Advances in Intelligent Systems Research, volume 161
International Conference on Transportation & Logistics, Information & Communication, Smart City (TLICSC 2018)

A Study on Code Transplantation Technique based on Program Slicing

Lupeng Liu[1, a, *] and Xiaoguang Mao[1, 2, b]

[1]College of Computer, National University of Defense Technology, Changsha 410073, China

[2]Laboratory of Software Engineering for Complex Systems, Changsha 410073, China

[a, *] liulupeng_a@163.com, [b]xgmao@nudt.edu.cn

Abstract. Code transplantation, which is using a function of code from one software directly into another, and make the function work in it. Several tools have been built to transplant code, but there is still no research solved the problem of transplanting open source software. Open source software has the characteristics of coming from unclear source, irregular, and function definition unclear, which is not conducive to transplant. In our research process, we find program slicing can deal with these characteristics, help programmer to get the code related to the function they want to transplant. In this paper, we introduce a method to transplant code from open source software, the results revealed that our method can significantly reduce programmer's work and can be used to real world open source software.

Keywords: Code Transplantation; Program Slicing; Open Source Software.

1. Introduction

With the development of software industry, more and more people learn coding and building software, and there are plenty of different kinds of open-source software. Upon people building a function in a new software, there often exist similar code in other already developed software, which can be reused here.

In academic research, many scholars have studied code reuse, component is a well-studied area in code reuse. Component is a technique which reuse code like building a house, but the function can be reused is specially designed, which is useful in a special new software, but it's not very useful to not designed open source software.

In 2015, Mark Harman et al. proposed the concept of 'code transplantation' [1]. The concept comes from the medical field, the software is compared to human body, the code we want to reuse is compared to organ in human body, and the code reuse process is compared to a surgery of transplanting organ. They proposed a tool 'μScalpel', which can transplant a function from one software to another automatically. In their paper they also proposed some new concepts: Organ: the code have the function to be reused; Donor: the software have the organ; Host: the software to reuse the organ. In this paper we will also use these concepts. In 2017, Stelios Sidiroglou-Douskos et al. proposed a tool 'CCC' [2], which have similar function as μScalpel.

Transplant code automatically is a very attractive idea, but current research is based on specially designed experiments, and the organ used is designed for the experiment, which means it cannot be used in common practice now.

Program slicing is a well-studied area in software engineering, it is the computation of the set of program statements, the program slice that may affect the values at some point on interest. Program slicing is widely used in fault localization [3] and automatic software repair [4], in code reuse research, we can use it to slice out our organ that to be transplanted.

Based on current research, we propose our method of using program slicing to code transplantation. The contribution of our work is as follows:

1. Reduce programmer's time of reading irrelevant code;
2. Help programmer to analyze programs;
3. Help programmer to transplant organs.

2. Experiment Design

2.1 Tools and Dataset

According to our initial analysis, there are several slicing tools, like Kaveri-Indus [5], JSlice, WALA, JavaSlicer [6]. We select JavaSlicer to carry out our work. The reason is: firstly, JavaSlicer is widely used. Secondly, JavaSlicer is open source and well documented. We choose projects from both Github and the book Core Java as our dataset. The project from Github can tell us the practical value of our method, and the project in Core Java can tell us the efficiency of program slicing in code transplantation.

2.2 Evaluation Metrics

We evaluate our experiment in three Metrics. Firstly, to illustrate that our study can reduce programmer's time of reading irrelevant code, we count the lines of code (Loc) of the project code and the slicing results (Loc-s), and calculate the percentage of Loc-s and Loc (Pct). Secondly, to illustrate that our study is helpful for programmer to analyze program, we count the number of variables (Vars) related to program slicing results. Thirdly, the most important is to prove the practical usage of our work, we chose projects from Github and implement the code transplantation procedure.

2.3 Implementation of the Experiment

We implement our experiment in four steps. Firstly, define the function we want to transplant. Secondly, browse the code in the Donor, locate the position of the code to be transplanted. Thirdly, use slicing tool to slice code in the Donor, get the results as Organ and record the data. Lastly, transplant the Organ into the Host.

3. Results and Analysis

We evaluate our work in two main aspects: the efficiency of program slicing in code transplantation and the practical value of our method.

3.1 The Efficiency of Program Slicing in Code Transplantation

Table 1 shows program slicing results of 20 projects. In the table, we can see that the percentage of Loc-s and Loc is around 10%, the biggest is 23%, which means the code in program slicing results is far less than the original code in the Donor. By this way, programmer don't have to read all the code in the Donor, they just need to read the program slicing results, irrelevant code is ruled out by the slicing process, so the time of reading code is significantly reduced. In the Vars column, we counted the number of variables related to the program slicing results, we can see the number of variables is from 2 to 8, and the number bigger than 5 is nine in twenty, around half of it. We know that while reading a foreign code from other software, we need to analyze the dependence of the variables, which is important to understand the function of the code. Moreover, when transplant code, the analysis of the dependence of variables is even more important, because we have to match the variables in the Donor and the Host. The program slicing process can narrow down the scope of code and tell us which variable is related to our organ in the Donor, this function replaces the work of programmer before.

3.2 The Practical Value of Program Slicing Based Code Transplantation

In this part, we choose projects from Github to act as our Donor and Host. We choose two project and name them as Calculator1 and Calculator2. Calculator1 is the Donor and Calculator2 is the Host. Calculator1 is a well-developed calculator which has calculate ability and a good appearance. Calculator2 is a framework of calculator which only has a framework without actual calculate function and is poorly looked. The initial Calculator1 and Calculator2 are shown follows in FIGURE 1.

Table 1. Slicing Results.

Project Name	Loc	Loc-s	Pct	Vars	Project Name	Loc	Loc-s	Pct	Vars
BigInteger	33	3	9%	3	Static	76	3	4%	2
Input	25	4	16%	3	Reflection	126	6	5%	5
CompoundInterest	57	7	12%	4	MethodTable	61	4	7%	4
LotteryArray	39	8	21%	6	CopyOf	58	9	16%	5
LotteryDrawing	46	8	17%	6	Enum	33	4	12%	2
LotteryOdds	30	3	10%	3	StackTrace	38	5	13%	5
Retirement	39	9	23%	6	GenericReflection	123	5	4%	4
Retirement2	44	8	18%	6	CircularArrayQueue	124	10	8%	8
Param	89	12	13%	4	LinkedList	59	13	22%	7
Constructor	83	3	4%	2	InetAddress	29	3	10%	3

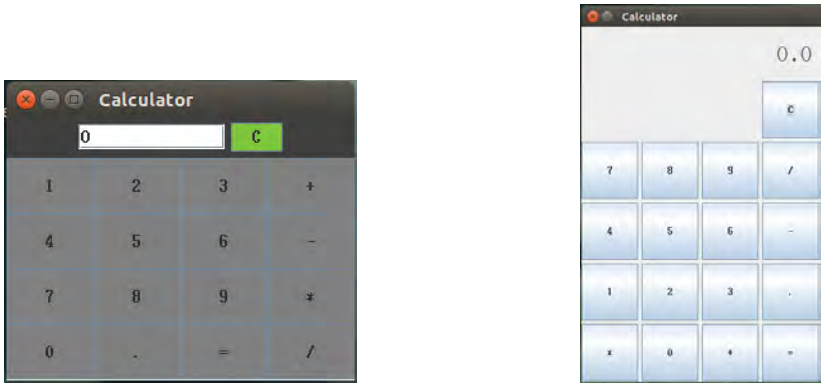

Figure 1. Initial Calculator1 and Calculator2.

As described in the implementation part above, we conduct our experiment in four steps:

Define the function we want to transplant. Our work is to transplant two functions from the Donor to the Host, the calculate function and the appearance function.

Browse the code in the Donor, locate the position of the code to be transplanted. After browsing code in Calculator1, we located that the calculate function is in line 78 in the file Calculator.java with the name public void doit () {}, and the appearance function is in line 66 in the file Calculator.java with the name of public void setFontAndColor () {}.

We use JavaSlicer to slice the project.

First, we slice for the calculate function. After slicing, we get the slice result and name it Organ-calculate. The Organ-calculate is in line 31, 34, 38 and 92 in file Calculator.java and line 5 in file Main.java. The related variables are jbs, c_jbs, s, i, s1, s2, str, str0 and flag. The code is shown as below:

```
Main.java:
5:    new Calculator (). ShowMe ();
Calculator.java:
27: public Calculator () {
31:    jbs = new JButton [16];
32:    c_jbs = new JButton("C");
33:    String s = "123+456-789*0. = /";
34:    for (int i = 0; i < jbs.length; i++) {
35:          jbs[i] = new JButton (s.substring(i, i + 1));
36:     }
38:    doit ();}
78:    public void doit () {
79:          c_jbs. addActionListener(new ActionListener() {
81:                public void actionPerformed (ActionEvent e) {...}});
92:          for (int i = 0; i < jbs.length; i++) {
```

```
93:    jbs[i]. addActionListener (new ActionListener () {
95:    public void actionPerformed (ActionEvent e) {...}});}}
```

Second, we slice for the appearance function. After slicing, we get the slice result and name its Organ-appearance. The Organ-appearance is in line 31, 63 and 73 in Calculator.java and line 5 in Main.java. The related variables are f, jtf, c, c_jbs, jbs, jp1. The code is shown as below:

```
Main.java:
5:    new Calculator (). showMe();
Calculator.java:
27: public Calculator () {
31:    jbs = new JButton[16];
32:    c_jbs = new JButton("C");
33:    }
57: public void showMe () {
63:    setFontAndColor ();
64:    }
66: private void setFontAndColor () {
68:            Font f = new Font ("Gungsuh", Font.BOLD, 14);
69:        jtf. setFont(f);
70:        Color c = new Color (128, 198, 077);
71:        c_jbs.setBackground(c);
72:        jp1. setBackground (Color.DARK_GRAY);
73:        for (int i = 0; i < jbs.length; i++) {
74:            jbs[i]. setBackground(Color.gray);
75:        }}
```

We transplant organs into the Host Calculator2.

First, we transplant Organ-calculate into Calculator2. After analyze the variables in Organ-calculate and Calculator2, we find the match (from Calculator2 to Organ-calculate): buttons - jbs, C - c_jbs, s - titles. Other variables are not existing in Calculator2 and need to be new declared. We adjusted the variables and transplanted Organ-calculate into Calculator2, the result is in FIGURE 2.

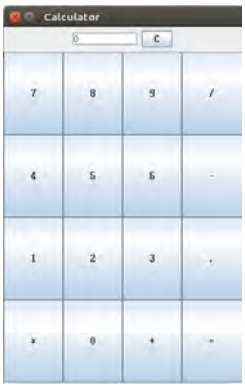

Figure 2. Calculator2 after transplantation of Organ-calculate.

Second, we transplant Organ-appearance into Calculator2. After analyze the variables in Organ-appearance and Calculator2, we find the match (from Calculator2 to Organ-appearance): C - c_jbs, buttons - jbs, buttonPanel - jp1. Other variables are not existing in Calculator2 and need to be new declared. We adjusted the variables and transplanted Organ-appearance into Calculator2, the result is in FIGURE 3.

Figure 3. Calculator2 after transplantation of Organ-calculate and Organ-appearance.

4. Related Work

Code transplantation was first proposed in Mark Harman's work in 2015, they created a tool ‘µScalpel’, which can transplant code automatically. After that, Stelios Sidiroglou-Douskos et al. proposed another tool ‘CCC’ in 2017, can also transplant code automatically. Though both the two tools can transplant code automatically, they are specially designed, they can only work in designed environment, and can't be use to open source software.

5. Conclusion and Future Work

In this paper, we proposed a study on code transplantation technique based on program slicing. We analyzed the efficiency of program slicing in code transplantation, explained our method can reduce programmer’s time of reading irrelevant code and can help programmer to analyze programs. At the end of our experiment, the transplantation of two Organs explained the practical usage of our method.

In the future, we plan to build a tool based on program slicing to handle code transplantation problem, make the transplantation of open source code easier.

Acknowledgements

This research is supported by the National Natural Science Foundation of China (Grant No.61672529).

References

[1]. Earl T. Barr, Mark Harman, Yue Jia, Alexandru Marginean, and Justyna Petke. Automated software transplantation. In ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA), pages 373–384, Baltimore, MD, USA, July 2015.

[2]. Sidiroglou-Douskos S, Lahtinen E, Eden A, et al. CodeCarbonCopy[C]// JointMeeting. 2017:95-10.

[3]. Mao Xiaoguang, et al. “Slice-based Statistical Fault Localization.” Journal of Systems and Software 89:51-62, 2014.

[4]. Y. Qi, X. Mao, Y. Lei, Z. Dai, C. Wang. The strength of random search on automated pro-gram repair. In: Proc. 36th Int’l Conference on Software Engineering (ICSE), 2014, 254-265.

[5]. Jayaraman, Ganeshan, Venkatesh Prasad Ranganath and John Hatcliff. “Kaveri: Delivering the Indus Java Program Slicer to Eclipse.” FASE (2005).

[6]. Hammacher C, Streit K, Hack S and Zeller A. “Profiling java programs for parallelism.” The Workshop on Multicore Software Engineering. 2009.