";
}
```

Code 1: Jane's skeleton.

```java
FileReader reader = new FileReader(file);
char[] chars = new char[(int) file.length()];
reader.read(chars);
content = new String(chars);
reader.close();
```

Code 2: Code example from Stack Overflow.

which is stored in `chars`, needs to be returned from the method instead of an empty string.

Existing techniques that automatically integrate reusable code snippets can provide little help in Jane's case, because they always assume that the reusable code snippet is from an available larger context such as another source package or project [6], [10], [11]. Since Jane copied the code snippet from Stack Overflow where the original context of the snippet is inaccessible, existing techniques cannot learn the proper way of adapting the code snippet from the missing context. To assist developers like Jane to integrate code examples regardless of the availability of their original contexts, this paper presents a tool called SmartCopy.

Approach

SmartCopy is a plug-in to the Eclipse integrated development environment(IDE), which helps in code integration, if the copied(and pasted) code does not match the context of the local code under implementation. Generally, a developer copies a snippet from Q&A websites like StackOverflow etc. and pastes it to his/her own source code. From this point onwards the local code under implementation, which now has code snippet pasted to it, would be known as raw code. It should be noted that at this stage the code snippet has not been integrated. SmartCopy now takes the raw code as input and acts on it. The tool stands on the shoulders of two algorithms(giants) namely

- *PROCESS algorithm*: The algorithm takes raw code as an input and returns a list of variables, which needs to be modified for successful integration of the copied code. It should be noted that for each undeclared variable in the list there is an associated unranked list of possible replacement candidates(for example declared variable names). This information can then be used to integrate code easily. The algorithm is further explained in subsection A
- *SELECTION algorithms*: The unranked list of possible replacement candidates returned by *PROCESS algorithm* does not help the developer to easily integrate the code snippet. This is because the list does not consider the context of the code. *SELECTION algorithms* are a set of heuristic algorithms, which leverage the knowledge of common coding practices to rank the list of possible replacement candidates. The algorithm is explained in further details in subsection B

```java
public String readFile(File inputFile) {
        FileReader reader = new FileReader(file);
        char[] chars = new char[(int)
            file.length()];
        reader.read(chars);
        content = new String(chars);
        reader.close();
        return "";
}
```

Code 3: Jane's skeleton after pasting.

A. Process Algorithm

The algorithm as mentioned in above section takes the raw code as input and returns a list of variables, along with an unranked list of possible replacement candidates. The algorithm can be further divided into following subroutines:

1) huntVariables(): The subroutine finds all the undeclared variables and types present in the raw code.
2) apiResolution(): The raw code is passed to the JavaBaker tool to resolve the API types. The Baker tool can be best defined as "a constraint-based, iterative approach for determining the fully qualified names of code elements in source code snippets" [12]. The subroutine returns a list of possible APIs related to each code element (from now called as API signatures).
3) typeResolution(): The subroutine uses the raw code and the result from apiResolution() to perform the type resolution of the undeclared variables present in the raw code.
4) findReplacementVariables(): The subroutine uses the raw code and the list of undeclared variables(output from huntVariables()) and returns a list of possible declared variables which can be used as a replacement for the undeclared variables. The code is traversed to look for all the declared variables. As the traversal takes place, the subroutine tracks the scope of all the variables that can used. This is to ensure that the mapping between variables follows the scoping rules of Java. Once declared variables within the scope(of undeclared variables) are collected, the list of replacement variables is returned.
5) addImportStatements(): The subroutine uses the output of apiResolution() to find the packages that need to be imported.
6) addReturnStatements(): The subroutine takes raw code as input and checks if all the methods declared in the code have a return statement. If a return statement is missing, a return statement with appropriate type of variable is added.
7) returnHints(): The subroutine takes the raw code as input and checks if any of the variables has a constant declaration. For example if the code snippet(pasted from Q&A sites) has statements like

```java
Connection con =
    DriverManager.getConnection
    ("jdbc:mysql:///database","root","");
```

the strings "jdbc:mysql:///database", "root" needs to be replaced with correct values. The subroutine returns a list of constant declarations which require attention of developer.

Results from all the subroutines are collected and assimilated into a list. The list contains undeclared variables, import statements, possible replacement candidates(result of findReplacementVariables()) etc required for the successful integration of the code snippet.

B. *SELECTION algorithms*

At this stage (i.e. after the *PROCESS algorithm* we have a list of undeclared variables with its types resolved along with possible candidates. The *SELECTION algorithms* are then use to rank the list of possible candidates so that the candidate variable which is ranked the highest would have the greatest likelihood of being the correct replacement for the undeclared variable. To find the correct ranking mechanism, we investigate a number of heuristic techniques proposed in existing literature to find the algorithm best suited for our purpose. The various metrics studied in this work are

1) Navigation Distance: Navigation distance is the number of lines between declaration of the variable(from the list of declared variables) and the first use of the undeclared variable. The rule of thumb while programming is to keep the scope of a variable as tight as possible. Following this rule programmers can make their code readable and reduce the chances of name collision. Simply put, while writing a method, thedeveloper will frequently use local variables (within the scope of the method) as opposed to a field declared in the same class. We assume that the programmer is experienced enough to follow the basic rules of programming.

Readers of programs have primarily two sources of domain information; comments and well formed identifier names. It cannot be stressed enough how there needs to be a mapping between the concept and the identifier names. The metrics are:

2) String edit Distance: String edit distance is considered to be a good metric to find relationships between two entities when little is known about them [13]. Distance function is a mapping of two entities to a real number which shows how similar the words are to each other. We use a variety of distance functions to find which function is useful for our purpose. The premise of this approach is that programmers use only a subset of the domain specific names while describing a concept. This means that most of the programmers use the same variable name to represent the same concept. This is the reason why we decided to use string edit distance as one of the ranking mechanisms.
3) Semantic Distance: Semantic distance means how two terms are related based on the likeness of meaning. For eg. semantic distance between fileName and file would smaller than the semantic distance between fileName and urlName. Antoniol et al had pointed out that "the application-domain knowledge that programmers process when writing the code is often captured by the mnemonics for identifiers;therefore, the analysis of these mnemonics can help to associate high-level concepts with program concepts and vice-versa". This implies that both the skeletal code and the raw code would have variable names associated to the high-level concepts. For ex. file name related variables would always have 'file' as a sub-string. Hence in our tool we leverage this concept and try to find similarity between the declared variable names and the undeclared variable names.
4) Type Distance: In Java, as in most of other object oriented languages, classes are arranged in a hierarchy and the subclass inherits behavior from its superclass. This means that class hierarchy can be thought as a tree with *class Object* as its root. So drawing from the tree terminology, we can define type distance as number of hops needed to get from a node A to node B, given that A is ancestor of B or vice versa. For example. typeDistance(java.lang.StringBuilder, java.lang.AbstractStringBuilder) is 1.

III. RUNNING EXAMPLE

```
import java.sql.Connection;

public class ConnectionProvider {
        String hostName;
        String userName;
        String userPassword;
        public static Connection getConnection()
            throws SQLException{
                Connection conn = null;

                return conn;
        }
}
```

Code 4: Harry's skeleton.

As an example we show the step by step integration done by SmartCopy. For the simplicity we do not include all subroutines discussed in the approach section.

Developer Harry wants to write a method, which would return a *java.sql.Connection* object. So, he started sketching a method getConnection as shown in Code 4 which would return an object of type java.sql.Connection. He declares conn (a *java.sql.Connection* object) and adds a return statement as well. Since Harry doesn't remember the syntax of DriverManager, he looks for answers in stackoverflow. There he found a code snippet, as shown in Code 5, which did exactly what he wants his method to do.

```
        if(cn == null){
            String driver = "com.mysql.jdbc.Driver";
            Class.forName(driver);
            dbHost = "jdbc:mysql://"+dbHost;
            cn = DriverManager.getConnection(
                dbHost, dbUser, dbPassword );
        }
```

Code 5: Code Snippet from StackOverflow.com

At this point Harry copies the code snippet and pastes it to his skeleton code. Harry's raw code now looks like Code 6.

```
import java.sql.Connection;

public class ConnectionProvider {
        String hostName;
        String userName;
        String userPassword;
        public static Connection getConnection()
            throws SQLException{
                Connection conn = null;
                if(cn == null){
                    String driver =
                        "com.mysql.jdbc.Driver";
                    Class.forName(driver);
                    dbHost = "jdbc:mysql://"+dbHost;
                    dbUser = "user";
                    cn =
                        DriverManager.getConnection(
                        dbHost, dbUser, dbPassword
                        );
                }
        }
}
```

Code 6: Input to SmartCopy

PROCESS Algorithm:

- SmartCopy first finds out all the undeclared variables and types using *huntVariables()*. The undeclared variables in the skeleton code are [cn, dbHost, dbUser, dbPassword and DriverManager]. (*DriverManager* is a package which needs to be imported to get the program compilable).
- Then the skeleton code is sent to JavaBaker using the subroutine *apiResolution()*. JavaBaker is a tool which uniquely identifies type references, method calls and field references in a code snippet. The code snippet is sent to the JavaBaker tool using HTTP POST and get a JSON response as the response. JavaBaker returns all the possible libraries. Our tool only considers API elements which are exact matches. Subramaniam et al has noticed that in the Android API there are only 6720 unique method calls out of 24,545 total method declarations. This means that given a partially qualified name it is very difficult to find the exact match for the method or field identifiers. Since most of the method and field identifiers found in code snippets are partially qualified, it is normal to have conflicts. This is the reason SmartCopy only uses results whose cardinality is 1.
- Once all the method calls have been identified, this information can then be used by *typeResolution()* to find the types of the various undeclared variables. For example in Code 6 all the undeclared variables can be resolved using the JavaBaker tool in the following way:
  - JavaBaker returns the fully qualified name(FQN) of getConnection() as *java.sql.DriverManager.getConnection()*. The FQN can then used to find the type of variable cn which is *java.sql.Connection*.
  - dbHost and dbUser can be resolved to java.lang.String. Signature of *getConnection()* is *getConnection(String url, String user, String password)* which implies *dbPassword* is of the type *java.lang.String*.

After executing the PROCESS algorithm on Harry's raw code, we find out the types of all the undeclared variables and a list of possible candidates for replacement. The resolved types of the undeclared variables are listed in Table I and possible candidates are listed in Table II.

Variable Name	Resolved Type
cn	java.sql.Connection
dbHost	java.lang.String
dbUser	java.lang.String
dbPassword	java.lang.String
DriverManager	java.sql.DriverManager

TABLE I: Resolved Types

Variable Name	Possible Candidates
cn	conn
dbHost	hostName, userName, userPassword, driver
dbUser	hostName, userName, userPassword, driver
dbPassword	hostName, userName, userPassword, driver
DriverManager	java.sql.DriverManager DriverManager

TABLE II: Possible Candidates for Replacement

SELECTION Algorithm:

- Navigation Distance: ...
- String edit Distance: ...
- Semantic Distance: ...
- Type Distance: ...

IV. Related Work

Question & Answer websites such as Stack Overflow are important channels for developers to share knowledge. Researchers conducted multiple studies to investigate how developers use these channels. For instance, Nasehi and colleagues collected high-quality code examples on Stack Overflow and summarized nine attributes of them [14]. Parnin and colleagues studied the documentation generated by Stack Overflow users and found that the crowd documentation is both comprehensive and useful [9]. Treude and colleagues studied why certain questions on Stack Overflow get answered while others do not; their preliminary results suggest that Stack Overflow is particularly effective at code review and conceptual questions [8].

Developers frequently leverage existing code examples to help implement their own software. Therefore, researchers studied the usage of code examples as well as prosed novel techniques to assist this activity. For instance, Holmes and colleagues proposed a tool called Strathcona that recommends structurally similar code examples with the code under development [15]. Proposed by Mandelin and colleagues, PROSPECTOR recommends code snippets that assist developers to get an object of the desired type [16]. Cottrell and colleagues proposed a tool called Guido that visually discerns various code examples to help developers choose the right one [**?**].

Another category of recommender systems explore online resources to help local code development. For instance, Thummalapenta and Tao proposed PARSEWeb to recommend a API call sequence to derive the destination object type from a source object type [5]. Kononenko and colleagues proposed a tool called Dora to recommend online posts that are contextually relevant to the developer's problem [17].

To save developers' effort in searching relevant code examples, developers proposed multiple recommender systems. For instance,

Acknowledgment

This research was conducted during the first author's internship at ABB Research. We thank the participants for helping us conduct the experiments. We thank the help from Vinay Augustine, Patrick Francis, and Will Snipes. We also thank the comments from the Development Liberation Front group members Titus Barik, Michael Bazik, Brittany Johnson, Kevin Lubick, John Majikes, Yoonki Song and Jim Witschey [18].

References

[1] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Computation Surveys*, vol. 44, no. 2, pp. 6:1–6:42, 2008.

[2] M. P. Robillard and R. Deline, "A field study of api learning obstacles," *Empirical Software Engineering*, vol. 16, no. 6, pp. 703–732, 2011.

[3] M. B. Rosson and J. M. Carroll, "The reuse of uses in smalltalk programming," *ACM Transaction of Computer-Human Interaction*, vol. 3, no. 3, pp. 219–253, 1996.

[4] R. Holmes, R. Cottrell, R. Walker, and J. Denzinger, "The end-to-end use of source code examples: An exploratory study," in *Proceedings of the International Conference on Software Maintenance*, 2009, pp. 555–558.

[5] S. Thummalapenta and T. Xie, "Parseweb: A programmer assistant for reusing open source code on the web," in *Proceedings of the International Conference on Automated Software Engineering*, 2007, pp. 204–213.

[6] R. Cottrell, R. J. Walker, and J. Denzinger, "Semi-automating small-scale source code reuse via structural correspondence," in *Proceedings of the International Symposium on Foundations of Software Engineering*, 2008, pp. 214–225.

[7] R. Cottrell, B. Goyette, R. Holmes, R. Walker, and J. Denzinger, "Compare and contrast: Visual exploration of source code examples," in *Proceeding of the International Workshop on Visualizing Software for Understanding and Analysis*, 2009, pp. 29–32.

[8] C. Treude, O. Barzilay, and M. Storey, "How do programmers ask and answer questions on the web? (nier track)," in *Proceedings of the International Conference on Software Engineering*, 2011, pp. 804–807.

[9] C. Parnin and C. Treude, "Measuring api documentation on the web," in *Proceedings of the International Workshop on Web 2.0 for Software Engineering*, 2011, pp. 25–30.

[10] D. Hou, F. Jacob, and P. Jablonski, "Exploring the design space of proactive tool support for copy-and-paste programming," in *Proceedings of the Conference of the Center for Advanced Studies on Collaborative Research*, 2009, pp. 188–202.

[11] R. Holmes and R. J. Walker, "Systematizing pragmatic software reuse," *ACM Transaction on Software Engineering Methodology*, vol. 21, no. 4, pp. 20:1–20:44, 2013.

[12] S. Subramanian, L. Inozemtseva, and R. Holmes, "Live api documentation." in *ICSE*, 2014, pp. 643–652.

[13] W. Cohen, P. Ravikumar, and S. Fienberg, "A comparison of string metrics for matching names and records," in *KDD Workshop on Data Cleaning and Object Consolidation*, vol. 3, 2003, pp. 73–78.

[14] S. Nasehi, J. Sillito, F. Maurer, and C. Burns, "What makes a good code example?: A study of programming q amp;a in stackoverflow," in *Proceedings of the International Conference on Software Maintenance*, Sept 2012, pp. 25–34.

[15] R. Holmes, R. J. Walker, and G. C. Murphy, "Strathcona example recommendation tool," in *Proceedings of the International Symposium on Foundations of Software Engineering*, 2005, pp. 237–240.

[16] D. Mandelin, L. Xu, R. Bodík, and D. Kimelman, "Jungloid mining: Helping to navigate the api jungle," in *Proceedings of the Conference on Programming Language Design and Implementation*, 2005, pp. 48–61.

[17] O. Kononenko, D. Dietrich, R. Sharma, and R. Holmes, "Automatically locating relevant programming help online," in *Proceedings of the Symposium on Visual Languages and Human-Centric Computing*, 2012, pp. 127–134.

[18] A. Bacchelli, L. Ponzanelli, and M. Lanza, "Harnessing stack overflow for the ide," in *Proceedings of the International Workshop on Recommendation Systems for Software Engineering*, June 2012, pp. 26–30.