EXTRACTING THE MINIMIZED TEST SUITE FOR REVISED SIMULINK/STATEFLOW MODEL

Han Gon Park[1], Geon Gu Park[1], Kihyun Chung[1] and Kyunghee Choi[2]

[1]Department of Electronic Engineering, Ajou University, Suwon, Republic of Korea
[2]Department of Computer Engineering, Ajou University, Suwon, Republic of Korea

ABSTRACT

Test case generation techniques are successfully employed to generate test cases from a formal model. A problem is that as the model evolves, test suites tend to grow in size, making it too costly to execute entire test suites. This paper aims to propose a practical approach to reduce the size of test suites for modified Simulink/Stateflow (SL/SF) model, which is popularly used for modeling software behavior in many industries like automobile manufacturers. The model for describing a system is frequently modified until it is fixed. The proposed technique is capable of extracting the minimized sized test suite in terms of test coverage, by taking into account both the modified and the affected portion of revised SL/SF model. Two real models for the ECUs deployed in a commercial car are used for an empirical study.

KEYWORDS

Test Suite Reduction, Simulink/Stateflow Model, Test Case Generation, Model Based Testing

1. INTRODUCTION

Software grows with time. The specification is modified for uses beyond the previous one, the number of functions increases, the lines of code become longer, and many other complications come with the growth of software. Some old portions are modified, some obsolete portions are deleted, and some new portions are added. Due to such inevitable maintenance activities, more than 70% of the time spent on software maintenance is in modifying and retesting the software. To reduce the cost introduced by testing the revised software, efficient testing techniques are definitely needed. The activity to revalidate the modified software is called regression testing. Reference [1] identifies regression testing as being of two types: progressive regression testing, and corrective regression testing. Progressive regression testing involves a modified specification, while corrective regression testing does not perform specification change, but specification correction. Most techniques focus on progressive testing, as do we in this paper.

Regression testing aims to ensure that no new faults are introduced into the previously validated software due to the modification. The regression testing also needs to confirm that the newly added requirements are properly integrated into the revised software. This is highly expensive, time consuming, and laborious work. One main factor that makes regression testing expensive is having to execute the whole set of test cases generated by the test generation method. A research focus of regression testing techniques is to extract an effectively minimized sub set of test cases from the whole test suite.

The test suite minimization problem is to select a subset of the entire test suite that can detect all the faults. For the minimization, the test minimization techniques not only remove the test cases from a suite that have become redundant with respect to a particular criterion or system output, but they also need to add test cases that validate the newly added features. Coverage techniques

[2] select test cases that cause a modified program to satisfya criterion (depending on the coverage criterion) that is different from that of the original program, while safe techniques [3] select test cases to produce different output.

Since model-based regression testing of software has several advantages [4], our work is to present a Simulink/Stateflow (SL/SF) model-based regression testing technique. We propose a test case minimization for modified programs, of which the test cases are generated from an SL/SFmodel [5].SL/SF has been popularly utilized in model-based development in many industries, such as automobile manufacturers. They use SL/SF for modeling system behaviors, describing specifications, producing auto codes [6], generating test cases [5], and utilizing the model for testing systems [7].Extracting the minimized test suite from generated test casesbased on an SL/SF model for the modified program is the focus of this study. The proposed technique belongs to coverage technique.

Many previous coverage based test case selection techniques focus on removing test cases from a test suite in such a way that redundant test cases are eliminated. The eliminated test suite should hopefully contain the test cases for all the obsolete requirements, and the portions affected by the obsolete requirements. If the specification is changed with some additional requirements, the model has to be modified to include the additional requirements, and the test suite generated from the revised model includes requirements for the added parts. In addition to the new test cases, the modified software also needs to be tested by the test cases for the portions affected by the added requirements.

To find the test cases for obsolete, added, and affected portions of a SL/SF model, we simulate the old model and the revised model with test suites generated from the models. In simulating SL/SF, it is possible to know whether each transition is evaluated as TRUE or FALSE, and which state is active. Most test items consist of transition and state evaluation information in SL/SF models. Two test suites generated from the original and modified models are applied to each model, and every test case that covers a specific test item (for example, a state or transition condition) of a model, but is not utilized to cover the test target of the other model, belongs to the minimized test suite. The minimized test suite consists of the test cases found during the simulation.

To confirm the effectiveness of the proposed technique, we conducted an experiment. As a system-under-test, we used models for the Intelligent Management Module (IMS) and the Drive Door Module (DDM) of a commercial passenger car provided by a major world class automobile manufacturer. The result demonstrates that the proposed technique can be used to find the minimized test suite for a revised SL/SF model that would otherwise be tested by the entire test suite.

In the remainder of the paper, Section2 reviews the relevant literature, while Section 3 addresses the proposed technique in detail. Section 4 describes the experiment results, while the Conclusion summarizes the research.

2.RELATED WORK

Many approaches relating to regression test suite minimization have been published. Researchers categorized regression test minimization techniques in different ways. One possible categorization is test case minimization, test suite prioritization, and test suite selection.

The survey in Ref. [8] contained various minimization, selection, and prioritization techniques, and discussed their open problems. Reference [9]used a combination of static slicing and delta debugging that automatically minimizes the sequence of failure-inducing method calls, and showed in a case study that the strategy could minimize failing unit test cases by 96%on average. Reference [10] used additional criterion to break ties in the minimization process if there was more than one test case with equal importance to a suite.

Reference [11] proposed a new metric to assess the rate of fault detection of prioritized test cases that incorporate varying test cases and fault costs; they also presented the results of a case study illustrating the application of the metric. Reference [12] presented results from an empirical study of the application of several greedy, meta-heuristic, and evolutionary search algorithms that belong to the prioritization technique category.

Reference [13] presented an approach to regression testing that handles the selection of test cases from the existing test suite that should be rerun, and identification of the portions of the code that must be covered by test cases. Both tasks were performed by traversing graphs for the program and its modified version. Rothermel etal.[14] suggested a regression test selection technique for use with object-oriented software; the technique constructs graph representations for software, and uses these graphs to select test cases. Reference[15] presented a methodology and a tool to support test selection from regression test suites based on change analysis in object-oriented designs. Reference[16] proposed a technique to select only a fraction of the test cases from the entire test suite to revalidate an object-oriented software system. Some researchers have proposed model based regression approaches. Reference [17] proposed a test case generation technique for UML designs using the symbolic execution method. Reference [18] presented a safe regression technique that used various UML diagrams. Chen et al. [19] proposed a regression test selection technique using an activity diagram of UML. Reference [20] presented a method to select the Test Dependency Graph subset that contains a modified portion. Reference[21] shows that the size of the specification based test-suites can be dramatically reduced, and that the fault detection of the reduced test suites is adversely affected.

Beydeda etal.[22] suggested a technique for class level regression testing based on specification and implementation information;regression test cases were selected by comparing two different versions of a model described by a class specification implementation graph (CSIG).Reference [23]reduced test cases for the EFSM (Extended Finite StateMachines) model based on dependence analysis, and identified the difference between the original and modified model based on the elementary modifications.

3.THE PROPOSED TECHNIQUE

This section presents a minimum test suite extraction technique for revised model in SL/SF model based test case generation. Let M_o be the model of a specification S_o, and M_m be the model for the specification S_m modified from S_o. In coverage based test case generation, the test suites TC_o and TC_m are generated from M_o and M_m, respectively, and the test suites are partially or fully adequate with respect to a coverage criterion, crt. The test coverage is defined as N1/N2, where N1 is the number of test items the test suite can cover, and N2is the number of test items in a model. A test item is a specific condition to be evaluated as TRUE or FALSE to satisfy crt. For example, when the test criterion, crt is transition coverage, all transitions in a SL/SF model become the test items.

S_m consists of the modified portion (MP), affected portion (AP), and unchanged portion (UP), compared with S_o. MP is the portion changed directly by the modification. MP may be introduced by modification, deletion and/or addition of specification. AP is the portion that is not changed directly, but affected by MP. AP may or may not appear, depending on MP. UP is neither AP nor MP. M_m is modified from M_o to adopt the modified specification.

For the system with the modified specification, the minimum test suite TC_{min} includes the test cases to cover only MP and AP. But those for UP should not be included in TC_{min}, since the test items for UP have already been tested during testing M_o with TC_o. With TC_{min}, the test coverage of M_m is preserved. Additionally the items deleted from M_o and their affected test items should be covered. Usually it is not hard to find the test cases for CPin coverage based test case generation. But it is not an easy job to extract the test cases for AP. The proposed technique aims to find TC_{min} for the SL/SF model.

In SL/SF, when states and transitions are added, deleted and/or modified, M_o and M_m behave differently. It is possible to extract the test items adequate to a certain criterion crt in the SL/SF model, where the test items are constructed with states or/and transitions. Let the test item set for M_i be ITM_i. Figure 1 shows the basic idea of our technique, which follows. $M_m$and M_o are simulated with both TC_o and TC_m. When simulating the models, there are several cases we need to consider.

1) If a test item itm_i appears in M_o but does not in M_m, itm_i is one of the test items that has been deleted during modelling M_m to adopt the modification. The test case t_i for itm_i is needed to check whether $M_m$has in fact accurately deleteditm$_i$ from M_o. Thusif t_i is in TC_o,t_i is included in TC_{min}. If t_i is not in TC_o, there is no way to include t_i in TC_{min}. This is the case that $TC_o$is not fully adequate to crt,and $t_i$has not been generated with the test case generation method.
2) A test item itm_i may appear in both M_o and M_m. If t_i in TC_ocovers itm_i in M_o, but does not cover the test item in M_m, $item_i$ is an item either modified or affected by the modification. Thus t_i is included in TC_{min}.
3) If a test item itm_j appears in M_m, but does not in M_o, itm_j is one of the test items added during modeling M_m. The test case t_j for itm_j is needed to check whether M_m accurately has addeditm$_j$ to M_o;and if t_j is in TC_m,t_j is included in TC_{min}.
4) itm_j may appear in both M_o and Mm. In this case, if t_j in TC_mcovers itm_j in M_m, but does not cover the test item in M_o, $item_j$ is an item that is either modified or affected by the addition. If t_j is not in TC_{min},t_j is included in TC_{min}.

```
find_minimum_test_suite ($M_o$,$M_m$,$ITM_o$,$ITM_m$,$TC_o$,$TC_m$)
    $TC_{min}$=empty;
    for each $itm_i$ in $ITM_o$
        if ($item_i$ is not in $ITM_m$)
            $TC_{min} \leftarrow t_i$
        else if (($item_i$ is in $ITM_m$) and
                (($t_i$does not cover$item_i$ in $ITM_m$)and ($t_i$ covers $item_i$ in $ITM_o$))
                $TC_{min} \leftarrow t_i$
    for each $itm_j$ in $ITM_m$
        if ($item_j$ is not in $ITM_o$),
            $TC_{min} \leftarrow t_j$
            else if ($item_j$ is in $ITM_o$) and
                (($t_j$does not cover$item_j$in $ITM_o$) and ($t_j$ covers $item_j$ in $ITM_m$))
                if ($t_j$ is not in $TC_{min}$)
                    $TC_{min} \leftarrow t_j$
return$TC_{min}$
```

Figure1. The proposed technique in pseudo code

Figures 2 and 3 show the SL/SF models used to explain the proposed technique in detail. Figure 2 shows the original model, and Figure3 its modified version. The models have two inputs, binary A and B. State "Active_3" and two transitions R_8 and R_9 between "Active_3" and "Active_1" are deleted from M_o, state "Active_4" is added with two new transitions R_{10} and R_{11} between "Active_4" and "UnActive",and R_5 is modified.

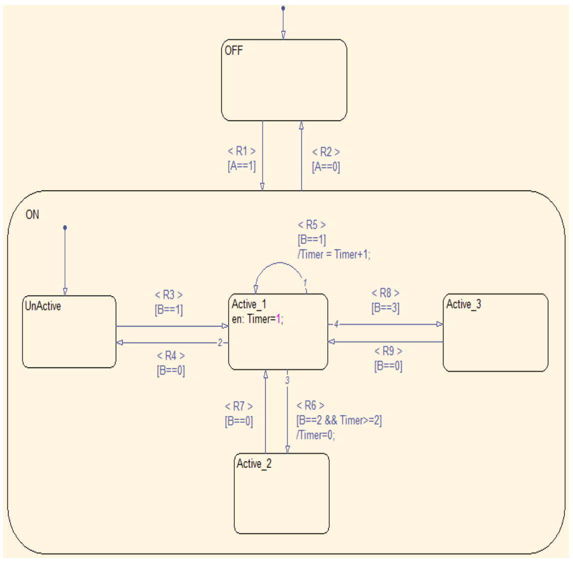

Figure2. The original model M_o.

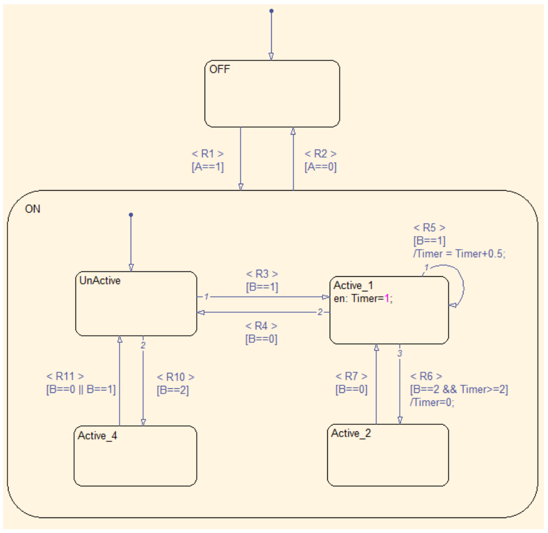

Figure3. The modified model M_m.

If the test cases are generated with respect to transition coverage, all transitions in the models become the test items.A test case, t_i is an input sequence to evaluate the transition, R_i as TRUE. With t_i,the model reaches the destination state of R_i from the initial state, OFF. We assume each test case starts from the initial state, as in many actual applications. The inputs are initially set to '0'. That is, (A, B)=(0,0). Tables 1 and 2 show possible TC_o and TC_m, respectively, that are fully adequate to transition coverage.

In real test case generation, the test cases are optimized after or during test case generation. By 'optimization' we mean that if a test case t_p in a test suite is a subset of another test case t_q in the same test suite, t_p is excluded from the suite. After optimization, the test cases for R_1, R_3, R_5, R_6,

R_8need to be excluded, and four test cases should remain in the test suite. But in this example and the next empirical study, we do not optimize test suites, since the effect of test minimization after the optimization is not clearly shown. Thus we assume that one test case is generated for each test item.

Table1. Test cases in TC_o.

Test Item (Transition)	Test case
R_1	{(0,0),(1,0)}
R_2	{(0,0),(1,0),(0,0)}
R_3	{(0,0),(1,0),(1,1)}
R_4	{(0,0),(1,0),(1,1),(1,0)}
R_5	{(0,0),(1,0),(1,1),(1,1)}
R_6	{(0,0),(1,0),(1,1),(1,1),(1,2)}
R_7	{(0,0),(1,0),(1,1),(1,1),(1,2),(1,0}
R_8	{(0,0),(1,0),(1,1),(1,3)}
R_9	{(0,0),(1,0),(1,1),(1,3),(1,0)}

Table2. Test cases in TC_m.

Test Item (Transition)	Test case
R_1	{(0,0),(1,0)}
R_2	{(0,0),(1,0),(0,0)}
R_3	{(0,0),(1,0),(1,1)}
R_4	{(0,0),(1,0),(1,1),(1,0)}
R_5	{(0,0),(1,0),(1,1),(1,1)}
R_6	{(0,0),(1,0),(1,1),(1,1),(1,1),(1,2)}
R_7	{(0,0),(1,0),(1,1),(1,1),(1,1),(1,2),(1,0}
R_{10}	{(0,0),(1,0),(1,2)}
R_{11}	{(0,0),(1,0),(1,2),(1,0)}

First, the test items in ITM_o are compared with those in ITM_m. R_8 and R_9 in ITM_o are not in ITM_m. Thus $t_{8,o}$ and $t_{9,o}$, the test cases for R_8 and R_9 in M_o, are inserted in TC_{min}. $t_{m,n}$ is defined as the nth test case in model M_n. The two cases are needed to check the correctness of deletion. R_1 through R_7 appear in both M_o and M_m. By simulating the models with TC_o,R_1 through $R_5$are found covered by $t_{1,o}$ through $t_{5,o}$in both models. But R_6 and R_7 in M_m are not evaluated as TRUE with $t_{6,o}$ and $t_{7,o}$; mean while, those in $M_o$are covered. This means that there is one or more modifications and/or affected portion sin R_6 and R_7 by deleting R_8 and R_9. In this case, there is an effect. Thus $t_{6,o}$ through $t_{7,o}$ are put in TC_{min}. Table 3 shows the simulation result of M_m with TC_o. The simulation result of M_o is not summarized, since all items are covered.

Now compare the test items in ITM_m, with those in ITM_o. R_{10} and R_{11} in ITM_m do not appear in ITM_o. Thus $t_{10,m}$ and $t_{11,m}$, which are needed to check the specification addition, are inserted into TC_{min}. The result of simulating the models with TC_m reveals that all test items in both models are satisfied. Thus, no extra test cases are inserted intoTC_{min}. Finally,TC_{min} for the model in Figure3 is determined as $\{t_{8,o}, t_{9,o}, t_{6,o}, t_{7,o}, t_{10,m}, t_{11,m}\}$.Table4 shows the result of simulating M_o with TC_m. The simulation result of M_m is not included.

Table3. Result of simulating M_m with TC_o.

Test Items		TC_o		Criterion satisfaction
ITM_o	ITM_m	Test cases	$t_{i,j}$	
R_1	R_1	{(0,0),(1,0)}	$t_{1,o}$	O
R_2	R_2	{(0,0),(1,0),(0,0)}	$t_{2,o}$	O
R_3	R_3	{(0,0),(1,0),(1,1)}	$t_{3,o}$	O
R_4	R_4	{(0,0),(1,0),(1,1),(1,0)}	$t_{4,o}$	O
R_5	R_5	{(0,0),(1,0),(1,1),(1,1)}	$t_{5,o}$	O
R_6	R_6	{(0,0),(1,0),(1,1),(1,1),(1,2)}	$t_{6,o}$	X
R_7	R_7	{(0,0),(1,0),(1,1),(1,1),(1,2),(1,0}	$t_{7,o}$	X
R_8	-	{(0,0),(1,0),(1,1),(1,3)}	$t_{8,o}$	NA
R_9	-	{(0,0),(1,0),(1,1),(1,3),(1,0)}	$t_{9,o}$	NA

Table4. Result of simulating M_o with TC_m.

Test Items		TC_m		Criterion satisfaction
ITM_o	ITM_m	Test cases	$t_{i,j}$	
R_1	R_1	{(0,0),(1,0)}	$t_{1,m}$	O
R_2	R_2	{(0,0),(1,0),(0,0)}	$t_{2,m}$	O
R_3	R_3	{(0,0),(1,0),(1,1)}	$t_{3,m}$	O
R_4	R_4	{(0,0),(1,0),(1,1),(1,0)}	$t_{4,m}$	O
R_5	R_5	{(0,0),(1,0),(1,1),(1,1)}	$t_{5,m}$	O
R_6	R_6	{(0,0),(1,0),(1,1),(1,1),(1,1),(1,2)}	$t_{6,m}$	O
R_7	R_7	{(0,0),(1,0),(1,1),(1,1),(1,1),(1,2),(1,0}	$t_{7,m}$	O
-	R_{10}	{(0,0),(1,0),(1,2)}	$t_{10,m}$	NA
-	R_{11}	{(0,0),(1,0),(1,2),(1,0)}	$t_{11,m}$	NA

4.EMPIRICAL STUDY

The proposed technique is verified using two ECU (Electrical Control Unit) SL/SF models for a commercial vehicle provided by a world class automobile manufacturer. One ECU is the Intelligent Management System (IMS), which is used for managing the ECUs for doors, mirrors, and wipers. The other is the driver side mirror ECU (DDM). The IMS model is simple, and has 34 states and 51 transitions; meanwhile, the mirror model is relatively complicated, and has 49 states and 135 transitions.

For the study, we use the models modified according to the three different modification types: transition modification, state/transition addition, and state/transition deletion. The modifications were not made arbitrarily, but were made while keeping the precise specifications of the car. First, the test suites for the original IMS (TC_{OI}), modified IMS (TC_{MI}), original DDM (TC_{OD}), and modified DDM (TC_{MD}) models were generated with a commercial test case generator, TCG [24]. The generated test suites were partially adequate to transition coverage, due to various reasons, such as incomplete model, poor performance of TCG, and unknown reasons. We performed an empirical study with the test suites. Since the test case generation is beyond the scope of this paper, we do not discuss the test case generation from the model. Using the original and modified models, and their test suites, the proposed technique tries to find the minimized test suites. The test cases in TC_{min} are compared with those of minimum test cases analyzed by the inspection with respect to the number of test cases for the modification itself (Direct in Table5), and that for the affected portion (Affected in Table5).

Table5 is a summary of the study. In the IMS model, one transition is modified among 51 transitions in the Modification type. Our technique finds TC_{min} has just one test case for Direct test cases, but no Affected test cases. This indicates that there is no portion affected by the modification. We confirm it by inspecting the model. One state and the three related transitions are added in the Addition type. It is found that TC_{min} includes three Direct test cases needed to check the modified transitions. As in the Modification type, no extra Affected test cases are needed. In the Deletion type experiment, one state and the connected three transitions are deleted from IMS_o. Like the previous two experiments, the test suite includes only three Direct test cases to check the deleted transitions. For IMS, TC_{min} consists of only Direct test cases in all three modification types. We confirmed that the test suites were minimum, and sufficient to cover the model change. Since the IMS model is small, and consists of seven almost independent sub-models that don't affect each other, the modification does not affect other parts, and TC_{min} contains the test cases to cover only the modified transitions.

The DDM model is relatively complicated, and some parts are closely related to each other. In the Modification type experiment, two out of 135 transitions are modified. By the modifications, 36 transitions are affected. The proposed technique finds two Direct and 36 Affected test cases for TC_{min}. Three transitions are added, and the added transitions affect three other transitions in the Addition type. In this case, the technique finds the minimized test suite that includes six test cases. In the Deletion type, three transitions are deleted with one state. In this case, TC_{min} consists of three Direct and three Affected test cases. TC_{min} is minimum in all three modification types.

Table5. Summary of the empirical study.

Model	Modification Type	No. of modified test items	No. of minimum test case (No. of found test cases)	
			Direct	*Affected*
IMS	*Modification*	1	1(1)	0
	Addition	3	3(3)	0
	Deletion	3	3(3)	0
DDM	*Modification*	2	2(2)	36(36)
	Addition	3	3(3)	3(3)
	Deletion	3	3(3)	3(3)

The empirical study is limited in terms of the number of models and modification complexity. But we believe that it demonstrates the effectiveness of the proposed technique, even though it is not sufficient to fully verify the performance.

Sometimes test engineers are forced to test a modified system with test cases only for the modified portion because of limited test time, losing the coverage. Without the reduced test suite, it is inevitable that the modified model has to be tested with the test suite with all test cases generated by the test case generator to preserve the original coverage. Even with the test cases, it is not possible to test the deleted portion that does not appear in the modified model. Table6 summarizes the statistics of test cases generated by TCG, and the minimized test cases found by the proposed technique. Even though the test case reduction surely varies with model and modification, the statistics demonstrate that the reduction in the number of test cases may be significant in some applications, while preserving the coverage of the full test suite.

Table6.Test case statistics.

Model	Modification Type	No. of test cases (TCG)	No. of minimized test cases (the proposed technique)
IMS	*Modification*	49	1
	Addition	52	3
	Deletion	46	3
DDM	*Modification*	102	38
	Addition	105	6
	Deletion	99	6

5.CONCLUSIONS

We have described a minimized test case extraction technique for a modified SL/SF model. The proposed technique is capable of extracting a minimized sized test suite by taking into account both the modified and the affected portion of the revised model. For the empirical study, we used two models for the ECUs deployed in a commercial vehicle. The result of our empirical study shows that the proposed technique achieved a significant reduction in terms of test cases. It extracts just the test cases needed for testing the portion modified from the original model, and those affected by the modification. The cost of testing was dramatically reduced for the models used in this study, even though the performance and cost reduction are surely dependent on the models and modifications. Although we cannot broadly generalize our results, and further studies are needed, the experiment indicates that the proposed technique may be an effective means of reducing testing effort.

ACKNOWLEDGEMENTS

This Research was supported by Defense Acquisition Program Administration(UD150042AD).

REFERENCES

[1] H. Leung and L. White, (1990) “A Study of Integration Testing and Software Regression at the Integration Level”, Proceedings of Conference on Software Maintenance,DOI:10.1109/ICSM.1990.131377, pp290-301.

[2] D. Nardo, N. Alshahwa1, L. Briand, (2013) “Coverage-Based Test Case Prioritisation: An Industrial Case Study”, Proceedings of IEEE Sixth International Conference on Software Testing.

[3] H. Agrawal, J.R. Horgan, E.W. Krauser and S. London, (1993) “Incremental Regression Testing”, Proceedings of IEEE Conference on Software Maintenance, DOI: 10.1109/ICSM.1993.366927, pp348-357.

[4] D. Deng, P.C.Y. Sheu and T. Wang, (2004) “Model-based Testing and Maintenance”,Proceedings of IEEE Sixth International Symposium on Multimedia Software Engineering, DOI: 10.1109/MMSE.2004.51,pp278-285.

[5] C.S. Pasareanu, J. Schumann, P. Mehlitz, M. Lowry, G. Karsai, H. Nine and S. Neema,(2009) "Model Based Analysis and Test Generation for Flight Software", Proceedings of IEEE Third International Conference onSpace Mission Challenges for Information Technology, DOI: 10.1109/SMC-IT.2009.18, pp83-90.

[6] Lutz Köster, Thomas Thomsen, and Ralf Stracke, (2001) “Connecting Simulink to OSEK: Automatic Code Generation for Real-Time Operating Systems with TargetLink”, SAE Technical Paper.

[7] Reactis, <http://www.reactive-systems.com/papers/bcsf.pdf>.

[8] S. Yoo and M. Harman, (2012) “Regression Testing Minimization Selection and prioritization: Survey”, Software TestingVerification & Reliability, Vol. 22, Issue 2, DOI: 10.1002/stvr.430, pp67-120.

[9] A. Leitner, M. Oriol, and A. Zeller, (2007) “Efficient Test Case Minimization”, Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, DOI: 10.1145/1321631.1321698, pp417-420.

[10] J. Lin, C. Huang and C. Lin, (2008) “Test suite reduction analysis with enhanced tie-breaking techniques”, Management of Innovation and Technology, Proceedings of IEEE4th International Conference on, DOI: 10.1109/ICMIT.2008.4654545, pp1228-1233.

[11] S. Elbaum, A. Malishevsky, and G. Rothermel, (2001) “Incorporating Varying Test Costs and Fault Severities into Test Case Prioritization”,Proceedings of the 23rd International Conference on Software Engineering,DOI: 10.1109/ICSE.2001.919106, pp329-338.

[12] Z. Li, M. Harman and R. Hierons, (2007) “Search Algorithms for Regression Test Case Prioritization”, Proceedings of the IEEE Transactions on Software Engineering, Vol. 33, Issue 4, DOI: 10.1109/TSE.2007.38, pp225-237.

[13] G. Rothermel and M. Harrold, (1994) “Selecting tests and identifying test coverage requirements for modified software”, Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis,DOI: 10.1145/186258.187171, pp169-184.

[14] G. Rothermel, M. Harrold and J. Dedhia, (2000) “Regression test selection for C++ software”, Software Testing Verification and Reliability, Volume 10, Issue 2, DOI: 10.1002/1099-1689(200006)10:2<77::AID-STVR197>3.0.CO;2-E, pp77-109.

[15] L.C. Briand, Y. Labiche and S. He, (2009) “Automating regression test selection based on UML designs”, Information and Software Technology, Volume 51, Issue 1, DOI: 10.1016/j.infsof.2008.09.010, pp16-30.

[16] P. Hsia, X. Li, D. Kung, C. Hsu, L. Li, Y. Toyoshima and C. Chen, (1997) “A technique for the selective revalidation of object-oriented software”, Journal of Software Maintenance: Research and Practice, Volume. 9, Issue 4, DOI: 10.1002/(SICI)1096-908X(199707/08)9:4<217::AID-SMR152>3.0.CO;2-2., pp217-233.

[17] L. Briand, Y. Labiche, and T. Yue, (2009) “Automated Traceability Analysis for UML Model Refinements”, Information and Software Technology, Volume 51, Issue 2, DOI: 10.1016/j.infsof.2008.06.002, pp. 512-527.

[18] L. Briand,Y. Labiche, and G. Soccar, (2002)“Automating Impact Analysis and Regression Test Selection Based on UML Designs”,Proceedings. International Conference on Software Maintenance, DOI: 10.1109/ICSM.2002.1167775, pp. 252-261.

[19] Y. Chen, R.L. Probert, and D.P. Sims,(2002) “Specification-based Regression Test Selection with Risk Analysis”, Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research, DOI: 10.1145/782115.782116, pp1.

[20] Y. Traon, T. Jeron, J.M. Jezequel, and P. Morel, (2000) “Efficient object-oriented integration and regression testing”, Proceedings of the IEEE Transactions on Reliability, Vol. 49, Issue 1, DOI: 10.1109/24.855533, pp12-25.

[21] M. Heimdahl and D. George, (2004) “Test-Suite Reduction for Model Based Tests: Effects on Test Quality and Implications for Testing”, Proceedings of the 19th IEEE international conference on Automated software engineering, DOI: 10.1109/ASE.2004.67, pp176-185.

[22] S. Beydeda and V. Gruhn, (2001) “Integrating White- and Black-Box Techniques for Class-Level Regression Testing”, Computer Software and Applications Conference, COMPSAC 2001. 25th Annual International, DOI: 10.1109/CMPSAC.2001.960639, pp357–362.

[23] B. Korel, L.H. Tahat, and B. Vaysburg, (2002) “Model Based Regression Test Reduction Using Dependence Analysis”, Proceedings. International Conference onSoftware Maintenance, DOI: 10.1109/ICSM.2002.1167768, pp214-223.

[24] Btstech, <http://www.btstech.co.kr/page_vaHk97>.

AUTHORS

Han Gon Park holds a Master Degree (M.S) in Electronic Engineering from Ajou University, Republic of Korea. His areas of research interest includes Embedded System Testing, Model Based Testing. He is currently working at LG Chem.

Geon Gu Park is a Master's course of Electronic Engineering from Ajou University, Republic of Korea. His areas of research interest includes Embedded System Testing, Model Based Testing.

Kihyun Chung holds a Doctoral Degree (Ph.D) in Electronic Engineering from Purdue University, USA. His areas of research interest includes Computer Architecture, VLSI, Real time/ Multimedia System. At present he is working as Professor, Electronic Engineering, Ajou University, Republic of Korea.

Kyunghee Choi holds a Doctoral Degree (Ph.D) in Computer Engineering from Paul Sabatier University, France. His areas of research interest includes Operating System, Real time/ Multimedia System. At present he is working as Professor, Computer Engineering, Ajou University, Republic of Korea.