Research Article

Requirement-based Selection Model for the Expansion of Regression Test Selection

Adtha Lawanna*
Department of Information Technology, Vincent Mary School of Science and Technology, Assumption University, Samutprakarn, Thailand

* Corresponding author. E-mail: adtha@scitech.au.edu DOI: 10.14416/j.ijast.2017.08.005
Received: 15 November 2016; Accepted: 16 March 2017; Published online: 15 August 2017

Abstract
Issue of maintaining software is to consider which test cases should be kept for the next modification where the size of test suite gets bigger. This makes performance of software development pull out. The objective of proposing requirement-based test case selection model is to improve ability of regression test selection, in particular, to moderate the size of test suite of the modified program, which gets larger after modification regarding the need of specific requirements, including preparing higher ability of removing faults. It comprises five main algorithms, which are finding reused test case, classifying, revising, deleting, and selecting the appropriate test cases. This paper uses six programs run on different four comparative studies, which are select-all, random, and regression test selection. It gives smaller size than the traditional techniques about 49.78% in average. Besides, it offers percent fixing faults that is higher than select-all, random, and regression test selection algorithm as around 0.06–1.32%.

Keywords: Test case, Test suite, Selection, Regression test, Requirement

1 Introduction

This article presents problem of maintaining software in terms of reducing numbers of selected test cases that are used as the representative of the modified program and avoiding faults that could be produced during the processes are running [1]. Also, there are some solutions provided for solving these issues, including the proposed model that is developed for the reason of improving the traditional methods particularly, the use of regression test selection to offer the better results [2]. In the past, using retest-all technique is the first method to evaluate the modified program by retesting all test cases, which gives no problem when dealing with small test suite [3]. In fact, when modifying software for several time can make size of test suite gets bigger until execution time becomes one of the problem of the whole processes of software maintenance [4]. Besides, random technique is available for the bigger size [5]. But the precision of this cannot be ensured after selection [6]. Regression testing is method for executing a set of test cases on a program, which guarantees its modification doesn't produce programming errors and preserve the efficiency of the system not less than it has been in the past [7], [8]. The principle of this concept is to build modified program from previously by generating suitable test suites [9]. It's happen when changing request is involved. This means adding new test cases into the previous test suites [10]. Accordingly, deleting the obsolete test cases become the first priority. Therefore, test suite maintenance will be the objective of regression testing [11]. Irrelevant test cases, e.g., obsolete, uncontrollable, and redundant test cases are removed regardless this reason. The problem of this technique is that it affects the cost-effective of regression testing, which involves its algorithms. This makes time consuming

Please cite this article as: A. Lawanna, "Requirement-based selection model for the expansion of regression test selection," *KMUTNB Int J Appl Sci Technol*, vol. 10, no. 3, pp. 223–230, Jul.–Sep. 2017.

to execute the test cases. Moreover, considering detecting faults and factors that affect ability of test selection techniques are realized [12]. Therefore, the experimental design will concern independent variable such as seven subject programs, regression test technique, and test suite generation criteria, including dependent variable, e.g., the ability of size reduction and fault detection are involved. The proposed model is given for increasing efficiency and effectiveness of using retest-all, random, and regression test selection by three evaluations, which are reducing the amounts of the selected test cases, fault detection, and percent fixing bugs or programming errors.

2 Materials and Methods

2.1 *Subject programs*

GZIP, GREP, FLEX, MAKE, BASH, VIM, and SED used in this paper can be downloaded from Software-artifact Infrastructure Repository (http://sir.unl.edu/php/previewfiles.php) as shown in Table 1. The amounts of the test suite are around 5,680–122,169. The first four programs are unix utilities provided by Gnu site that pass several-release versions. They involve treatment of smokers who want to quit smoking from different sources. Each program is created to find factors that can help them from that problem. Providers conclude that these programs cannot increase code coverage because of the measurement at statement level is poor. This is the reason why this paper uses them as a part of studies. For BASH, VIM, and SED, they written by C-Language and test type is ad-hoc that are served as regression suites for subsequent version.

Table 1: Subject program

Program	Item	*T*
GZIP	β_1	5,680
GREP	β_2	10,068
FLEX	β_3	10,459
MAKE	β_4	35,545
BASH	β_5	59,846
VIM	β_6	122,169
SED	β_7	14,427

2.2 *Test case selection*

Before, starting to explain the concept of test case selection, it is necessary to understand two definitions listed as follows;

Definition 1: Test Suite

Test suite (T) or test pool is a set of test cases created for each program (P). After the modification, T will be denoted as T' and P is changed to P'.

Definition 2: Test Case

Test cases are member of T, which says $T = \{t_1, t_2, t_3,...t_n\}$, while $T' = \{t'_1, t'_2, t'_3,...t'_n\}$. Therefore, P gives T and P' offers T'. Under the concept of test case selection, T' can be generated by specific tools such as test case generator. In this paper, T' is created by using classification tree generator. However, this paper doesn't show the details of generating T' because we focus on selection technique, which concerns different algorithms. Test case selection is technique that can reduce the size of T'. It implies determining $\exists t'$ from large T'. Example of test case selection are integer programming, slicing technique, graph walking control, program dependence graphs, system dependence graph, control flow graphs, code-based, coverage-based, model-based, and others. Accordingly, It is the concept of finding some of test case ($\forall t'$) instead of using all of test cases ($\exists t'$). The principle of dividing types of test case selection characteristics of program, e.g., programming language and test case template, which generated by properties of code and specification under the concept of defining test case specification and execution. After this, three traditional algorithms will be described as comparative studies in this paper [9], [11], [12].

2.3 *Select-all (α_1)*

α_1 is the prior method that is developed for selecting $\forall t'$, which gives good result when dealing with small T' as shown in Figure 1, whereas the left axis is number of T' and the right axis represent % accuracy. As we can see on the left hand side of Figure 1, if T' is smaller, the percent accuracy is higher. The value of percent accuracy is dropping when the size of T' is growing. This makes the minimum T' that gives the maximum % accuracy. Therefore, α_1 may not guarantee the selection will be good at high level. This motivates researchers to study and develop different technique for handling this issue [13]–[15]. Algorithm of α_1 is shown below;

If $T' = \forall t'$ then
Select $\forall t'$
End,

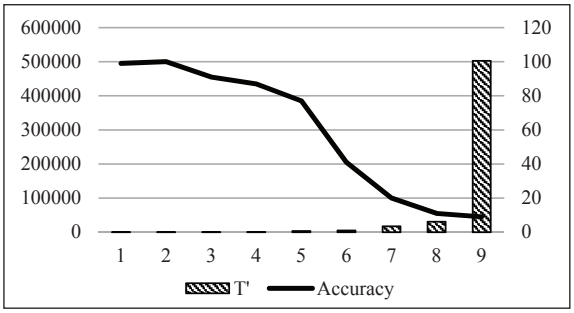

Figure 1: Test suite increases, Accuracy decreases.

However, it cannot be used well enough as he reason explained above. This is why random technique becomes an important methods for the next development.

2.4 *Random (α_2)*

α_2 is technique for choosing t' from T' of P' randomly to solve the problem that α_1 cannot do. It is a good technique for dealing with T', which is large. It doesn't suitable for a small T' when compared with α_1 [16]–[18]. Algorithm of α_2 is explained as;

Input: T'
Output: $\exists t'$
Process:
If $T' = \{t'_1, t'_2, t'_3,...t'_n\}$
then select $\exists t'$ from T' randomly
End,

Whereas $\exists t'$ is the numbers of selected test case and T' is a test suite. However, the main problem of α_2 is not about how the algorithm works but it concerns the percent accuracy of selecting a good set of $\exists t'$. Allowing to the process, α_2 is used to represent the method of selecting $\exists t'$ which can displays altered outcomes for defining $\exists t'$ based on the number of selection.

2.5 Regression test (α_3)

α_3 is method for selecting $\exists t'$ from $\forall t'$ existed in T' of the modified program by using algorithm below [19]–[21];

Algorithm
Input: T'
Output: $\exists t'$
Process:
If $T = \{t_1, t_2, t_3,...t_n\}$ then
$T' = \{t'_1, t'_2, t'_3,...t'_n\}$

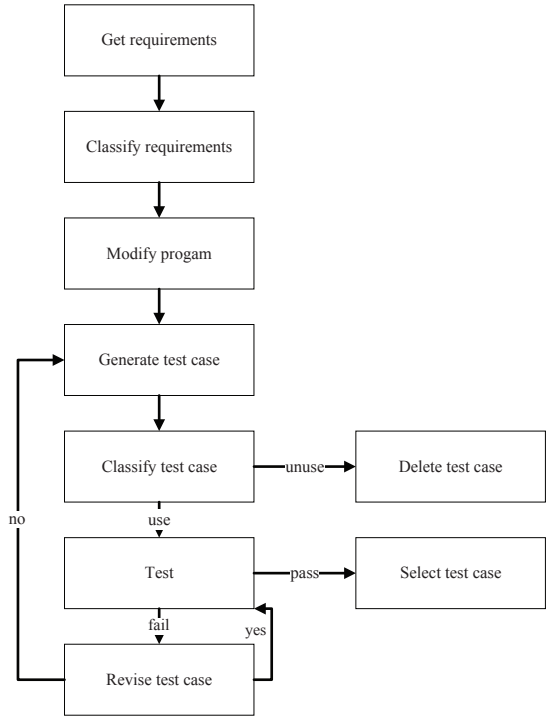

Figure 2: Conceptual model.

ElseIf necessary, generate T'' to get a set of t'' then Select $\exists t''$
ElseIf necessary, generate T''' to get a set of t''' then Select $\exists t'''$
ElseIf necessary, generate $T^{(n)}$ to get a set of $t^{(n)}$ then Select $\exists t'^{(n)}$
EndIf,
End,

2.6 *The proposed model (α_4)*

α_4 is the alternative test case selection based on classifying different requirements under the concept of software engineering that focusses on maintenance.

2.6.1 Concept of the proposed model

The total picture of the proposed model follows nine main processes as shown in Figure 2. The process starts with requirement gathering from stakeholders who get involve the entire project. After this, classifying requirements is necessary before coding or modifying program. After getting a new version software, T'

will be generated to get a set of t'. Accordingly, types of t' will be classified before testing them to know that the results of the tests are pass or fail regarding test case specification, which show detail of t'. In case, t' that pass the test will be proceeded for the selection. If fail, then it is necessary to they will be revised before the selection process has done otherwise some of t' will be back to be generated and redo the processes again. If it cannot be revise, it becomes unused t' and will be removed from the whole processes.

2.6.2 Algorithms

Process 1: Get requirement
Requirements (θ) are gathered depending on different requirements for the next software modification.

Process 2: Classify requirement
Requirements are mainly divided to five types, which are business (θ_1), user (θ_2), system (θ_3), functional (θ_4), and non-functional (θ_5) requirements by using Table 2, including reused t' (θ_7), which is always selected. It is designed for typify each requirement by checking them regarding retrieval resources.

Table 2: Requirement metric

Item	*B*	*U*	*S*	*F*	*F'*
θ_1	✓	-	-	-	-
θ_2	-	✓	-	-	-
θ_3	-	-	✓	-	-
θ_4	-	-	-	✓	-
θ_5	-	-	-	-	✓

*Types of Requirement: Business (*B*), User (*U*), System (*S*), Functional (*F*), and Non-functional (*F'*)

Process 3: Modify program
After the process 2 has done, modifying the previous software starts in order to delete, add or change the code relying on all of clear θ.

Process 4: Generate test case
This process creates T' from P' by using algorithm below;
Input: P, P' and T
Output: $T' = \{t'_1, t'_2, t'_3, ...t'_n\}$
If $P \Rightarrow T$ then
$P' \Rightarrow T'$
End,

Whereas $T = \{t_1, t_2, t_3, ...t_n\}$ and $T' = \{t'_1, t'_2, t'_3, ...t'_n\}$

Process 5: Classify test case
Classifying gives seven different test cases, which are listed as follows;
$\theta_1 = \{\theta_{1,1}, \theta_{1,2}, \theta_{1,3}, ..., \theta_{1,n}\}$
$\theta_2 = \{\theta_{2,1}, \theta_{2,2}, \theta_{2,3}, ..., \theta_{2,n}\}$
$\theta_3 = \{\theta_{3,1}, \theta_{3,2}, \theta_{3,3}, ..., \theta_{3,n}\}$
$\theta_4 = \{\theta_{4,1}, \theta_{4,2}, \theta_{4,3}, ..., \theta_{4,n}\}$
$\theta_5 = \{\theta_{5,1}, \theta_{5,2}, \theta_{5,3}, ..., \theta_{5,n}\}$
θ_6, $\{x, y, z\}$ see details in process 6.
$\theta_7 = \{\theta_{7,1}, \theta_{7,2}, \theta_{7,3}, ..., \theta_{7,n}\}$ or reused test cases
Input: $T' = \{t'_1, t'_2, t'_3, ...t'_n\}$
Output: θ_1, θ_2, θ_3, θ_4, θ_5, θ_6, and θ_7
Process:
If $t' = \theta_7$ then get a set of θ_7
ElseIf $t' = B$ then get a set of θ_1
ElseIf $t' = U$ then get a set of θ_2
ElseIf $t' = S$ then get a set of θ_3
ElseIf $t' = F$ then get a set of θ_4
ElseIf $t' = F'$ then get a set of θ_5
Else Get θ_6
End,
Process 6: Delete unused test case
Input: θ_6
Output: ϕ
Process:
If $t' = \theta_6$ then
Remove θ_6
End,

θ_6 can be classified to three types, which are obsolete $t'(x)$, redundant $t'(y)$, and unrevised $t'(z)$, whereas $x = t' \cap T' = \phi$, $y = t' \cap T' \neq \phi$, and $z = t' \cup T' = T'$

$$\theta_6 = \begin{cases} x//x = \{x_1, x_2, x_3, ..., x_n\} \\ y//y = \{y_1, y_2, y_3, ..., y_n\} \\ z//z = \{z_1, z_2, z_3, ..., z_n\} \end{cases}$$

Therefore, deleted t' or $\theta_6 = \sum(x + y + z)$

Process 7: Test
After doing process 5, what we will get is; $\theta = \sum(\theta_1 + \theta_2 + \theta_3 + \theta_4 + \theta_5)$. However, they will be tested to find test cases that give the result "pass ($\theta = 1$)" or "fail ($\theta = 0$)". Accordingly, test cases that pass the tests will be selected during some that fail the test move to the next process for revising or fixing problems.

Process 8: Revise test case
If some θ that can be revised will be brought back to process 7 and tested again for the selection process. If they cannot be revised because of weak design, then they will go back to process 4 and move throughout the process 5 to 7 again until they are deleted or chosen regarding the reason explained in process 7.
Algorithm of revising θ is explained as;
If $\theta_1 = 0$ then revise θ_1
ElseIf $\theta_2 = 0$ then revise θ_2
ElseIf $\theta_3 = 0$ then revise θ_3
ElseIf $\theta_4 = 0$ revise θ_4
ElseIf $\theta_5 = 0$ then revise θ_5
End,
Whereas, the condition of revising test case is shown below;

$$revise = \begin{cases} yes // \theta_i = 0 \\ no // \theta_i' = \phi \end{cases}$$

Therefore, θ that cannot be revised or θ' equals $\sum\theta_1' + \sum\theta_2' + \sum\theta_3' + \sum\theta_4' + \sum\theta_5'$

Process 9: Select test case
The algorithm of selection is described as follows;
Input: θ and θ'
Output: $\theta = \sum(\theta - \theta')$
Process:
If $\theta_1 = 0$ and $\theta_1' = \phi$ then group 1= $\theta_1 - \theta_1'$
ElseIf $\theta_2 = 0$ and $\theta_2' = \phi$ then group 2= $\theta_2 - \theta_2'$
ElseIf $\theta_3 = 0$ and $\theta_3' = \phi$ then group 3= $\theta_3 - \theta_3'$
ElseIf $\theta_4 = 0$ and $\theta_4' = \phi$ then group 4= $\theta_4 - \theta_4'$
ElseIf $\theta_5 = 0$ and $\theta_5' = \phi$ then group 5= $\theta_5 - \theta_5'$
End,
Therefore, the total number of selected test case (t_s') equals $\theta_7 + \sum(\theta_1 - \theta_1') + \sum(\theta_2 - \theta_2') + \sum(\theta_3 - \theta_3') + \sum(\theta_4 - \theta_4') + \sum(\theta_5 - \theta_5')$

2.7 *Evaluation*

This paper takes three kinds of formulation as the criteria of measuring the performances of the traditional methods including the proposed model explained as follows;

2.7.1 Percent Reduction (%R)

In order to evaluate the performance of alternative studies can be carried on by using Equation (1);

$$\%R = 100\% \times \left(\frac{T' - t_s'}{T'} \right) \quad (1)$$

2.7.2 Percent fault Detection

One of the most important evaluation is to avoid bugs found in the selected test cases, which need to use Equation (2) calculated as % bugs removal (%E) as;

$$\%F = \left(\frac{t_s' - f}{t_s'} \right) \times 100\% \quad (2)$$

Whereas, $t_b^{'}$ is the selected test case that has bugs or programming errors.

2.7.3 Ability of fixing bugs

This is another important evaluation to measure the performance of the comparative studies in terms of the percent ability of fixing bugs (%FB) by using Equation (3) as;

$$\%FB = 100\% \times \left(1 - \left(\frac{t_b^{'} - t_{b-}^{'}}{t_b^{'}} \right) \right) \quad (3)$$

Whereas, $t_{b-}^{'}$ is a set of the selected test case that has bugs, which can be solved by programmers.

The different between 2.7.2 and 2.7.3 is avoiding and fixing bugs, which depend upon the characteristics of the test suite and the ability of programmers.

2.8 *Scopes and limitations*

The scopes of preparing this research is using the large size of the test suite, which should greater than 5,000 but not higher than 130,000 cases. The subject programs used in the experiment are already provided and written by C-Language. Besides, all comparative studies follow the concept of test case selection. The reason is control the properties and constraints, which must be the same.

3 Results and Discussions

It is important that types of test cases need to be classified regarding change request to perform clear

requirements before testing and retaining software. Regarding seven subject programs used in experiment's part, Table 3 shows three main things, which are the amount of θ_1 to θ_5 for each program (β_1 to β_7), including T' and θ_7. Accordingly, number of the specific requirements are in range 53 to 146 requests, while the reused test case are found and countable in 57 to 271 cases. However, the report is concluded only on these seven programs. In some situation, if changing the related programs used in different experiments are added in the future, we will get different results depending on characteristics of software, e.g., software requirements, including system, hardware, and other requests. After this, the set of unrevised test case ($\theta_1' - \theta_5'$) are determined as shown in Table 4. We can find them by revising cases from θ_1 to θ_5 that cannot pass the test. Relevant to this, percent of fixing these cases are found from 8.51%–35.19% as given from Table 4. This means some of θ cannot be fixed, then they will be removed from the system and leave relevant θ for the proper selection throughout the algorithms. This makes the advantage over the old methods, while they focus on testing and picking the representatives. However, the proposed model may concern how to find the effect of different cases, which may hidden in a whole test suite. This is because it may produce some faults that drop the performance of using software. After that, Table 5 computes the difference between θ and θ' of each β to examine the amount of the selected test case. For example; Group 1, $\beta_1 : \theta_1 - \theta' = 54 - 17 = 37$ cases. The same procedure is applied for the rest and recorded. However, the total number of the chosen t' is the summation of $\Delta\theta + \theta_7$, which is shown in Table 6 at the last column (α_4), while the picked t' of α_1, α_2, and α_3 are also reported. From reading this table, the amounts of selected t' by the proposed method are smallest, while α_1 is biggest because it chooses all of t', which affect the whole system when the size gets larger, in particular, the value of β_6 cannot run software quickly but for β_1 that the size may not cause any problem. Including, Table 6 also prepare the results of finding percent reduction (%R) by all studies, which can be computed by using Equation (1);

Example of finding %R;

$$\beta_1 \text{ for } \alpha_4 : \%R = \frac{(5,680 - 499)}{5,680} \times 100\% = 92.10\%$$

Table 3: Amounts of T' and θ

β	T'	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6
β_1	5,680	54	96	56	110	128	99
β_2	10,068	94	53	109	78	77	57
β_3	10,459	63	50	146	83	127	82
β_4	35,545	87	77	90	75	103	113
β_5	59,846	72	106	112	55	81	85
β_6	122,169	121	127	75	114	141	271
β_7	14,427	54	102	139	72	132	99

Table 4: Amounts of θ'

β	θ_1'	θ_2'	θ_3'	θ_4'	θ_5'
β_1	17	22	20	21	14
β_2	13	13	14	11	21
β_3	21	12	16	18	19
β_4	17	20	12	21	17
β_5	21	18	18	16	13
β_6	17	12	13	13	12
β_7	19	12	12	15	14

Table 5: Amounts of θ'

β	Group 1	Group 2	Group 3	Group 4	Group 5	θ_7
β_1	37	74	36	89	114	99
β_2	81	40	95	67	56	57
β_3	42	38	130	65	108	82
β_4	70	57	78	54	86	113
β_5	51	88	94	39	68	85
β_6	104	115	62	101	129	271
β_7	35	90	127	57	118	721

Table 6: Selected test case and percent reduction by four algorithms

β	Item	α_1	α_2	α_3	α_4
β_1	$t'_{1,s}$	5,680	843	620	449
	%R	0.00	85.16	89.08	92.10
β_2	$t'_{2,s}$	10,068	1,550	710	396
	%R	0.00	84.60	92.95	96.07
β_3	$t'_{3,s}$	10,459	1,156	742	465
	%R	0.00	88.95	92.91	95.55
β_4	$t'_{4,s}$	35,545	1,079	786	458
	%R	0.00	96.96	97.79	98.71
β_5	$t'_{5,s}$	59,846	1,468	743	425
	%R	0.00	97.55	98.76	99.29
β_6	$t'_{6,s}$	122,169	1,066	1.025	782
	%R	0.00	99.13	99.16	99.36
β_7	$t'_{7,s}$	14,427	1,815	1,459	1,148
	%R	0.00	87.42	89.89	92.04

The interpretation of this activity says that there is impossible for α_1 to decreases the size, while α_4 can prepare the minimum. However, the problem is to use α_2 because it cannot guarantee reducing the size will give competency to the system as in range 85–99%, which is too big. On the other hand, α_3 gives 89–99%. This means that some program can be carried on by using α_2, which is no need to spend harder algorithm like α_3. Moreover, Table 7 offers the amount of fault (f) and percent avoiding fault (%F), which will be taken by using Equation (2).
Example of defining %F;

$$\beta_1 \text{ for } \alpha_4 : \%F = \left(\frac{499-8}{499}\right) \times 100\% = 99.86\%$$

Table 7: Selected test case and percent reduction by four algorithms

β	Item	α_1	α_2	α_3	α_4
β_1	f_1	24	16	15	8
	%F	99.58	98.10	99.74	99.86
β_2	f_2	24	20	15	7
	%F	99.76	98.71	99.85	99.93
β_3	f_3	30	15	13	4
	%F	99.71	98.70	99.88	99.96
β_4	f_4	23	15	12	3
	%F	99.94	98.61	99.97	99.99
β_5	f_5	25	15	15	1
	%F	99.96	98.98	99.97	100.00
β_6	f_6	28	19	13	7
	%F	99.98	98.22	99.99	99.99
β_7	f_7	29	16	13	6
	%F	99.80	99.12	99.91	99.96

The rest computations will be hold by the same procedure and presented in Table 7 as well. From the observation, this issue doesn't significant for all methods because all results of ability to avoid fault are in high range (almost 100%). The reason is that before selecting test cases, avoiding errors or faults is necessary when modifying software. Therefore, this point may not give much different output. As we can see all results reported, the ability of using the proposed model is better than previous studies. This is alternative way when dealing with change request and different types of requirement are concerned when modifying software becomes the focus of the whole software development.

4 Conclusions

Regression test selection is one of the most essential method for handling the process of software maintenance when new requirements are involved. The development team need to find the appropriate test case generator for designing good test suite that avoids new faults as well. The complexity of using this technique in particular, designing test cases becomes a new problem, which is the cost-effective. Therefore, to fix this problem and to keep the ability of the selection model turns a necessary job. The proposed model focuses on selecting the reused test cases from the previous test suite of each program first to guarantee that the competency of the program will be preserved, while the traditional methods skip this issue. According to the experiments, retest-all gives lowest performance of reducing the test suite' size while random technique gives better outcomes but when look at the whole picture of the selection is still not good enough, which the regression test selection can offer good yields. However, due to the objective of present this paper is to develop better algorithm for controlling total of selected test cases are still large regarding using the old methods. Moreover, numbers of fault after selection is bigger the proposed model because they cannot avoid those errors, which can be found in the set of chosen test cases. From doing the experiment, the ability of reducing test suite size by using the proposed model is better than the traditional methods as around 0.32%–99.36%. Besides, it also gives percent of avoiding fault as about 99.86%–100.00%. For the future work, the algorithms of deletion should be considered before the selection process to the reason of removing all irrelevant test cases. This can reduce execution time of testing the selected test cases, including preparing well cost-effective, which the regression test selection techniques cannot avoid.

Acknowledgements

The research project was funded by Assumption University.

References

[1] P. Bhatt, G. Shroff, and A. K. Misa, "Dynamics of software maintenance," *ACM SIGSOFT Software Engineering Notes*, vol. 29, no. 5, pp. 1–5, 2004.

[2] D. I. K. Sjoberg, B. Anda, and A. Mockus, "Questioning software maintenance metrics: A comparative case study," in *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, 2012, pp. 107–110.

[3] O. Denninger, "Recommending relevant code artifacts for change requests using multiple predictors," in *Proceedings of the 3rd International Workshop on Recommendation Systems for Software Engineering*, 2012, pp. 78–79.

[4] A. Lawanna, "Filtering test case selection for increasing the performance of regression testing," *KMUTNB Int J Appl Sci Technol*, vol. 9, no. 1, pp. 19–25, 2016.

[5] A. Magalhaes, F. Barros, A. Mota, and E. Maia, "Automatic selection of test cases for regression testing," in *Proceedings of the 1st Brazilian Symposium on Systematic and Automated Software Testing*, 2016, pp. 19–20.

[6] L. Mariani, O. Riganelli, M. Santoro, and M. Ali, "G-RankTest: Regression testing of controller applications," in *Proceedings of the 7th International Workshop on Automation of Software Test*, 2012, pp. 131–137.

[7] L. Gong, D. Lo, L. Jiang, H. Zhang, "Diversity maximization speedup for fault localization," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pp. 30–39, 2012.

[8] A. Lawanna, "Test case based selection for the process of software maintenance," *Silapakorn University of Science and Technology Journal*, vol. 7, no. 2, pp. 36–45, 2013.

[9] A. Lawanna, "Technique for test case selection in software maintenance," *Walailak Journal of Science and Technology*, vol. 11, no. 2, pp. 69–77, 2014.

[10] H. Do, S. G. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empirical Software Engineering*, vol. 10, no. 4, pp. 405–435, 2005.

[11] H. Do, G. Rothermel, "An empirical study of regression testing techniaues incorporating context and lifetime factors improved cost benefit models," in *Proceedings of the 14th ACM SIG-SOFT International Symposium on Foundations of Software Engineering*, 2006, pp. 141–151.

[12] G. Rothermel and M. J. Harrold, "Analyzing regression test selection techniques," *IEEE Transactions on Software. Engineering*, vol. 22, no. 8, pp. 529–551, 1996.

[13] G. Rothermel and M. J. Harrold, "A safe, efficient regression test selection technique," *ACM Transactions on Software Engineering and Methodology*, vol. 6, no. 2, pp. 173–210, 1997.

[14] M. Grindal, B. Lindstrom, J. Offutt, and S.F. Andler, "An evaluation of combination strategies for test case selection," *Empirical Software Engineering*, vol. 11, no. 4, pp. 1–31, 1977.

[15] P. Hsia, X. Li, D. Kung, C-T. Hsu, L. Li, Y. Toyoshima, and C. Chen, "A technique for the selective revalidation of OO software," *Software Maintenance: Research and Practice*, vol. 9, pp. 217–233, 1997.

[16] D. L. Bird and C. U. Munoz, "Automatic generation of random self-checking test cases," *IBM System Journal*, vol. 22, no. 3, pp. 299–245, 1983.

[17] T. Y. Chen, F. C. Kue, R. G. Merkel, and T. H. Tse, "Adaptive random testing: The ART of test case diversity," *Journal of Systems and Software*, vol. 83, no. 1, pp. 60–66, 2010.

[18] Z. Zhou, "Using coverage information guide test case selection in adaptive random testing," in *Proceedings of Computer Software and Application Conference Workshops*, pp. 1–8, 2010.

[19] E. Engström, P. Runeson, and M. Skoglund, "A systematic review on regression test selection techniques," *Information and Software Technology*, vol. 52, no. 1, pp. 14–30, 2010.

[20] A. Hooda and S. Panwar, "A roadmap for effective regression testing," *International Journal of Scientific and Engineering Research*, vol. 7, no. 5, pp. 214–220, 2016.

[21] A. Ansari, A. Khan, A. Khan, and K. Mukadam, "Optimized regression test using test case prioritization," in *Proceedings of the 7th International Conference on Communication, Computing and Virtualization*, pp. 152–160, 2016.