I J C T A, 9(7), 2016, pp. 3109-3115 ISSN: 0974-5572

Substitution Based Rules for Efficient Code Duplication on Object Oriented Systems

Anoop Sreekumar R.S.* and R.V. Sivabalan**

ABSTRACT

Designing of the Software engineering principles considerably reduce the software complexity on application projects. A critical issue of utilizing software engineering principles is to avoid the code duplication on object-oriented systems. However, traditional methods on software engineering suffer problems such as code duplication on object oriented systems compromising the code duplication efficiency and software quality rate. In this paper, a method to avoid code duplication on object oriented architectural design called Liskov Substitution Software principle with DupCode Removal (LSSP-DR) is employed. Liskov Substitution Software principle with DupCode Removal method clearly detects the duplicated code and improves the software quality rate on application projects. The experimental results based on the software program code show the outperformance of LSSP-DR method over state-of-the-art methods. In particular, using LSSP-DR method to avoid code duplication not only results in the improvement of code duplication detection, but also requires the least code duplicate removal time.

Keywords: Code Duplication, Object-oriented Systems, Liskov Substitution Software, DupCode Removal, Hoare Software Logical rules

1. INTRODUCTION

Efficient designing of software engineering principles significantly minimizes the software complexity on application projects. However, the software engineering principles need to evolve an improved quality system. Pool of research works has been concentrated for reducing the duplicate code using software engineering principles. Identification of Extract Class Refactoring in Object Oriented Systems (IECR) [1] extracted class opportunities aiming at improving the design quality using agglomerative clustering algorithm. Observe Model Exercise with Undetermined Input Spaces (OME-UIS) [2] observed the existence of new events during test case execution resulting in the improvement of software quality testing. Identification of duplicate code related to web applications was presented in [3] using library functionalities and page optimization

In [4], relative defect proneness was addressed using Cox Proportional Hazards model resulting in the improvement of duplicate code detection. Though accuracy of duplicate code was concentrated in the above said methods, the duplicate code removal time was not focused. The duplicate code removal time is focused in the LSSP-DR method by applying Hoare Software Logical rules.

One of the pernicious problems faced by software engineering is the rate of software quality to be addressed. A generic method for automatic software repair was presented in [5] using selection and genetic operators. Fault localization in web applications Two different tools namely, clone detection in ant, clone detection in gannt project was presented in [6] aiming at reducing the clone detection time Supervised and semi supervised adaptation was applied in [7] to improve the duplicate code accuracy. Based on the aforementioned methods and techniques in this paper, a novel method called, Liskov Substitution Software

* Research Scholar, CSE Noorul Islam University, *Email: anoopsree369@gmail.com*

** Associate Professor, MCA Noorul Islam University, *Email: rvsivan@gmail.com*

principle with DupCode Removal (LSSP-DR) is presented. The elaborate description of LSSP-DR method is presented in the following sections.

1.1. Liskov Substitution Software principle with DupCode Removal

We first propose a method for extracting strong object code behavioral properties using BNF notation and estimate the duplicate code detection by analyzing the syntax and semantics through a correlation analysis between the arbitrary variable and commands. The following sections describe our method to avoid code duplication on object oriented architectural design that improves the software quality rate on application projects.

Based on the object behavioral properties with BNF notation given above, the Strong Object Code behavioral properties in the proposed LSSP-DR method restructure the code (i.e. in application project) based on the behaviors of the code structure in the application project. Figure 2 given below shows the algorithmic description of BNF-based Object Code Behavioral.

Input: Variables "$Var_i = v_1, v_2, ..., v_n$", application project "$P$",
symbols "$Sym_i = s_1, s_2, ..., s_n$", commands
"$C_i = c_1, c_2, ..., c_n$"

Output: Efficient detection of code duplication

Step 1: Begin
Step 2: For each application project "P"
Step 3: For each Variables "Var_i" and symbols "Sym_i"
Step 4: Form object code behavioral properties using (1) and (2)
Step 5: Form BNF notation using (3), (4), (5), (6) and (7)
Step 6: End for
Step 7: End for
Step 8: End

Figure 1: BNF-based Object Code Behavioral algorithm

As shown in the above figure, the BNF-based Object Code Behavioral algorithm includes two main parts. For each application project, object code behavioral properties are formed followed which the BNF notation is formalized aiming at improving the rate of duplicate code detection.

1.2. Liskov Substitution Software principle

One of the important principles in object oriented architectural design is the substitutability. The third step in the design of LSSP-DR method is Liskov Substitution Software principle. The Liskov Substitution Software principle in the LSSP-DR method states that, in an application program "*P*" if "Sub_t" is a subtype of type "*t*", then objects of type "*t*" may be replaced with objects of type "*Sub*" without changing the occurrence and expression. The Liskov Substitution Software is based on the semantic condition rather syntactic relation that makes context aware code mapping in an efficient manner. Figure 5 shows the block diagram of Liskov Substitution Software principle.

As shown in the figure, the Liskov Substitution Software principle performs context aware code mapping aiming at improving the software quality rate. The Liskov Substitution

Software principle with DupCode Removal method clearly detects the duplicated code and improves the software quality rate on application projects. The Liskov Substitution Software principle includes a predicate "*Subs*" with four arguments and is formulated as given below

$$LS = Subs(var, value, E, E) \quad (1)$$

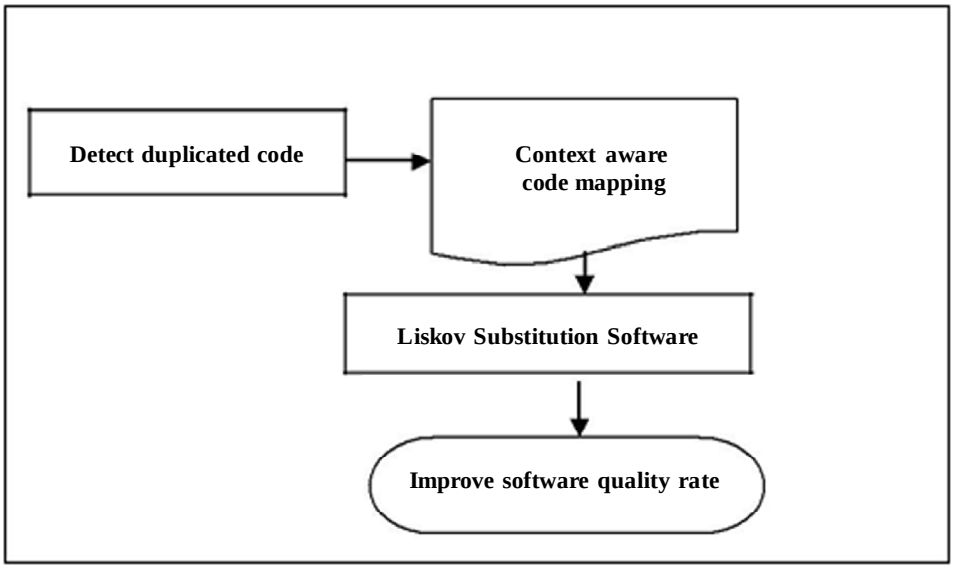

Figure 2: Block diagram of Liskov Substitution Software principle

From (), the Liskov Substitution "*LS*" includes the name of the variable "*var*", value of the variable "*value*", expression before substitution "*E*" and expression after substitution "*E*" respectively. Finally, the context aware code mapping in LSSP-DR method easily detect the repeated or unwanted code in relation with DupCode Removal method on the object oriented architectural design.

3. EXPERIMENTAL DESIGN

The proposed method Liskov Substitution Software principle with DupCode Removal (LSSP-DR) is experimented using JAVA program code. The program written in JAVA code used to identify and avoid code duplication level on the application project using the software Engineering principles. The LSSP-DR method is implemented using the following seven open-source program from http://sourceforge.net (for sample the table includes five source program list) to minimize the code duplications [4].

The code duplication level using the above method is minimized in a significant manner. Proposed work is compared against the existing work such as Identification of Extract Class Refactoring in Object Oriented Systems (IECR) [1] and Observe Model Exercise with Undetermined Input Spaces (OME-UIS) [2]. Experiment is conducted on factors such as duplicate code detection rate, duplicate code removal time, software quality rate and duplicate code detection accuracy.

The duplicate code detection rate is the size of duplicate code detected from the actual size of software program code. The duplicate code detection rate is mathematically formulated as given below.

$$DCDR = \left(\frac{Duplicate\ code}{Size} \right) * 100 \tag{2}$$

From (2), the duplicate code detection rate "*DCDR*" is the ratio of size of duplicate code being detected "*Duplicate code*" to the actual size of software program code "*Size*". Lower the duplicate code being detected more efficient the method is said to be. The duplicate code removal time measures the time taken to removal the duplicate code present in the application program. The duplicate code removal time is mathematically formulated as given below.

$$DCRT = Size * Execution\ time\ (duplicate\ code\ removal) \tag{2}$$

From (2), the duplicate code removal time "*DCRT*" is measured on the basis of the size of the application program "*Size*" with respect to time taken for each software program code. Lower the duplicate code removal time more efficient the method is said to be and it is measured in terms of milliseconds (ms). The duplicate code detection accuracy is the ratio of size of correct assessments made to the total size of all assessments considered for conducting experiment. The duplicate code detection accuracy is mathematically formulated as given below.

$$A = \left(\frac{Number\,(size)\,of\ \ correct\ \ assessments}{Number\,(size)\,of\ \ all\ \ assessments} \right) * 100 \tag{3}$$

From (3), the duplicate code detection accuracy ‘’ is calculated and measured in terms of percentage (%). Higher the duplicate code detection accuracy more efficient the method is said to be.

4. DISCUSSION

Experimental results are provided in this section, to evaluate the theoretical framework and to demonstrate the performance of the proposed method Liskov Substitution Software principle with DupCode Removal (LSSP-DR) with some implementation results.

4.1. Scenario 1: Duplicate code detection

In this section to check the efficiency of LSSP-DR method, the metric duplicate code detection is evaluated and compared with the state-of-the-art works, IECR [1] and OME-UIS [2]. To deliver with a detailed performance, in Table 2 we apply duplicate code with respect to the total size to obtain the duplicate code detection rate and comparison is made with two other existing methods, IECR and OME-UIS respectively. The duplicate code detection rate in LSSP-DR method refers to the amount of duplicate code detected on object oriented systems for differing size of software program code. Lower duplicate code being detected results in the improvement of the method.

A comparative analysis for duplicate code detection with respect to different size of software program code was performed with the existing IECR and OME-UIS is shown in Figure 6. The increasing size of software program code in the range of 15 KB to 105 KB is considered for experimental purpose. As illustrated in figure, comparatively while considering increased size of software program code, the

duplicate code detected is also increases, though betterment achieved using the proposed method LSSP-DR which records lesser number of duplicate codes being obtained.

The measurement of duplicate code detection is comparatively reduced using the LSSP-DR method when compared to two other existing methods [1] [2].

Table 1
Tabulation for duplicate code detection rate

Size of software program code (KB)	*Duplicate code detection (%)*		
	LSSP-DR	*IECR*	*OME-UIS*
15	41.35	47.83	54.44
30	48.71	54.73	62.78
45	53.23	59.25	67.30
60	55.86	61.88	68.93
75	50.21	56.23	64.28
90	53.14	59.16	67.21
105	60.29	66.31	74.36

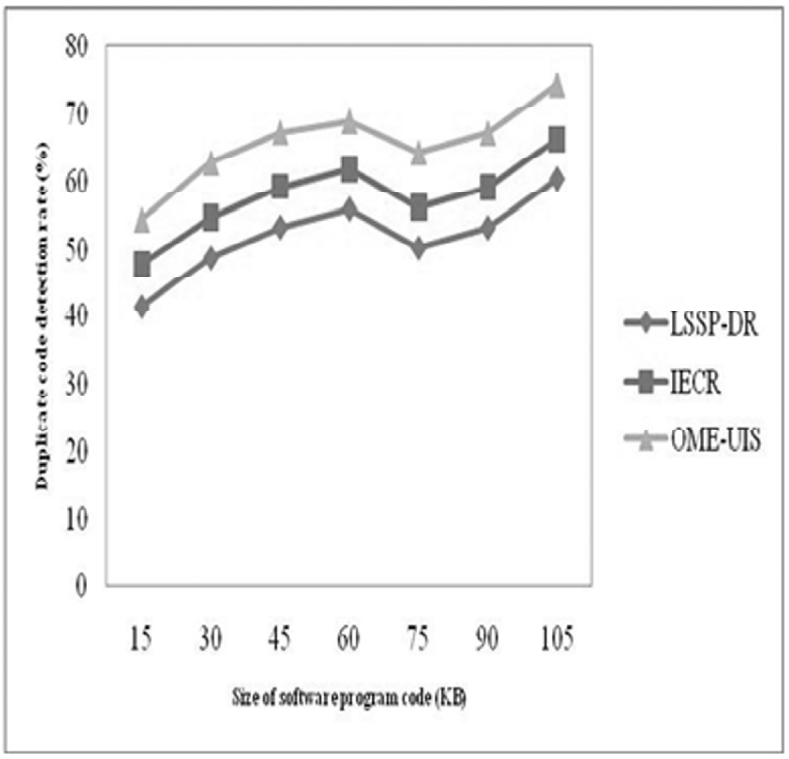

Figure 3: Measure of duplicate code detection

This improvement in duplicate code detection is because of the strong object behavioral properties that are based on the behaviors of the internal object oriented method processing. In addition, the BNF notation obtained from the internal object oriented processing helps for any size of software program code to obtain the duplicate code rate in a dynamic manner minimizing the duplicate code being detected by 11.91% and 26.88% compared to IECR [1] and OME-UIS [2] respectively.

4.2. Scenario 2: Software quality rate

In order to measure the efficiency of software quality rate obtained through the method LSSP-DR, the application program size is considered during the experiment.

The software quality rate using LSSP-DR method is provided in an elaborate manner in table 1 with different code size and experiment using JAVA. Figure 8 shows the software quality rate on object oriented systems with respect to differing application program size during experimental settings at different time intervals.

As depicted in the figure with the increase in the application program size, the software quality rate is also increased. But when compared to the state-of-the-art works, the data software quality rate is increased using the proposed method LSSP-DR. The software quality rate is improved owing to the fact that the proposed method uses the Liskov Substitution Software principle with DupCode Removal method. By

Table 2
Tabulation for software quality rate

Methods	*Software quality rate (%)*
LSSP-DR	71.83
IECR	65.21
OME-UIS	58.36

applying Liskov Substitution Software principle with DupCode Removal method, duplicate code is detected in an efficient manner including a substitution predicate. So, the software quality is improved by 9.21% when compared to IECR and 10.50% compared to OME-UIS respectively.

4.3. Scenario 3: Duplicate code detection accuracy

In this section, duplicate code detection accuracy is measured with varying software program code size using five open-source programs from http://sourceforge.net.

Table 1 and figure 4 shows the measure of duplicate code detection accuracy with respect to seven source programs of differing size. From the figure, it is evident that the rate of duplicate code detection accuracy increases with the increase in the size of software program and comparatively better than the two other methods [1] [2].

The application of DupCode removal algorithm with Liskov Substitution Software principle that measures the context aware code mapping to minimize the duplicate code and therefore to improve the rate of duplicate code detection accuracy. Moreover, by identifying the duplicate code in an efficient manner,

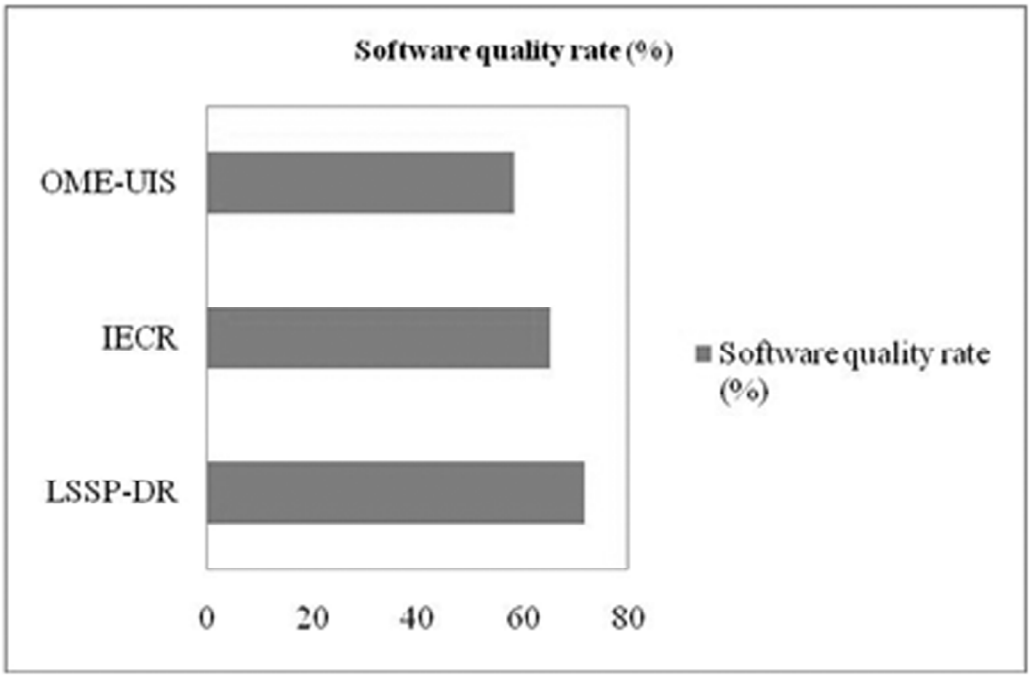

Figure 4: Measure of software quality rate

Table 2
Tabulation for duplicate code detection accuracy

Size of software program code (KB)	*Duplicate code detection accuracy (%)*		
	LSSP-DR	*IECR*	*OME-UIS*
15	78.12	71.49	64.31
30	80.14	74.12	67.09
45	82.13	76.11	69.08
60	79.67	73.65	66.62
75	81.49	75.47	68.44
90	83.55	77.53	70.50
105	85.73	79.71	72.68

LSSP-DR method increases the duplicate code detection accuracy by 7.49% compared to IECR and 16.15% compared to OME-UIS.

5. CONCLUSION

A Liskov Substitution Software principle with DupCode Removal method has been designed to avoid code duplication on object oriented architectural design and to improve the duplicate code detection accuracy. We adopt Strong Object Code behavioral properties to improve the duplicate code detection that forms BNF notation for differing software program code size for minimizing code duplication. With the Strong Object Code behavioral properties and BNF-based Object Code Behavioral algorithm, duplicate code being detected in minimized in an efficient manner. Then, the Hoare Software Logical rules are applied for each module in system software to perform partial correctness using assignment axiom scheme. Finally, Liskov Substitution Software principle is applied on the semantic condition context aware code mapping detects the duplicated code and improves the software quality rate on application projects in a significant manner. Experimental evaluation is conducted with the seven open-source program from http://sourceforge.net. To measure the effectiveness of the proposed method parameter analysis are performed in terms of duplicate code detection, duplicate code removal time, software quality rate and duplicate code detection accuracy with respect to differing application program size. Compared to the existing object oriented methods, the proposed LSSP-DR method decreases the duplication code by 19.40% and software quality rate is improved to 19.72% compared to IECR and OME-UIS.

REFERENCES

[1] Marios Fokaefsa, Nikolaos Tsantalisa, Eleni Strouliaa, Alexander Chatzigeorgioub, "Identification and application of Extract Class refactoring in object-oriented systems", Elsevier, *Journal of Systems and Software*, Volume 85, Issue 10, October 2012, pp. 2241-2260.

[2] Bao N. Nguyen and Atif M. Memon, "An Observe-Model-Exercise Paradigm to Test Event-Driven Systems with Undetermined Input Spaces", *IEEE Transactions on Software Engineering*, Volume 40, Issue 3, March 2014, pp. 216-234.

[3] Josip Maras, Maja Stula, Jan Carlson, and Ivica Crnkovic, "Identifying Code of Individual Features in Client-side Web Applications", IEEE Transactions on Software Engineering, Volume 39, Issue 12, December 2013, pp. 1680-1697.

[4] Mark D. Syer, Meiyappan Nagappan, Bram Adams, and Ahmed E. Hassan, "Replicating and Re-Evaluating the Theory of Relative Defect-Proneness", *IEEE Transactions on Software Engineering*, Volume 41, Issue 2, February 2015, pp. 176-197.

[5] Claire Le Goues, ThanhVu Nguyen, Stephanie Forrest, and Westley Weimer, "GenProg: A Generic Method for Automatic Software Repair", *IEEE Transactions on Software Engineering*, Volume 38, Issue 1, January/February 2012, pp. 54-72.

[6] Francesca Arcelli Fontana, Marco Zanoni, Andrea Ranchetti, and Davide Ranchetti, "Software Clone Detection and Refactoring", *Hindawi Publishing Corporation, ISRN Software Engineering*, Volume 2013, January 2013, pp. 1-9.

[7] Wen Li, Lixin Duan, Dong Xu, and Ivor W. Tsang, "Learning with Augmented Features for Supervised and Semi-Supervised Heterogeneous Domain Adaptation", *IEEE Transactions on Pattern Analysis and Machine Intelligence archive*, Volume 36, Issue 6, June 2014, pp. 1134-1148.