A Case Study of Calculation of Source Code Module Importance

Takaaki Goto[1,a)] Setsuo Yamada[2,b)] Tetsuro Nishino[1,c)] Kensei Tsuchida[†1,d)]

Abstract: Open Source Software (OSS) has been widely used in software development. However, OSS often lacks adequate documentation and as a result developers cannot obtain enough information to develop software with OSS. The popularization of OSS, or free software, has enabled developers to more easily obtain source codes to improve their coding skills. Developers read source codes in order to understand the architecture and behavior of OSS. However, it is difficult for developers to find the information they need due to the lack of documentation .
In this paper, we propose a method for calculating the importance of source code modules based on the dependency of the source codes. We target Java language, and we calculate the degrees from class dependency.

Keywords: open source software, software maintenance, source code reading

1. Introduction

As a result of the short-term nature of software development and the creation of large software programs, creating and managing many software documents poses problems. In recent years, software development using Open Source Software (OSS) has increased; however, developers find it difficult to obtain information about OSS due to the lack of documentation. Therefore, developers have to read OSS source code that offers them less information.

On the other hand, developers have found it easier to obtain source codes with the popularization of OSS or free software. They use this information to either revise the OSS or improve their coding skills. However, it is difficult for developers to determine the order of reading source codes; and this is especially true for primary developers.

Many research studies targeting source code summarization [1], [2], [3] and source code mining [4], [5], [6] have been done. Source code summarization helps developers to quickly understand software. However, methods for reading source code modules are also important.

In this paper, we propose a method to calculate the importance of source code modules, using the dependency of source codes. Here we target Java programming, and calculate the importance degree using class dependency.

[1] Graduate School of Informatics and Engineering, The University of Electro-Communications, Chofu, Tokyo, Japan
[2] Nippon Telegraph and Telephone Corporation, Japan
[†1] Presently with Faculty of Information Science and Arts, Toyo University, Kawagoe, Saitama, Japan
[a)] gototakaaki@uec.ac.jp
[b)] yamada.setsuo@lab.ntt.co.jp
[c)] nishino@uec.ac.jp
[d)] kensei@toyo.jp

2. Source Code Reading

Developers develop various methods to read source codes. Generalized methods for source code reading are as follows [7]:

(1) First, developers find the focus point for reading by keyword searching, and then start reading the source code from the located point.
(2) Developers start reading from the main function or the main method.
(3) Developers use a debugger when they read source code. They set break points to focus areas, after which the debugger is executed. Developers can read source code with various debugging information such as a variable's value.

In addition to the methods listed above, it has been shown to be effective for developers to read the source code module, which is primarily obtained from other modules. Frequently accessed source code may have an important function. As such, it is considered to be an effective method for reading and understanding source code. Developers can detect the reading points from the dependency of the source codes.

3. Class Dependency

In the Java program, each part of the source code refers to various classes. For instance, one class uses another class. If class A uses class B, a relationship between class A and class B is established, which is identified as "class A refers to class B". Figure 1 illustrates the relationship.

In our method, dependency is computed by using other classes in one class. However we do not consider usage frequency. For example, if class A uses class B many times in one class, we only count one relation between class A and class B.

Because some class dependencies arise from these referring relationships among classes, developers can understand the links in

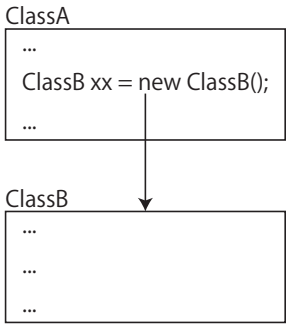

Fig. 1 Class dependency relationship: “Class A refers to Class B”

the source code.

Some open source software have functions that enable developers to analyze class dependency. “ispace” [8] is one example of such software. “ispace” is a plug-in software for eclipse, and it provides a function that enables users to draw a class dependency graph. Users can gain an understanding of class dependency in source codes.

However, when users describe a large software program using such tools, it is hard for users to understand dependency intuitively because of its complexity.

In this way, it is important for developers to have a function that provides a guide for source code reading. Calculation of the source code module importance is required. We propose such a method in the next section.

4. Module Importance

When developers try to read source codes without software documentation, they find it difficult to decide where to start reading the source code when they have to refer to complicated class dependency diagrams. Here, we propose a method that defines the importance of source code modules using information obtained from source code analysis. In this paper, we apply our source code analysis method to Java.

Module importance is calculated by analyzing strings in source codes. After the analysis, the reference and referenced relationships are obtained. In this context, we refer to classes with a large value of importance as “frequently-used” classes and these are treated as important classes in the source code. Here, we show an algorithm for obtaining module importance.

(1) Analyze input source code in order to collect tokens relevant to classes or instances of classes.
(2) From obtained information, the reference and referenced class numbers are counted.
(3) Sort classes in descending order of summation of reference number and referenced number for each class. Sorting is based on the following rules:
  (a) If the summation of the reference numbers and the referenced number of the classes is equal, then the classes should be sorted in descending order based on the referenced number.
  (b) If the summation of the reference numbers and the referenced number is equal, and also the referenced number is equal, then the classes should be sorted in descending order based on the reference number.
  (c) If the summation of the reference number and the referenced number is equal, and also the reference number and the referenced number are equal, then the classes should be sorted in descending order based on the sum of the reference number and the referenced number of the neighboring class.
  (d) If neighboring class values are equal, then it does not matter which number (the reference number or the referenced number) you choose. Here neighbor means classes that are connected to a focus class.
  (e) If it is not possible to decide upon the precedence order of the classes, then the antecedence classes are given priority in the ordering.

Module importance serves as a guide for source code reading. We defined it as, “frequently used classes are important.” Therefore we designed the above algorithm. In order to sort the importance in detail in the case in which the summation of the reference number and the referenced number is equal, the algorithm uses the neighbor class’s value for sorting.

5. Case Study

Here, we show a case study of our method using a small Java source code. The intended software is a template of image processing software, which can be used by students in a practical software development class to modify the software in order to develop a more sophisticated program. The name of the template software is “SimColorBase” [9], which can process file filtering for images such as “brighten up” or “darken up”. Figure 2 shows a SimColorBase screen shot.

Fig. 2 SimColorBase screen shot

In this case study, we target classes included in SimColorBase and do not cover classes provided by Java.

This software consists of two packages “dyschromatopsia” and “dyschromatopsia.filter”. The “dyschromatopsia” package contains the following four classes: “ImageFileChooserFilter.java,” “ImageOpenFile.java,” “ImagePanel.java,” and “SimWindow.java”.

On the other hand, the “dyschromatopsia.filter” package has “BrighterFilter.java” and“DarkerFilter.java” classes. We show the package structure of the SimColorBase in Figure 3.

The ImagePanel class provides functions for image processing, such as “brighten up” or “darken up”, by selecting radio buttons on the Panel. This class is a core class of SimColorBase. The

Table 1 Class dependency data for the SimColorBase class

Importance	Class name	Referenced class	Refer class	total
1	ImagePanel	1	2	3
2	ImageOpenFile	1	1	2
3	SimWindow	0	2	2
4	BrighterFilter	1	0	1
5	DarkerFilter	1	0	1
6	ImageFileChooserFilter	1	0	1

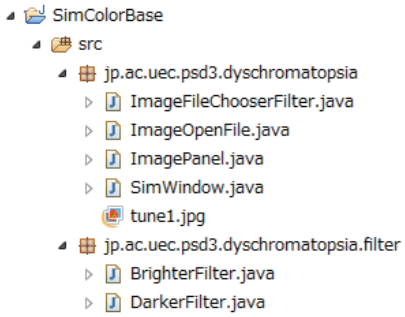

Fig. 3 Package structure of SimColorBase

SimWindow class contains the main method; that is, this class is the entry point of this program. The SimWindow class includes a setting for the menu bar and file processing codes.

The result of the source code analysis shows that the SimWindow class creates an instance of the ImagePanel class and the ImageOpenFile class ; that is, the relationship information indicates that the SimWindow class refers to the ImagePanel class and the ImageOpenFile class.

From the above the analysis, we can determine that the number of the “refer class” is two. We can also determine that the ImagePanel class and the ImageOpenFile class are referred from the SimWindow class; therefore, the number for each “referenced class” is one. Those relationships are shown in Figure 4.

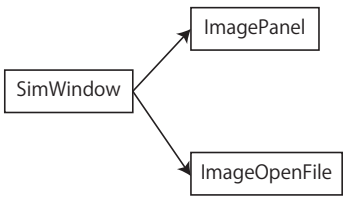

Fig. 4 Relationship among the three classes

We analyzed all the SimColorBase source codes using a similar procedure. We then tallied the “refer class” numbers and the “referenced class” numbers and sorted them into a tabular form (Table 1).

From Table 1, it can be seen that the ImagePanel class is the most important class because the sum of the refer class numbers and referred class numbers was larger than the sum of the refer class numbers and the referred class numbers for the ImageOpenFile class and the SimWindow class, respectively. The ImageOpenFile class and the SimWindow class have the same value of importance; however, the ImageOpenFile class is located on the second rank because of rule 3b (as noted above in Section 4). In this case, the source code should be shown in the following order: ImagePanel → ImageOpenFile → SimWindow → BrighterFilter → DarkerFilter → BrighterFilter.

A class dependency diagram of SimColorBase that is generated by ispace [8] is shown in Figure 5. The sum of the refer class number and the referred class number conforms closely to the degree of the corresponding node on the class dependency diagram.

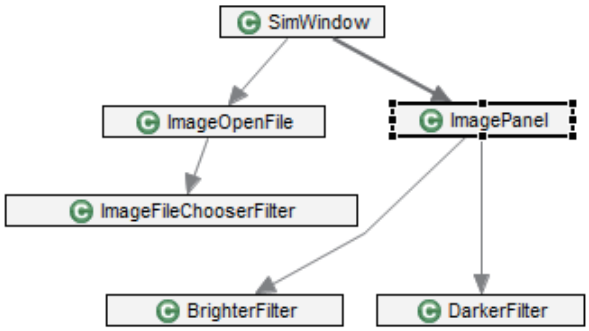

Fig. 5 Class dependency for the SimColor Base class

Here, we discuss the validity of our method using the result of the case study. In the case study, the ImagePanel class was evaluated as the most important class. The ImagePanel class is the core class of the SimColorBase application and defines the graphical user interface of the SimColorBase. For practical purposes, the ImagePanel class is the class for reading first if a developer wants to understand the SimColorBase. Therefore, we find that our method works in practice.

On the other hand, SimColorBase is an application for filtering image files. However the importance of BrighterFilter and DarkerFilter classes that process filtering are low. Such frequently performed classes can not be judged as important in our method. In order to calculate importance in consideration of frequently performed classes, dynamic analysis is required.

In this case study, we analyzed a small software program; however, when targeting large software programs, it is difficult to find points for reading the source code due to the size of the source code or the dependency graph. Our method can reduce the challenges developers face when analyzing source codes by enabling them to calculate the importance of the source code modules. We are also considering other methods for calculating the importance of the dependency using natural language processing methods.

6. Presentation of important class

After calculating the importance of the source code module, the source code is shown according to its order of importance. The importance of the source code can be presented in two different ways. First, the source code can be identified by its appropriate class. Second, the source code’s appropriate class can

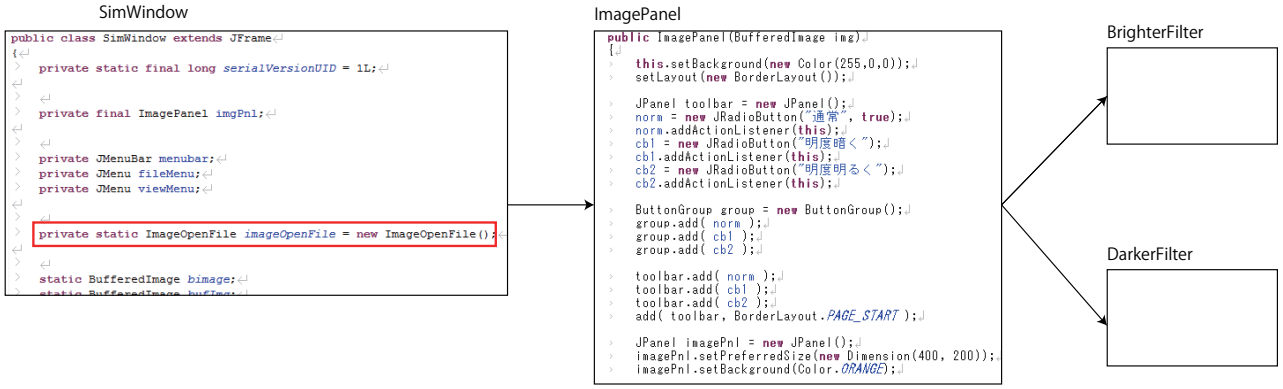

Fig. 6 Presentation of the ImagePanel class (focus class) and related classes

be shown with its related classes. In order to understand the program, it is also important to understand the context of a class; therefore, it is important to also identify the refer class and referred class for each class.

Figure 6 illustrates a case showing the ImagePanel class and its related classes. As can be seen, the ImagePanel class is the focus class. Figure 9 also includes information about the SimWindow class , the BrighterFilter class and the DarkerFilter class. The point of instantiation for the ImagePanel class in the SimWindow class is also focused on.

7. Related works

To date, only research that has targeted supporting source code reading has been done.

Karrer et al. [10] propose a method that presents a focus method centered around a call graph. DeLine et al. [11] propose a source code navigation method. In that method, the source code is displayed with a thumbnail and users can understand the source code using the spatial memory of the entire code. In these research studies, users move the focus point manually; on the other hand, the focus points we propose are presented automatically. Therefore, differences exist between the method we use for detecting the focus class and the methods used by other researchers.

Inoue et al. [12] proposed a component ranking model. In the model, rank is computed using class inheritance, interface implementation, abstract class implementation, variable declaration, instance creation, field access, and method invocation. Moreover the ranking model also considers similarity between the two components. In our method, we restricted use to the appearance of class names that can be obtained from the source code, we do not consider similarity among classes.

8. Conclusion

In this paper, we proposed a method for calculating the importance of a source code module that supports a developer's ability to read source code. Moreover, we conducted a substantive experiment using a small software program. In future works, we have a plan to develop a system based on the proposed method. Moreover, we will construct a source code reading support environment and conduct a survey of software developers. A source code reading support environment requires a supporting function that shows the focus points which software developers are currently reading. As such, the Focus+Context presentation of the source code must also be considered.

References

[1] Haiduc, S., Aponte, J. and Marcus, A.: Supporting program comprehension with source code summarization, *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*, New York, NY, USA, ACM, pp. 223–226 (online), available from ⟨http://doi.acm.org/10.1145/1810295.1810335⟩ (2010).

[2] Haiduc, S., Aponte, J., Moreno, L. and Marcus, A.: On the Use of Automated Text Summarization Techniques for Summarizing Source Code, *Proceedings of 2010 17th Working Conference on Reverse Engineering (WCRE)*, pp. 35 –44 (2010).

[3] Rastkar, S., Murphy, G. C. and Bradley, A. W.: Generating natural language summaries for crosscutting source code concerns, *Proceedings of 2011 27th IEEE International Conference on Software Maintenance (ICSM)*, pp. 103 –112 (2011).

[4] Poshyvanyk, D., Marcus, A. and Dong, Y.: JIRiSS - an Eclipse plug-in for Source Code Exploration, *Proceedings of 14th IEEE International Conference on Program Comprehension, 2006. ICPC 2006*, pp. 252 –255 (2006).

[5] Enslen, E., Hill, E., Pollock, L. and Vijay-Shanker, K.: Mining source code to automatically split identifiers for software analysis, *Proceedings of 6th IEEE International Working Conference on Mining Software Repositories, 2009. MSR '09*, pp. 71 –80 (2009).

[6] Kobayashi, T. and Hayashi, S.: Recent Researches for Supporting Software Construction and Maintenance with Data Mining, *Computer Software*, Vol. 27, No. 3, pp. 3_13–3_23 (2010). (in Japanese).

[7] Matsumoto, Y.: *How to read source code*, No. Nikkei Software 2007, January, Nikkei BP (2007). (in Japanese).

[8] ispace Team: ispace, `http://ispace.stribor.de/index.php?n=Ispace.Home`.

[9] Repository, U. S.: Software development material Image processing program SimColorBase, `https://www.repository.uec.ac.jp/`.

[10] Karrer, T., Krämer, J.-P., Diehl, J., Hartmann, B. and Borchers, J.: Stacksplorer: call graph navigation helps increasing code maintenance efficiency, *Proceedings of the 24th annual ACM symposium on User interface software and technology*, New York, NY, USA, ACM, pp. 217–224 (online), DOI: 10.1145/2047196.2047225 (2011).

[11] DeLine, R., Czerwinski, M., Meyers, B., Venolia, G., Drucker, S. and Robertson, G.: Code Thumbnails: Using Spatial Memory to Navigate Source Code, *Visual Languages and Human-Centric Computing, 2006. VL/HCC 2006. IEEE Symposium on*, pp. 11 –18 (online), DOI: 10.1109/VLHCC.2006.14 (2006).

[12] Inoue, K., Yokomori, R., Yamamoto, T., Matsushita, M. and Kusumoto, S.: Ranking significance of software components based on use relations, *Software Engineering, IEEE Transactions on*, Vol. 31, No. 3, pp. 213–225 (online), DOI: 10.1109/TSE.2005.38 (2005).