Extended Regression Test Model for Test Suite Reduction

Adtha Lawanna
Department of Information Technology
Science and Technology, Assumption University
Samuthprakarn, Thailand
adtha@scitech.au.edu

***Abstract*—Problem of test suite size increases in the process of software maintenance when numbers of revision are required can be solved by reduction algorithms such random and regression test reduction. Nevertheless, the whole performance need to be improved. Therefore, this research paper proposes the new model that is used for improving the regression test reduction, which guarantees the best results compared with traditional methods by adding four algorithms; testing, ordering, deleting test cases and fixing bugs. Accordingly, it gives smaller size and higher percent fixing bugs as about 40% and 200% respectively.**

Keywords—test case; test suite; reduction; deletion

I. INTRODUCTION

Test suite reduction techniques are provided for controlling the size of the modified software and to guarantee that after adaptation, it can work properly [1-2]. This means that the modification can be used to avoid problem of programming errors and support users regarding their requirement changes [3]. As we know that one of the main problems of software maintenance is the size of a new test suite that is used to guarantee the ability of using a modified program increases. Accordingly, it drops the performance of execution and processing times including producing a software bug after the adaptations. Therefore, size reduction methods are designed to fix this issue, which includes retest-all, random and regression test techniques [4-6]. Benefit of applying retest-all is given when the size is small regarding test suite reduction [7]. If the size gets larger, it increases the mentioned issues explained above [8-11]. To solve these problems, random technique is proposed but the weakness is that the accuracy of reducing the test suite size may not give an appropriate result, when dealing with numbers of bug [12]. Hence, the regression test model is applied for better performance compared with the traditional methods. This technique provides smaller size with less programming error by preparing iterations of selecting good representative by avoiding those bugs. However, we found that if we can reduce the size of test suite without the effect on the use of software, then it can be new alternative for program developer and tester to prepare effective modified software. However, the traditional methods [13] still need the improvement when the different size of test suites are given. Therefore, the proposed model is offered by improving the performance of regression test reduction to contribute two benefits, which are giving a lower size and higher percent of fixing bugs when compared to the original approach.

II. TEST SUITE REDUCTION

Test suite reduction becomes the idea of removing irrelevant test cases under the field of software engineering, which the main objective is to increase ability of the modified program, while user requirements get involved. Accordingly, this paper focuses four techniques, which are retest-all, random, and regression test including the extended regression test model.

A. Retest-all test cases (R_1)

It is the traditional method that verifies and validates a new version of the software modification by testing all test cases in a specified test suite of each program. Besides, R_1 is a well-known technique that is useful since this study becomes the hot issue of software maintenance because the algorithm is simple to use. But, when records of test suite increase that means the selected test cases also grow as well. Thus, this difficulty cannot be fixed whereas the size is too big. Accordingly, the different selection approaches are proposed for this reason, e.g., random technique, regression test model, minimization and prioritization. R_1 tests and chooses all test cases (t_i) in a test suite (T') after modifying program (P') [1]. It works very well when amounts of t_i or the size of T' are small, while $\forall t_i$ are tested and chosen. However, the weakness is that the ability of P' drops when dealing with a large T'.

B. Random Deletion (R_2)

Using R_1 cannot guarantee the good results when the size of T' is to big. Therefore, random Deletion (R_2) is proposed to fix the problem of R_1 by randomly removing $\exists t_i$ in T' [2]. One of the objectives of applying R_1 is reduce only the size of T' without avoiding bugs (b) that could be occurred when $\exists t_i$ fail after test. Moreover, this technique can help developer to save testing time as well regarding the reason explained at the beginning. However, the performance of R_2 is better than R_1 whereas T' gets bigger but another problem of using R_2 is that bugs will affect a new program when they are not fixed. This is the reason why regression test selection is proposed.

According to this, numbers of the selected test cases are reduced with less errors.

C. Regression Test Reduction (R_3)

It is a better technique used for choosing a set of test cases that avoid programming errors, while the size is control. It can solve the critical issue in software maintenance. R_3 is concerned as a suitable technique that reduces the size of T' by picking $\exists t_i$, which has no b, while T' is being controlled. Accordingly, R_3 gives better results when compared the traditional methods (R_1 and R_2). There are five steps to run R_3 listed as follows [4-6];

(i) execute P' to find b by using R_1,

(ii) build generate $\forall t_i$ or T' for the executed P' based upon the functional and non-functional requirements,

(iii) design the coverage matrix to identify relevant and relevant t_i in T',

(iv) determine b in T' to confirm that the coverage matrix is a well design, and

(v) remove $\exists t_i$ that doesn't meet the conditions, however, revising $\exists t_i$ may be required when $t_i = \phi$.

The objective of this paper is to improve ability of R_3 by adding an effective methods at the last step, which will be explained in section III.

III. The Proposed Model

The proposed model (R_4) is created to improve ability of the last process of R_3 regarding (i) T' of P' are given, (ii) $\forall t_i$ are tested to avoid b, (iii) $\forall t_i$ are ordered, (iv) irrelevant test cases are deleted, and (vi) checking bugs. However, this paper presents three algorithms for extending the regression test, which are described as follows,

Algorithm 1: Testing $\forall t_i$

Input: T'

Output: t_p, t_f, and t_v

(1) Let $t_i \in T'$, $t_p \in t$ and $t_f \in t$, whereas t_i comprises passed (t_p) and failed (t_f) test case respectively.

(2) If $t_f \in t$ then t_f will be removed, or else if $t_p \in t$ then keep t_p for the next classification.

(3) Moreover, if $t_v \in t$ then we have to rewrite t_v, whereas t_v is a revised test case. These results that T' combines t_p, t_f, and t_v.

Algorithm 2: Ordering t_u and t_d

Input: t_p, t_f, and t_v

Output: t_u and t_d

Where, $\{t_o, t_e, t_c\} \in t_u$ and $\{t_g, t_n\} \in t_d$

(4) If $t_u \in t_p$ then t_p is ordered to give a set of t_u and t_d, which are denoted as unused and used test cases.

(5) Ordering t_u presents an obsolete (t_o), redundant (t_e), and complex (t_c) test cases, while t_d concludes t_g and t_n, which are reused and new test cases with faultless, respectively.

Algorithm 3: Deleting t_u

Input: t_u and t_d

Output: t_g and t_n

(6) t_o, t_e, and t_c will be removed from T'

(7) if "$t_o \in t_u$" then delete t_o or

(8) else if "$t_e \in t_u$" then delete t_e or

(9) else if "$t_c \in t_u$" then delete t_c.

Algorithm 4: Finding bugs

(10) For $\exists b \in t_d$ and $b \neq \phi$, Delete b

(11) While $\exists b \notin t_d$ or $b = \phi$, Receive $t_d = t_d - 0$

Evaluation of the performance of test suite reduction can be considered by using percent Size Reduction (%S), Percent bugs (%B), and percent efficiency of removing bugs(%R) shown as follows;

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

$$\%B = 100\% \times \left(\frac{b}{t_d} \right) \quad (2)$$

$$\%R = 100\% \times \left(\frac{b - b'}{b} \right) \quad (3)$$

Where, b and b' is amounts of bugs and fixed bugs in the modified program.

As shown by Fig. 1, R_4 focuses the result of R_3 by adding three algorithms discussed previously to determine new T' that combines t_p, t_f, and t_v. For T' combines t_f, and t_v, they will be removed from the system in order to avoid redesigning test cases, which may take long time. The main objective is to find t_p that combines t_u and t_d. However, t_u gives t_o, t_e, and t_c, which will be removed from new T'

Finally, t_d will be classified into t_g and t_n for to be chosen as a good representative by selection strategy.

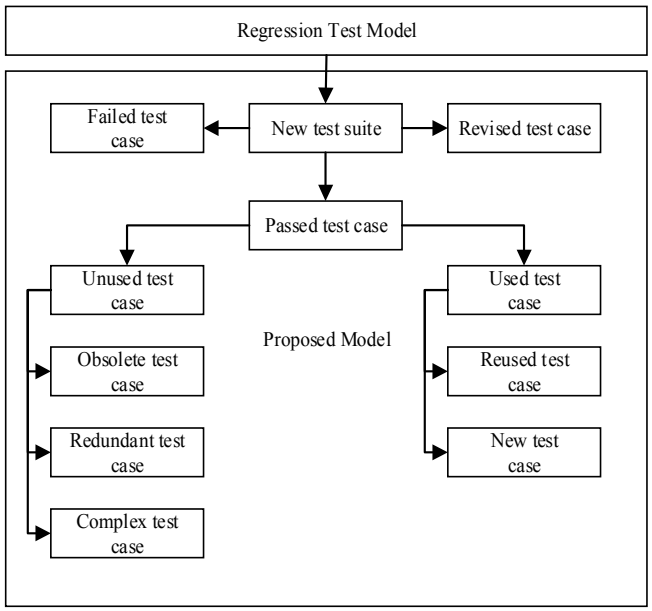

Figure 1: Structure of the proposed model

IV. Results and Discussion

There are five factors discussed in this section, which can show ability of the proposed model to control the size of a new test suite of each selected program (Tcas, Totinfo, Schedule, Schedule2, Print-tokens, Print-tokens2, and Replace). The results of R_4 will be compared with the three traditional algorithms explained as follows;

A. Size Reduction

Table I gives the amounts t_p, t_f, and t_v. On the part of doing the experiments by checking the results of testing t, which getting "passed" will be typed as *p*. Accordingly, this gives the outputs as follows; $\alpha = 1{,}267$, $\alpha 2 = 717$, $\beta = 2{,}308$, $\beta 2 = 2{,}401$, $\chi = 3{,}802$, $\chi 2 = 3{,}722$, and $\delta = 5{,}251$. For the cases that getting "failed" or *f* refer to investigating the programming errors, which cannot be integrated to the modified programs. It shows the value of $\alpha = 241$, $\alpha 2 = 276$, $\beta = 271$, $\beta 2 = 289$, $\chi = 255$, $\chi 2 = 297$, and $\delta = 270$. Table II reports the results of running R_4, which are the amount of unused and used test cases. For studying t_v, it is not detailed in this paper because revising test cases interferes the design of appropriate criteria, e.g., setting scopes and limitations of modifying codes, which takes time. Moreover, data in Table II shows more information for determining t_u and t_d, which are member of t_p as;

$$t_p = t_u + t_d \tag{4}$$

Example of computing p by using (4) for dataset named α gives that $p = 1{,}138 + 129 = 1{,}267$. The same procedure can be applied for the rest. For the details of finding a set of t_u and t_d use (5) and (6) respectively.

$$t_u = t_o + t_e + t_c \tag{5}$$

$$t_d = t_g + t_n \tag{6}$$

Therefore, the outcomes of choosing d by different four studies offers that R_4 can prepare the smaller size that others as shown in Table IV. Accordingly, the average *%S* of R_2, R_3, and R_4 are 28.43%, 74.14%, and 93.56% respectively. The lowest performance of decreasing the size of the selected test case refer to using R_2. This means that R_2 technique does not appropriate for the test suite size, where the number of test cases are in the range 1,000-5,000. Besides, Table V compares performance of reducing the size of test suite of each program between R_4 versus R_2 and R_3 respectively. According to this, R_4 gives better results than others.

B. Bugs

For *d* or selected test cases from different technique explained in this research have no bugs when they are tested individually. However, after the integration, some bugs are found reported in Table VI. Besides, Table VI and VII presents amount and percentage of bugs by those studies. It shows that the satisfied reports can be done by the proposed model, which better than the comparative studies as 2-25 times. Therefore, to apply R_4 for the test suite size between 1,000-5,000 cases can give the best answer, which the traditional methods cannot offer the small size because if the size becomes smaller, numbers of bugs may increase.

C. Efficiency

Fixing bugs as reported in Table VIII will be held to make the program works properly. It reports that some number of bugs can be solved. Unfortunately that this part has been done under the limitation of testers and supports. As shown by Table IX, R_4 can give 100.00% bugs removal, while R_2 and R_3 mostly offer lower percentage. Probably, the problem of fixing bugs after the selection deals the complexity of preparing technique, e.g., code deletion and/or addition. Therefore, designing test cases at the beginning of the process need to be done carefully.

TABLE I. Types of Test Case

Name	Program	T'	t_p	t_f	t_v
Tcas	α	1,608	1,267	241	100
Totinfo	$\alpha 2$	1,052	717	276	59
Schedule	β	2,650	2,308	271	71
Schedule2	$\beta 2$	2,710	2,401	289	20
Print-tokens	χ	4,130	3,802	255	73
Print-tokens2	$\chi 2$	4,115	3,722	297	96
Replace	δ	5,542	5,251	270	21

TABLE II. UNUSED AND USED TEST CASE

Program	t_u	t_o	t_e	t_c	t_d	t_g	t_n
α	1,138	310	365	463	129	108	21
$\alpha 2$	643	52	205	386	74	69	5
β	2,149	423	259	1,467	159	127	32
$\beta 2$	2,265	361	264	1,640	136	95	41
χ	3,595	259	314	3,022	207	181	26
$\chi 2$	3,352	397	202	2,753	370	343	27
δ	4,974	456	317	4,201	277	247	30

TABLE III. SIZE

Program	R_1	R_2	R_3	R_4
α	1,608	1,126	354	129
$\alpha 2$	1,052	736	305	74
β	2,650	1,882	636	159
$\beta 2$	2,710	1,897	650	136
χ	4,130	3,015	1,156	207
$\chi 2$	4,115	3,045	1,029	370
δ	5,542	4,046	1,607	277

TABLE IV. PERCENT REDUCING SIZE

Program	R_2	R_3	R_4
α	29.98	77.99	91.98
$\alpha 2$	30.04	71.01	92.97
β	28.98	76.00	94.00
$\beta 2$	30.00	76.01	94.98
χ	27.00	72.01	94.99
$\chi 2$	26.00	74.99	91.01
δ	26.99	71.00	95.00

TABLE V. PERCENT REDUCING SIZE OF R_4 VERSUS R_2 AND R_3

Program	R_2	R_3
α	62.00	13.99
$\alpha 2$	62.93	21.96
β	65.02	18.00
$\beta 2$	64.98	18.97
χ	67.99	22.98
$\chi 2$	65.01	16.02
δ	68.01	24.00

TABLE VI. NUMBERS OF BUG

Program	R_2	R_3	R_4
α	60	10	3
$\alpha 2$	66	17	2
β	67	15	1
$\beta 2$	63	10	2
χ	69	15	2
$\chi 2$	68	16	3
δ	65	20	1

TABLE VII. PERCENT BUGS

Program	R_2	R_3	R_4
α	5.33	2.82	2.33
$\alpha 2$	8.97	5.57	2.70
β	3.56	2.36	0.63
$\beta 2$	3.32	1.54	1.47
χ	2.29	1.30	0.97
$\chi 2$	2.23	1.55	0.81
δ	1.61	1.24	0.36

TABLE VIII. FIXED BUGS

Program	R_2	R_3	R_4
α	50	10	3
$\alpha 2$	50	6	2
β	24	6	1
$\beta 2$	22	9	2
χ	32	9	2
$\chi 2$	21	6	3
δ	34	6	1

TABLE IX. EFFICIENCY OF FIXING BUGS

Program	R_2	R_3	R_4
α	83.33	100.00	100.00
$\alpha 2$	75.76	35.29	100.00
β	35.82	40.00	100.00
$\beta 2$	34.92	90.00	100.00
χ	46.38	60.00	100.00
$\chi 2$	30.88	37.50	100.00
δ	52.31	30.00	100.00

V. CONCLUSION

Problem of offering test suite reduction claims knowledge of evading bugs and reducing size of modified programs. Random technique cannot solve problem of bugs that occurred in a program and cannot give low percent size reduction, while regression test reduction giver the better results. When compared all performances with the proposed model, we found that the traditional methods offers greater size and less percent of fixing bugs. These will affect executing, processing, and testing times throughout the whole processes of software maintenance.

ACKNOWLEDGMENT

The research project was funded by Assumption University.

REFERENCES

[1] V. Basili, L. Briand, S. Condon, Y.M. Kim, W.L. Melo, and J.D. Valett, "Understanding and Predicting the Process of Software Maintenance Releases," Proceedings of the 18th International Conference on Software Engineering, pp. 464-474, May 1996.

[2] G. Arnicans, and V. Arnicane, "Simplified Design of Test Cases Based on Models," TAPOST2011 12th Annual Software Testing Conference, pp. 28-29, May 2011.

[3] J.C. Burguillo, M. Llamas, and M.J. Fernandez, "Hueristic-driven Techniques for Test Case Selection," Elsevier Science B.V., pp. 51-66, 2002.

[4] W. Jin, A. Orso, and T. Xie, "Automated behavioral regression testing," 3rd International Conference on Software Testing, Verification and Validation, Washington DC, USA, 2010.

[5] X. Zhang, T.Y. Chen, and H. Liu, "An Application of Adaptive Random Sequence in Test Case Prioritization," ASE'09 24th International Conference on Automated Software Testing, pp. 233-244, November 2009

[6] S. Musa, A.B.M. Sultan, A.B.A. Ghani, and S. Bahaarom, "Regression Test Cases Selection for Object-Oreiented Programs based on AffectedStatements," International Journal of Software Engineering and Its Applications, vol. 9, no. 10, pp. 91-108, 2015.

[7] J.A. Jones, and M.J. Harrold, "Test-suite reduction and prioritization for modified condition/decision coverage," Proceedings of IEEE International Conference on Software Maintenance, pp. 92-101, 2001.

[8] Y. Yu, J. Jones, and M.J. Harrold, "An empirical study of the effects of test-suite reduction on fault location," ACM/IEEE 30th International Conference on Software Engineering, pp. 201-210, 2008

[9] L. Zhang, D. Marinov, L. Zhang, and S. Khurshid, "An Empirical Study of JUnit Test-Suite Reduction," IEEE 22nd International Symposium on Software Reliability Engineering, pp. 170-179, 2011.

[10] J. Prabhu, N. Malmurugan, G. Gunasekaran, and R. Gowtham, "Study of ERP Test-Suite Reduction Based on Modified Conditiona/Decision Coverage," 2nd International Conference on Computer Research and Development, pp. 373-378, 2010.

[11] H.K.N. Leung, and L. White, "A cost model to compare regression test strategies," Proceedings of Conference on Software Maintenance, pp. 201-208, 1991.

[12] Z.Q. Zhou, "Using Coverage Information to Guide Test Case Selection in Adaptive Random Testing," IEEE 34th Annual Computer Software and Applications Conference Workshops (COMPSACW), pp. 208-213, 2010.

[13] G. Rothermel, and M.J. Harrold, "Empirical studies of a safe regression test selection technique," IEEE Transactions on Software Engineering, pp. 401-419.