재사용 SW Component에 대한 Test 자동화

김종석
삼성전자, DM 연구소
js11.kim@samsung.com

Automated Testing for Reusable Software Components

Jongseok Kim
DM R&D Center, Samsung Electronics

ABSTRACT

Reusing software components without proper analysis is very risky because software components can be used differently from the way that they were developed. This paper describes a new testing approach for reusing software components. With this new approach, it is possible to automatically decide if software components can be reused without retesting. In addition, when retesting is required for reusing software, test cases are generated more efficiently using the previous testing history.

1. Introduction

The reusability of software components is a major issue of software developers because reusing software components can save development time and effort. In addition, it can reduce errors [1]. Generally, reusable software components are regarded safe because they are tested during development; however, reusing software components can be risky if they are not reanalyzed for reuse. Although software components have passed testing in the development environments, there can be problems in reusing them in their new environment. Therefore, the reanalysis must be required before they are reused, and it often requires additional tests. This paper describes a new testing approach for reusable software components. This new approach is based on studies of software testing using a Markov chain [6].

2. Testing procedure for software reuse

Generally, reliable software testing is testing that finds many errors, but this concept is too abstract. We define reliable testing differently. Software testing is defined as the process of demonstrating correspondence between a program and its specification [3][8]. Therefore, the procedure for testing software can be considered as a simulation of the software usage, and when the software is tested as close as possible to its real usage, we can say that it is reliable software testing. Our approach is based on this idea. In this section, the procedure of the new approach is described.

Our testing approach consists of five steps. In the first step, the usage model [2][6][10] for the new environment is created when a software component is reused. The usage model is a Markov chain that shows the expected behavior of a software component in the new environment. In the second step, the initial test cases are generated and executed if the previous test history is not available. The third step is to create and update the test model [2][10], which keeps the previous test history. The fourth step is to measure the stopping criterion using the difference between the usage model and the test model. In this step, it will be decided whether or not retesting is required when a software component is reused. The fifth step is to generate and to execute test cases for retesting a software component when additional tests are required. After generating and executing test cases for software reuse, the fourth step is repeated to decide if the software component needs more testing before reusing it. Figure 1 illustrates the general process of the approach

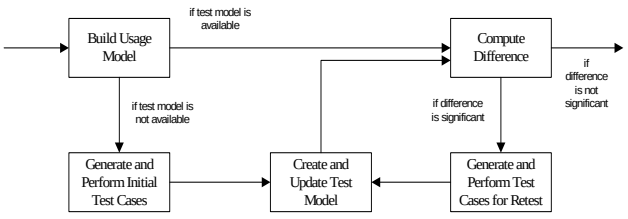

Figure 1. Overview

2.1 Usage model creation

The usage model is a Markov chain that simulates the actual usage of a software component by assigning probabilities on arcs based on the behavior of a software component. The usage model consists of states and arcs. The states represent statuses of the software behavior, and all states must be unique to present different statuses of the software behavior. In addition, the states must obey the Markov property that the next state is determined independently of the previous states; furthermore, all states are connected by the directed arcs. The arcs in the usage model usually represent inputs applied to a software component. Inputs can come from users, systems, and

other software components; thus, they can cause transitions from one state to other states. Each arc in the usage model is also associated with a probability to simulate the actual usage. There are several ways to assign the probabilities into arcs, such as real data from the prototypes, customer surveys, or the judgment of domain experts [6]. However, in the case where the actual usage is unknown, the probabilities can be assigned uniformly [10].

Generally, when a software component is reused in a new environment, the functions of the software component are not changed. Only their usages are changed. Therefore, the usage model can be rebuilt by assigning different probabilities to reflect the new environment.

2.2. Initial test case generation

This step is performed only when the previous test model is not available. To create the initial test model, several test case generation methods using the usage model can be used in this step, such as all path generation and statistical methods. In this paper, the random test case generation, which is one of the statistical methods, is used to generate the initial test cases for the case study. After the initial test is performed, all test results are recorded on the test model, which has the same structure of the usage model but has frequencies instead of probabilities.

2.3. Test model update

The test model is critical in finding the areas where more tests are needed when a software component is reused. The test model is created by initializing the usage model with the same states and arcs; the difference is that it uses the frequency counts instead of the probabilities in the usage model as the arc labels. Initially, the frequencies of the arcs in the test model are set to zero, and whenever inputs are used in test cases during testing, the frequency of the corresponding arc increases by one. Moreover, to handle errors, the failure state is used, so whenever an error occurs, the frequency of the arc between the current state and the failure state increases by one.

In the new approach, multiple test models are maintained instead of the original single test model. The problem of using a single test model is that unnecessary retesting may be performed when a software component is reused. For example, sometimes, the difference between the new usage model and the test model may be significant although a very similar usage model was previously used. In this case, unnecessary testing is required to reduce the difference between the new usage model and the test model. To avoid this problem, all test models must be kept separately instead of using a single test model to keep the test history. Therefore, when a software component is reused, the new usage model is compared to the history of the test models. If there is a similar test model to the new usage model, a software component is reused without retesting. However, if there is no test model that is similar enough, retesting is necessary.

In this approach, all test models are kept in addition to the main test model that is the same test model in another articles [2][10], and test cases are generated based on the most similar test model. When test cases are applied, not only the most similar test model is updated but also the main test model is updated, and both updated test models are kept in the test history. Using this approach, it is possible to find out if there is a similar usage model that was used before. When test case generation is needed for the new usage environment, the number of test cases can be reduced because test cases are generated based on the most similar test model. Figure 2 illustrates the updating of the test model.

Using this method, all test models can be maintained in addition to the main test model, so when a software component is reused, it is possible to find out if a very similar usage model has been previously used. In the case where retesting is required, the number of test cases can also be reduced because the test cases are generated based on the closest test model.

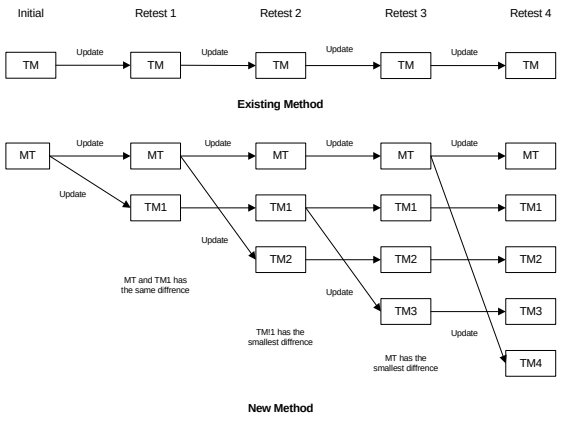

Figure 2. Updating the Test Model

2.4. Stopping criterion

For software reuse testing, the same stopping criterion used in the software development process is not very useful. Therefore, a new stopping criterion is needed for software reuse testing.

In this paper, the difference between the usage model and the test model is used as the stopping criterion for software reuse. When a software component is reused, the new usage model might have different arc probabilities from

the previous usage model. Therefore, in order to find that the new usage model is similar enough to the test model, the difference is measured. If the usage and test models are similar, it means that a software component has been tested according to the actual usage in the new environment. In this case, no more tests are required to reuse a software component. However, if the difference is significant, additional tests are required because a software component has been tested significantly different from the usage in the new environment. In this case, retesting is performed until the test model is similar enough to the new usage model. Therefore, the simple linear regression method is used to measure the difference between the usage model and the test model focusing on the inputs.

2.4.1. Simple Linear Regression

Regression analysis is generally used to describe the relationship between two or more variables [4], and linear regression is used to show variables are linearly related. In this study, simple linear regression is used to show if the usage model and the test model are the same. In this approach, the usage model becomes the independent variable *X*, and the test model becomes the dependent variable *Y*. In addition, the expected frequencies are computed for the arcs in the usage model. To show that the difference between two models is not significant, the intercept (β_0) and the slope (β_1) are estimated using the least square method, and we check if the intercept is close to zero and if the slope is close to one. In addition, to check how precise the estimated slope and intercept are, the confidence intervals for the slope and the intercept must be estimated.

Since both the slope and the intercept are considered in our problem, the joint confidence intervals for the slope and the intercept should be calculated. In this study, the Bonferroni method [5] is used. The Bonferroni method is very simple and guarantees joint confidence coefficient intervals of at least 2α when the slope and the intercept are separately estimated with confidence intervals for α. If the confidence intervals are small, the estimated value is more precise.

To decide the difference between the usage model and the test model, the hypothesis test is performed. In this study, we want to check that the usage and the test models are the same. To be the same, the slope must be one, and the intercept must be zero. Therefore, the null hypothesis and the alternative hypothesis for our problem are as follow:

$$H_o : \beta_0 = 0 \ and \ \beta_1 = 1,$$

$$H_a : \beta_0 \neq 0 \ or \ \beta_1 \neq 1$$

The values of t_0 *and* t_1 for the hypothesis test are computed as follows:

$$t_0 = \frac{a}{S_a}, \qquad t_1 = \frac{b-1}{S_b}$$

If both values are between $-t(1-\alpha/4;n-2)$ and $t(1-\alpha/4;n-2)$, the null hypothesis is accepted with $(1-\alpha)$ confidence, and it is concluded that the usage model and the test model are the same. This being the case, a software component can be reused without additional testing. Otherwise, it is concluded that the usage model and the test model are different, and retesting is required.

2.5. Test case generation of retest

When a software component needs additional testing for software reuse, it is possible to use existing test case generation methods. However, the problem with using the existing methods is that unnecessary testing may be performed because they do not use the previous testing information when generating test cases. If it is possible to use the previous testing information to generate test cases, then unnecessary testing can be avoided.

The purpose of this step is to minimize the differences between the usage model and the test model using a smaller number of new test cases. In this step, the random test case generation method is also used like the initial test case generation. However, the difference of each arc between the usage and the test models is used to generate test cases instead of the probability in the usage model. When the difference is used for testing, it is useful to find which areas need more testing, and more test cases are generated from the areas with high differences.

Generally, the differences *d* between the usage model *UM* and the test model *TM* are calculated by subtracting the frequencies of the arcs in the test model from the expected frequencies of the arcs in the usage model.

$$d_{i,j} = UM_{i,j} - TM_{i,j}$$

In this case, the differences can have three values, positive, negative, or zero, and they can have different meanings. The positive values mean that more tests are required in the arcs. The negative values show that too many tests have been performed on the arcs. The zero value means that the proper amount of tests has been performed on the arcs. Therefore, only positive values are needed to generate test cases because the areas with positive values need additional testing. However, the information about the arcs with negative values should be considered for the test case generation. To keep the order of the differences and to make all values of the differences positive, all differences

are subtracted by the smallest value. In this case, the smallest difference becomes zero, but this can cause a problem when generating test cases. When the value is zero, the arc with a zero difference is not used to generate test cases. Therefore, if the arc with a zero difference is only the arc from one state to another state, generating test cases might be problematic because some test case sequences cannot be completed. Therefore, to avoid this problem, we must add one to all the differences to eliminate the zero value. Therefore, the differences used in generating test cases are computed as follows:

$$d'_{i,j} = d_{i,j} - \min_{j}(d_{i,j}) + 1$$

After the differences for all arcs are calculated, these differences can be easily converted to probabilities and used to generate test cases in the same manner as the random test case generation with probabilities.

3. Case study

For the case study, double list com of the Ada 95 Booch components[http://www.pogner.demon.co.uk/components/bc/] was used. Booch components have 30 components including Bags, Graphs, Lists, Queues, Rings, Sets, Stacks, etc. We selected Double_List for the case study. It has 28 public, 4 generic and 12 private functions and procedures.

In the case study, first, an initial usage model had to be created. Once the initial usage model was created, then the initial test cases were generated. After that, the initial test was performed, and the initial test model was created using the test results. Once the initial usage and the initial test models were created, then the software component was tested for reuse.

3.1. Usage model creation

In this case study, the usage model was created using the operational mode [12], which is a formal characterization of the status of one or more internal data objects that affect software behavior.

In the double list Booch component, there are two operational modes, number_of_lists and null_list, and they have values as follow:

- no_of _lists = (0, 1, 2)
- null_list = (null_both, null_1, null_2, not_null)

Using the operational modes, states in the usage model are created by the combination of the operational mode values, and there are the following 10 states.

1. (Start) 2. (0)
3. (1, null_1) 4. (1, not_null)
5. (2, null_both) 6. (2, null_1)
7. (2, null_2) 8. (2, not_null)
9. (Error) 10. (Termination)

After the state creation, inputs should be defined. To define the inputs, the category partition method [7] was used. Using the category partition method, 65 valid and 66 invalid inputs were defined for the double list Booch component. These inputs were grouped in 26 groups, from A to Z. The following figure shows the usage model for the software component. In this model, uniform probabilities are used.

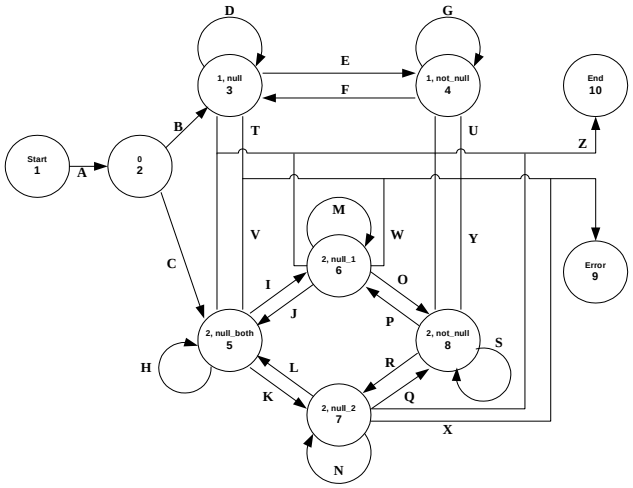

Figure 3. Usage Model for Double List

3.2. Initial test case generation

Because the testing history and the test model were not available for the Booch components, the initial test cases were generated using the random test case generation method. In this particular case, it generated 2588 test cases. Using these results, the initial test model is created.

3.3. Results analysis

The first step was to check to see if the simple linear regression method was appropriate for our problem, so the aptness of model test was performed. Second, to check the efficiency of the new method, the results of the new test case generation method were compared to the results of the random test case generation.

3.3.1. Aptness of Model

To check if the simple linear regression method was appropriate for our problem, the following four types of departures were considered using the residual analysis [5][9].

1. The regression function is not linear.
2. The error terms do not have constant variance.
3. The error terms are not independent.
4. The error terms are not normally distributed.

In addition to the residual analysis, ANOVA (Analysis of Variance) table was created and used to test if there was a linear relationship. The results were analyzed after 10,000 test cases were run, and both, the new method and the existing method, had no problem about using simple linear regression method. In this paper, the results of the analysis are described in detail.

3.3.2. Reducing differences

In the new approach, it is expected that the difference between the usage model and the test model is more efficiently reduced. Therefore, in this case study, the results of the new test case generation method are compared to the results of the random test case generation to show how efficiently the new approach is in reducing the difference. To compare the results, the intercept, and the slope, are monitored at every 2,000 test cases. These plots help to see how both the usage model and the test models become close in both methods. In this study, 90% confidence for the slope and the intercept is used.

Figures 4 and 5, display the change of the slope and the intercept for the new test case generation and the random test case generation.

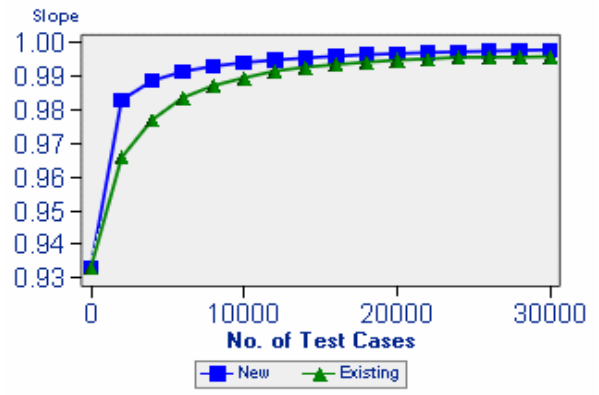

Figure 4. Slope

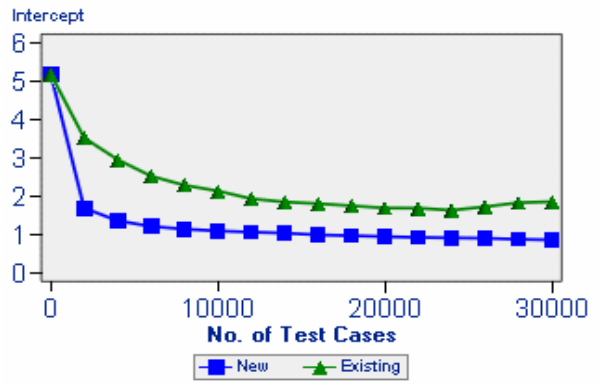

Figure 5. Intercept

Figure 6 illustrates the initial scatter plot for the test model and the fitted test model for the new test case generation method. Figure 7 and 8 show the scatter plots for the test model and the fitted test model for the new test case generation method and the random test case generation after 30,000 test cases.

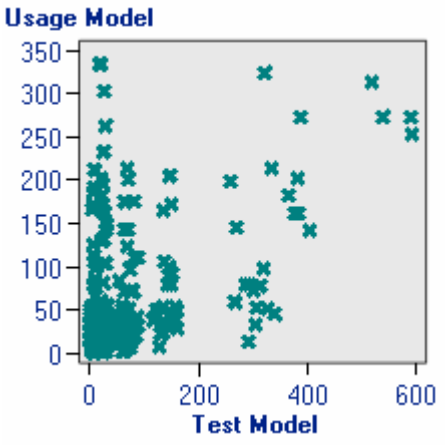

Figure 6. Initial

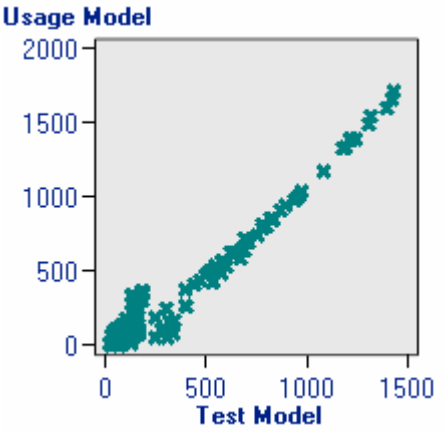

Figure 7. Scatter Plot for New Method

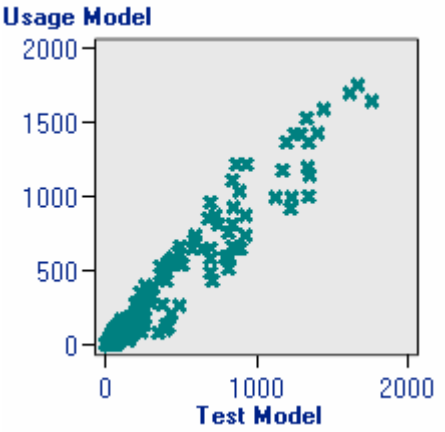

Figure 8. Scatter Plot for Existing Method

As shown in Figure 4, the values of the slope are getting close to one in both methods. However, the slope of the new method is always closer to one, and the new method is

faster in becoming close to one. In the new method, the slope reaches 0.99 after 6,000 test cases, but when random test case generation is used, the slope reaches 0.99 after 12,000 test cases. In addition, when the new method is used, the width of the confidence intervals is always smaller as displayed in Figure 5. It shows that when the new method is used, the estimated slope is more precise.

The figures 7 and 8 show that the data in both methods become straight-lines. It means that the test model becomes closer to the usage model when test cases are applie, but the data in new method is much closer to a straight-line than the data in the random test case generation.

In our study, the Bonferroni method is also used to compute the joint confidence intervals for stopping criterion. In our problem, we want to examine if the usage model and the test model are the same. Therefore, to stop testing, the following hypothesis testing is performed.

$$H_o : \beta_0 = 0 \ \ and \ \ \beta_1 = 1,$$

$$H_a : \beta_0 \neq 0 \ \ or \ \ \beta_1 \neq 1$$

When the new test case generation method is used, the null hypothesis is accepted after 2,000 test cases 90% confidence for the slope and the intercept. However, when the random test case generation is used, the null hypothesis is accepted after 12,000 test cases.

According to the results, it is possible to conclude that the new method is more efficient in reducing the difference between the usage model and the test model. Therefore, with smaller test cases, the test model becomes similar to the usage model when the new test case generation is used.

4. Conclusion

In this paper, the new approach for reusing a software component is developed to solve the problems in testing for software reuse. This new approach is based on studies of software testing using Markov chain [6][10][11]. First of all, the differences are used to generate test cases and to measure the stopping criterion of testing instead of the probabilities. In addition, multiple test models are used to keep the test history, and the average frequencies are used to update the test model.

The case study shows that the new approach needs fewer test cases to reach the acceptance point when comparing to the existing method. For both methods, the slope gets close to one, and the intercept is getting close to zero when test cases are applied. However, when the new method is used, the results are always better, and the estimated values are more precise.

This new approach will be useful for software engineers when they reuse software components. In this method, the statistical approach is used to decide if more testing is needed for software reuse based on the information of the usage model and the test model. In addition, when additional testing is required, the differences are used to generate test cases. Because all methods use mathematical approach, it is easy to automate. The automation will help engineers save time and effort in reusing software components.

References

[1] Basili, Victor R., Briand, Lionel C., and Melo, Walcelio, How reuse influences productivity in object-oriented system, Communication of the ACM 39 (10), 104-116, 1996.

[2] Becker, Shirley A. and Whittaker, James A., Cleanroom Software Engineering Practices, IDEA Group Publishing, 1997.

[3] Beizer, B., Software System Testing and Quality Assurance, International Thompson Publishing Inc., New York, 1996.

[4] Harnett, Donald L., Statistical Methods Third Edition, Addison-Wesley Publishing Company, 1982.

[5] Neter, John, Wasserman, William, and Kunter, Michael H., Applied Linear Statistical Models 2nd Ed., Richard D. Irwin Inc, 1985.

[6] Oshana, Robert, Software Testing with Statistical Usage Based Models, Embedded Systems Programming, January, 1997.

[7] Ostrand, Thomas J and Balcer, Marc J., The Category-Partition Method for Specifying and Generating Functional Tests, Communication of ACM 31(6), 676-686, 1988.

[8] Pressman, R., Software Engineering: A Practitioner's Approach 3rd Ed., McGraw-Hill, New York, 1992.

[9] Rekab, Kamel, Design of Experiments Made Easy, Florida Institute of Technology, 1998.

[10] Whittaker, James A., Markov Chain Techniques for Software Testing and Reliability Analysis., Ph.D. Thesis, University of Tennessee, Knoxville, 1992.

[11] Whittaker, James A. and Thomason, Michael G., A Markov Model for Statistical Software Testing, IEEE Transaction on Software engineering 20 (10), 812-824, 1994.

[12] Whittaker, James A., Stochastic software testing, Annals of Software Engineering 4, 115-131, 1997.