Filtering Test Case Selection for Increasing the Performance of Regression Testing

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

* Corresponding author. E-mail: adtha@scitech.au.edu DOI: 10.14416/j.ijast.2015.11.002
Received: 16 October 2015; Accepted: 26 November 2015; Published online: 4 January 2016

Abstract
Under the concept of using software testing, the problems are preparing the smallest size of the selected test cases, fixing bugs, predicting the estimated testing time and numbers of the tester. The traditional methods are developed to solve these issues. Unfortunately, they cannot be applied for all reasons. Therefore, the filtering test case selection model is proposed to solve these problems and to increase the efficiency of the regression testing. It prepares the methods of filtering, classifying, and selecting the appropriate test cases. Accordingly, it gives the smaller size of the selected test cases than the traditional selections as 0.23–3.96%. When consider the fault rate measured by the developed technique is also less than those methods equivalent to 13–69%. Furthermore, the testing time and the amount of testers are also less than the comparative studies as about 2–71%, and 13–69% respectively.

Keywords: Regression testing, Test case selection, Test suite

1 Introduction

Regression testing is the part of software testing the software development life cycle, which is one of the methodologies of software engineering [1]. The main function of regression testing technique is to modify the old program to the new version and to moderate the size of the test cases before testing the new codes [2]. The reason of using this technique is to guarantee that newly added or modified code works correctly [3]. However, the important problem is that which technique is appropriate in the differentiated software development environments [4]. The scopes of using regression testing listed as follows; it is suitable for only the modified program, testing time is accounted, testers may not be from the previous team, and many testing cycles are required when the modifications are done [5]. In general, it needs six processes, which are modifying the latest program, preparing test data, testing the sequence of code, execution, verifying the test cases, and fixing bugs [6]. In fact, the requirement specifications are directed to write a code, including the new software version [7]. Moreover, the amount of the specifications of the revised software is greater than the old [8]. This is because the needs of user requirements increase [9]. The main problems are what should be the selected test cases of the modified program, how to select, and the size of the chosen set is still too big [10]. These could affect the efficiency of whole processes of applying the concept of providing the suitable cases to get correctly modified codes [11].

There are many regression testing methods developed for solving these problems, e.g., retest-all [12], random [13], obsolete [14], re-testable [15], reusable [16], redundant [17], execution traces [18], fault-revealing [19], modification-revealing [20], inclusive [15], precise [15], safe regression [8], and others. This paper studies the random technique because it is practical for general

Please cite this article as: 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, Jan.–Mar. 2016.

purpose, especially for the small test suite. Besides, re-testable is reviewed, which the unchanged test cases are chosen. This gives a high stability of measuring the success of the system. Additionally, reusable technique is used for finding the test cases that benefit to the developer. Including the fault revealing method, this can show the programming errors in the testing process. As well as an execution trace that performs the high efficiency for fixing many bugs in a code.

The same statements of problem are realized, e.g., the large size of the test suites, bugs increase, and testing time consuming. The goal of offering the proposed model, named the filtering test case selection is to give the minimum amount of these problems compared with those old-style processes. In addition, the estimated testing time will be shown with the procedure and calculation for offering the alternative decision making, when this becomes the issue of adapting the codes.

2 Concepts of Regression Test Selection

2.1 *Terms used*

Table 1 is intended for understanding each technical term used in this research paper.

Table 1: Terms used in this research paper

Term	Description
P	The previous program [1]
P'	The modified program [1]
T	Test suite of the old software [1]
T'	Test suite of the adopted software [1]
t	No. of test case in the original software [1]
t'	The amount of the selected test cases in a modified code [1]
t_r	Redundant test case [2], [17]
T'_r	No. test case after removing the redundancy
Q	Frequency (%)
n	Nodes in the control flow graph [8]
G	The control flow graph [8]
b	Bugs or faults [6]
R	Reduction Rate [1]–[5]
F	Fault Rate [7]
S'	Estimated testing time (hours)
S	Exactly testing time required by users (days)
E'	Expected no. of testers
E	A real no. of testers
λ_1	Random approach [13]
λ_2	Re-testable [15]
λ_3	Reusable [16]
λ_4	Fault-revealing [19]
λ_5	Execution trace [18], [21]
λ_6	Filtering test case selection

2.2 *Regression testing*

It is a technique used in software testing, which reduce the size of T' generated for P' by choosing t' existed in this T'. If P' is adopted from using the P, then $t' \in T'$. There are three main algorithms used in this concept explained as follows;
Algorithm of the modification;
If input P then
constitute P'
End,
Algorithm of generating a test suite for the new program;
If there is P' then
generate T'
End,
Algorithm of selecting t'
If there is T' then
select only t' that is valid or without false positive
End,

2.3 *Re-testable*

It executes the unchanged test cases of both previous (P) and P', which $t^u \in T$ and T'. It avoids the errors and the changes, which could be performed in P'. Unfortunately, that to re-examining t' is a difficult job to do.
Algorithm of selecting t'
If there is T' then
select only t^u that is « unchanged »
End,

2.4 *Reusable*

It needs the reusable test cases (t^r) of P', which $t^r \in T'$ is considered. It focuses the reusable test cases the new software version, which numbers will equivalent to the previous one. Using this algorithm may not handle the cost-effective and errors through the whole program. Comparing this technique to the previous methods, it gives the better performances.
Algorithm of picking t'
If there is T' then
select only t^u that is « reusable »
End,

2.5 *Fault-revealing*

It tests the sequence of the original code and builds the control flow graph that consists of important node, which the test cases can be performed. One problem can be occurred, which may be about revealing the multiple types of the faults and taking much analysis.
Algorithm of selecting t^f
If modify P then
execute P′
Else If find a fault in T′ then
select t^f, which $t^f \in T'$
End,

2.6 *Execution trace*

After modifying the original software, P' will be the output that needs to be executed for finding bugs or programming errors, which found in T'. According to this, the maximum number of faults will be examined. Therefore, the test cases with bugs are chosen. However, this technique can make the complex tasks in finding and fixing the faults.
Algorithm of picking t'
If execute P then
record G when it is denoted as the control flow graph of P
Else If compare P′ with P then
select node (n), which $n \in G$
Else If choose $t^n \in n$
End,

2.7 *The proposed model for improving the ability of the regression test selection*

2.7.1 Algorithm

There are three algorithms created as followings;

Algorithm 1: Filtering a test suite

A test case (t'_r) is redundant for T' if the test suite has exercised method execution equivalent to all method execution exercised by the test case. Suppose that there are six test cases in a test suite as $\{t'_1, t'_2, t'_3, \ldots, t'_6\}$, which has the value of testing requirements {4, 2, 2, 1, 2, 1} respectively [2]. Therefore, t'_1 is a redundant test case because it has the highest value of the testing requirement, which equals 4. The reason of creating algorithm 1 is to remove the redundant test cases, which are produced during the process of making the template. The testing is not affected by this activity because they are not significant in term of checking the programming errors. They will be covered the suitable test cases, e.g., the utility of cases. The expectation is to use the highest frequency of utilizing and choosing the cases, which will be demonstrated in the algorithm 2 and 3.
Input: T'
Output: $T' - t'_r = T'_r$ (1)
If $t'_r \in T'_r$ *then*
Remove t'_r
End,

Algorithm 2: Classifying the range of the utilization

As shown in Table 2, it segments the test cases into five groups depending on the frequency of using t'. For example, if the Q equals 81–100%, this range (t'_5) can deliver the maximum number of t'. The reasons of the segmentation are explained as follows;
(i) prepare the percent range of difference frequency of use, which are 1–20%, 21–40%, 41–60%, 61–80%, and 81–100%.
(ii) Utility value can be provided from (i).
(iii) Maximum value of utility is selected.
Input: T'_r

Output: $T'_r = \begin{cases} t'_1 \\ t'_2 \\ t'_3 \\ t'_4 \\ t'_5 \end{cases}$ (2)

If $1 \le Q \le 20$ *then*
$t'_1 \in T'_r$
Else If $21 \le Q \le 40$ *then*
$t'_2 \in T'_r$
Else If $41 \le Q \le 60$ *then*
$t'_3 \in T'_r$
Else If $61 \le Q \le 80$ *then*
$t'_4 \in T'_r$
Else If $81 \le Q \le 100$ *then*
$t'_5 \in T'_r$
End,

The difficulty of this algorithm is to build the rank or the range of the exploitation, which relies on several factors, e.g., programmers, skills, knowledge, template or details of the test cases. For the different modifying the software, the conditions and the software environments may change than can yield other outcomes.

Table 2: Utility of the test cases

Q(%)	Utility	Output
1–20	Lowest	t'_1
21–40	Low	t'_2
41–60	Normal	t'_3
61–80	High	t'_4
81–100	Highest	t'_5

Algorithm 3: Selecting the appropriate test cases

The most important thing is to pick the t', which relies on many factors. However, this algorithm considers the results from the previous algorithm. Besides, bugs are produced when $t' = 0$. Therefore, the collection of the test case that has $t' = 0$ will be selected for this purpose.
Input : t'_5
Output : $t' = 0$
When $t' = 0$, this means the test case that has bugs. Accordingly, the t'_5 will be selected. On the other hand, $t' = 1$ when there is no bug, which is not required because it works properly. This is because those test cases are unchanged from the original turning to the adopted codes.
If $t' = 0$ *then*
    Select t'
End,

2.7.2 Experiment

Seven steps of doing the experiment are listed as follows;
Step 1: Download the subject program from http://sir.unl.edu/php/previewfiles.php
Step 2: Get the specifications requirement from users
Step 3: Prepare P'
Step 4: Generate T'
Step 5: Remove the redundant test cases
Step 6 : Classify the test cases regarding the range of frequency (%)
Step 7 : Select the test cases in the highest range.

2.8 *Evaluation*

In this part, the evaluation methods are used to compare the capabilities of the comparative studies, which can be checked by the ability of decreasing the size of the chosen test case regarding the reduction rate, fault rate, estimated testing time and testers explained as the followings;

2.8.1 Reduction rate

One factor that is important in challenging the motivation of creating the regression test selection refers to using reduction rate to measure the rate of decreasing the test suite size, while the competency is not damaged, as indicated in the Equation (3) below,

$$R = \frac{T - t'}{T} \tag{3}$$

2.8.2 Fault rate

Bugs or faults can be produced after modifying the codes, which are found in t' of each study. Therefore, Equation (4) is acquired for checking the fault rate. However, the amounts of b can be shown in the test cases that result the failures of the testing program.

$$F = \frac{t' - b}{t'} \tag{4}$$

2.8.3 Estimated testing time

This criteria is added to find the value of the S', when S is fixed by the needs of users. According to this, the value of t' and T' are required and used in Equation (5).

$$S' = \frac{t'\,S}{T'} \tag{5}$$

2.8.4 Expected no. of testers

The last calculation needed in this section is shown in Equation (6). It can help the development team to predict the amounts of tester depending on t', T', and the real E.

$$E' = \frac{t'\,E}{T'} \tag{6}$$

3 Results and Discussion

Table 3 contributes name, abbreviation, test suite, real testing time and numbers of tester for the programs. In fact, T' is given and used for the regression test selection techniques. But the value of S and E are prepared by taking data from the real situations. For the part of evaluation, in order to measure the S' and E' can be done relying on Equation (5) and (6). The results given in Table 4 are the outputs of finding the different t' of using λ_6 . For example, in a test suite of TC, there are 257 redundant test cases. Besides, the test cases in t'_1, t'_2, t'_3, t'_4, and t'_5 are 498, 354, 289, 193, and 16 respectively. Therefore, 16 test cases are chosen because they existed in the highest range of use regarding the algorithm 3 of the proposed model. After this, Table 5 describes the final results by selecting test cases for seven programs by taking those six studies. It interprets that using λ_6 can give the smallest amounts of t'. With this, it guarantees that the less size the more abilities of the regression test selection. Besides, Table 6 calculates the R by Equation (3) for all techniques. It proves that λ_6 results the highest rate compared with others. Therefore, the highest of the reduction rate are formed by λ_6. This implies that λ_6 is reasonable for reducing and giving the smallest size. Table 7 shows numbers of bugs reported for each technique after the modification. Table 8 represents the description of computing F, which λ_1 is highest and λ_6 is lowest for all modified software. While, λ_2, λ_3, λ_4, and λ_5 offer the smaller values from the left to the right of each row. In Table 9, the estimated test time is calculated to recommend that λ_6 prefers the smallest values, while λ_1 takes longest. Besides, the results of predicting the amounts of testers are offered in Table 10. Let's say that λ_2, λ_3, λ_4, and λ_5 propose the higher number of testers than λ_6, while λ_1 does many errors.

Table 3: The programs used in the experiment

Name	Abbr	T'	S	E
Tcas	TC	1,608	27	13
Totinfo	TO	1,052	17	15
Schedule	SC	2,650	17	13
Schedule2	SC2	2,710	6	18
Print-tokens	PT	4,130	17	10
Print-tokens2	PT2	4,115	13	8
Replace	RP	5,542	24	14

Table 4: Size of the test suites provided by λ_6

	t'_r	t'_1	t'_2	t'_3	t'_4	t'_5
TC	257	498	354	289	193	16
TO	137	295	263	210	137	11
SC	292	875	663	530	265	27
SC2	352	759	705	515	352	27
PT	743	1033	867	826	578	83
PT2	864	905	864	782	617	82
RP	1108	1164	1330	942	831	166

Table 5: Numbers of the selected test case by the comparative studies

	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6
TC	69	24	21	20	18	16
TO	64	16	15	14	13	11
SC	57	41	36	34	30	27
SC2	124	39	38	34	32	27
PT	417	120	109	106	95	83
PT2	202	122	107	102	97	82
RP	370	242	232	206	184	166

Table 6: The percent reduction rate of λ_6 is greater than the traditional studies

	λ_1	λ_2	λ_3	λ_4	λ_5
TC	3.44	0.51	0.32	0.25	0.13
TO	5.36	0.48	0.39	0.29	0.19
SC	1.16	0.54	0.34	0.27	0.11
SC2	3.75	0.45	0.41	0.26	0.19
PT	9.00	0.92	0.65	0.57	0.30
PT2	3.07	1.00	0.62	0.50	0.37
RP	3.94	1.43	1.24	0.75	0.34

Table 7: Bugs

	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6
TC	69	24	21	20	18	16
TO	64	16	15	14	13	11
SC	57	41	36	34	30	27
SC2	124	39	38	34	32	27
PT	417	120	109	106	95	83
PT2	202	122	107	102	97	82
RP	370	242	232	206	184	166

Table 8: Fault rate

	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6
TC	0.7681	0.3333	0.2381	0.2000	0.1111	0.0000
TO	0.8281	0.3125	0.2667	0.2143	0.1538	0.0000
SC	0.5263	0.3415	0.2500	0.2059	0.1000	0.0000
SC2	0.7823	0.3077	0.2895	0.2059	0.1563	0.0000
PT	0.8010	0.3083	0.2385	0.2170	0.1263	0.0000
PT2	0.5941	0.3279	0.2336	0.1961	0.1546	0.0000
RP	0.5514	0.3140	0.2845	0.1942	0.0978	0.0000

Table 9: Estimated testing time (hours)

	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6
TC	28	10	8	8	7	6
TO	25	6	6	5	5	4
SC	9	6	6	5	5	4
SC2	7	2	2	2	2	1
PT	41	12	11	10	9	8
PT2	15	9	8	8	7	6
RP	38	25	24	21	19	17

Table 10: Expected number of testers

	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6
TC	13	5	4	4	3	3
TO	22	5	5	5	4	4
SC	7	5	4	4	4	3
SC2	20	6	6	5	5	4
PT	24	7	6	6	6	5
PT2	9	6	5	5	5	4
RP	22	15	14	12	11	10

4 Conclusions

This article presents a new technique under the concept of regression test selection, named the filtering test case selection. By using this model, it gives the better cost-effectiveness, which refers to the efficiency of removing the test cases while fault rate is provided. Besides, it shows the methods of determining the expected testing time and number of programmers for handling the software modification. When, it is compared with the traditional methods mentioned in this research. Taking the proposed model, the ability of decreasing the sizes is approximately 2% (on average) lower than others. Accordingly, it selects the maximum percent of using the adopted test cases that consider the minimum number of faults. This can aim the testers to solve those problems in a new program. From Table 7, the numbers of bug by using the proposed model less than the old methods, which mean avoiding testing overhead. The challenges of preparing the filtering selection are to present the techniques for minimizing testing time and testers, which can be the recommendation for the long-run behavior of improving the competency and preservation of the software testing. In the future works, the interested test case selection should be realized such as prioritization and minimization technique.

References

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

[2] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 1–60, 2007.

[3] E. Rogstand, L. Raiand, and R. Torkar, "Test case selection for blak-box regression testing of database applications," *Information and Software Technology*, vol. 55, no. 10, pp. 1781–1795, Oct. 2013.

[4] Y. C. Huang, K. L. Peng, and C. Y. Huang, "A history-based cost-cognization test case prioritization technique in regression testing," *Journal of System and Software*, vol. 85, no. 3, pp. 626–637, Mar. 2012.

[5] J. Porter and P. Yu, "Regression discontinuity designs with unknown discontinuity points: testing and estimation," *Journal of Economics*, vol. 189, no. 1, Nov. 2015.

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

[7] H. Zhong and L. Mei, "An experimental study of four typical test suite reduction techniques," *Information and Software Technology*, vol. 50, no. 6, pp. 534–546, 2008.

[8] G. Rothermel, M. J. Harrold, J. Ronne, and C. Hong, "Empirical studies of test suite reduction," *Software Testing, Verification, and Reliability*, vol. 4, no. 2, pp. 219–249, 2002.

[9] C. T. Lin, K. W. Tang, and G. M. Kapfhammer, "Test suite reduction methods that decrease regression testing costs by identifying irreplaceable tests," *Information and Software Technology*, vol. 56, no. 10, Oct. 2014.

[10] G. Dandan, W. Tiantian, S. Xiaohong, and M. Peijun, "A test-suite reduction approach to improving fault-localization effectiveness," *Computer Languages, Systems and Structure*, vol. 39, no. 3, pp. 95–108, Oct. 2013.

[11] T. Y. Chen and M. F. Lau, "Dividing strategies for the optimization of a test suite," *Information*

Processing Letters, vol. 60, no. 3, pp. 135–141, 1996.

[12] H. K. N. Leung and L. A. Leung, "A cost model to compare regression test strategies," in *Proceeding ICSM*, 1991, pp. 201–208.

[13] W. E. Wong, J. R. Horgan, and A. P. Mathur, "Effect of test set minimization on fault detection effectiveness," *Software Practiceand Experience*, vol. 28, no. 4, pp. 347–369, Apr. 1998.

[14] H. K. N. Leung and L. White, "Insight into regression testing," in *Proceeding ICSM*, 1989, pp. 60–69.

[15] I. Granja and M. Jino, "Technique for regression testing: selecting test case sets tailored to possibly modified functionalities," in *Proceeding CSMR*, 1999.

[16] S. Bates and S. Horwitz, "Incremental program testing usig program dependence graphs," in *Proceeding ACM SIGPLAN-SIGACT*, 1993, pp. 384–396.

[17] D. Jeffrey and N. Gupta, "Test suite reduction with selective redundancy," in *Proceeding ICSM'05*, 2005, pp. 549–558.

[18] H. Agrawal, J. R. Horgan, E. W. Krauser, and S. A. London, "Incremental regression testing," in *Proceeding ICSM*, 1993, pp. 348–357.

[19] D. Leon and A. Podgursski, "A comparison of coverage-based and distribution-based techniques for filtering and optimizing test cases," in *Proceeding ISSRE*, 2003, pp. 442–456.

[20] G. Rothermel and M. J. Harrold, "A framework for evaluating regression test selection techniques," in *Proceeding ICSE*, 1994, pp. 201–210.

[21] I. Alazzam, I. Alsmadi, and M. Akour, "Test case selection on source code features extraction," *International Journal of Software Engineering and Its Applications*, vol. 8, no. 1, pp. 203–214, 2014.