



Test Cases Minimization and Prioritization Based on Requirement, Coverage, Risk Factor and Execution Time

Wasiur Rhmann¹, Taskeen Zaidi¹ and Vipin Saxena^{1*}

¹Department of Computer Science, B. B. Ambedkar University, Lucknow, India.

Authors' contributions

This work was carried out in collaboration between all authors. Author VS designed the study, wrote the protocol and supervised the work. Authors WR and TZ carried out all laboratories work and performed the statistical analysis. Author WR managed the analyses of the study. Author VS wrote the first draft of the manuscript. Author TZ managed the literature searches and edited the manuscript. All authors read and approved the final manuscript.

Article Information

DOI: 10.9734/BJMCS/2016/23269

Editor(s):

(1) Dariusz Jacek Jakóbczak, Chair of Computer Science and Management in this Department, Technical University of Koszalin, Poland.

Reviewers:

- (1) Lirong Wang, Stonybrook University, USA.
(2) M. Bhanu Sridhar, GVP College of Engineering for Women, Visakhapatnam, India.
Complete Peer review History: <http://sciencedomain.org/review-history/13030>

Received: 23rd November 2015

Accepted: 18th December 2015

Published: 20th January 2016

Original Research Article

Abstract

Large numbers of test cases are designed for effectively testing the quality of the developed software products. Due to limited resource and time constraint it is not possible to test the software with large number of test cases. Test case minimization selects the test cases from test suites which have higher probability of finding errors. Test case prioritization effectively improves various performance goals by executing test cases in appropriate order. This paper presents a test case minimization and prioritization approach based on several factors related to the software projects. Proposed approach prioritizes the test cases based on faults exposed by test cases, requirement coverage, risk, statement coverage and test case execution time. In the present work, test cases are selected and prioritized within the given time constraint.

Keywords: Test suite reduction; test case prioritization; requirement priority; risk.

*Corresponding author: E-mail: vsax1@rediffmail.com;

1 Introduction

Software testing is a quality assurance activity and mainly responsible for producing reliable software. The objective of testing is to find the maximum possible errors within realistic time span by putting manageable amount of effort for development of the software product. Test case execution satisfies preconditions and provides test case input and gives output which is compared with the expected output to ensure that it satisfies expected post conditions to determine whether the test passed [1]. A good test case has high probability of finding undiscovered errors [2]. In test suite reduction a subset of test cases is selected for testing the software to uncover errors while maintaining the quality of the software. Prioritization of test cases can detect faults earlier in the software testing phase. In software projects, almost 40-50% of the total efforts are consumed by software testing. Test case prioritization is done with the intent of earlier fault detection from test suite. A huge amount of time is spent in testing phase by software developer. Different requirement of the software projects have different priority and almost 45% of the requirements of the software products are rarely used [3]. Therefore, for test case prioritization, we also considered the priority of the requirement based on their importance to different man-powers involved in software projects. Test case prioritization can adjust testing efforts which is due to limited resource and budget constraints. Due to this various researchers have actively studied test case prioritization techniques. Srivastava [4] presented a new test case prioritization algorithm in which average fault per minutes are calculated. Majority of researchers used source code for test case prioritization while some researchers also investigated test case prioritization techniques based on different software artifacts like design, document and system requirements [5]. For test case prioritization, many researchers used software risk information to test the portion of the software code which is more error prone [6-8]. Srikanth et al. [9] have proposed a technique of test case prioritization. In this, authors used several requirement related components like requirements volatility and requirement complexity for early detection of the faults in software projects. Arafeen and Do [10] used text mining technique to cluster similar requirements then relationship between requirements and test cases is used for prioritization of test cases. Authors find out that use of requirement information for test case prioritization improved the results. Rothermal et al. [11] defined the test case prioritization as follows:

Given: T is a test suite; PT is the set of permutations of T ; f is a function from PT to the real numbers.

Problem: To find T' belongs to PT such that (for all T'') (T'' belongs to PT) ($T' \neq T''$) [$f(T') \geq f(T'')$].

Here, PT represents the set of all possible prioritizations (orderings) of T and f is a function that, applied to any such ordering, yields an award value for that ordering.

Test case minimization is also explained in [12] and described below in brief:

Given: A test suite $T(t_1, t_2, \dots, t_n)$ and a set of test requirements $R(r_1, r_2, \dots, r_n)$

Problem: To find the smallest T_0 such that T_0 is subset of T , for all r element R (T_0 satisfies r).

The objective of present work is to develop test case prioritization techniques based on the requirement priority and risk factor associated with the software projects and statement coverage within the given time constraint. Proposed test case prioritization technique maximizes fault exposed and statement coverage of test cases.

2 Research Methodology

The procedure of test case selection and prioritization is described below:

2.1 Requirement priority

Customer, Manager and developer assign different values from 1 to 10 to the requirements based on their importance and sum of these values for each requirement is calculated to assign the priority to the

requirement. Let us consider two requirements like R₁ and R₂ and assign the priority factors as shown in the following Table 1.

Table 1. Requirement priority table with prioritization factor (sample values)

| Requirements | Prioritization factor (1 to 10) | | | Total |
|----------------|---------------------------------|-----------|----------|-------|
| | Manager | Developer | Customer | |
| R ₁ | 4 | 4 | 5 | 13 |
| R ₂ | 4 | 3 | 8 | 15 |

2.2 Risk exposure

Let us consider four types of risk which may occur in the software projects [13]. Risk exposure is computed by the formula

$$\text{Risk-Exposure (RE)} = \text{Probability of occurrence of risk (P)} * \text{Severity (S)} \tag{1}$$

Then risk exposure for each requirement is the sum to calculate total risk exposure for each requirement. For calculating risk factor, several problems which may occur in the development of the software project are listed then their probability of occurrence and severity is assigned to each problem. Historical data of some projects can be used for calculating the probability and severity values which are assigned based on established data [13]. Let us consider four types of attacks which may occur during the development of the software project, these are

- LP → Loss of Power
- CFD → Corrupt File Data
- UUD → Unauthorized User Access
- ST → Slow Throughput

Then the following table is constructed based on the values of P and S for R₁ and R₂.

Table 2. Requirement and risk factor (sample values)

| Requirement | Risk | Prioritization factor (value 1 to 10) | | | | Total of risk exposure |
|----------------|------|---------------------------------------|-----|-----|----|------------------------|
| | | LP | CFD | UUD | ST | |
| R ₁ | P | 2 | 2 | 2 | 3 | 30 |
| | S | 4 | 5 | 3 | 2 | |
| | RE | 8 | 10 | 6 | 6 | |
| R ₂ | P | 3 | 4 | 5 | 4 | 37 |
| | S | 4 | 2 | 3 | 5 | |
| | RE | 12 | 8 | 8 | 9 | |

Here we find the subset of test suites such that it covers maximum requirement specified and risk of the requirement while maintaining statement coverage and fault detection capability high within the given time. In the present work, test case selection is based on 0-1 integer programming [14] of optimization techniques.

2.3 0-1 integer programming

Let us describe 0=1 integer programming which has the following objective function and associated constraint:

$$\text{Max } z = \sum_1^n w_i t_i$$

$$\text{Where, } w_i = S_i + F_i + RP_i + R_i \tag{2}$$

In the above equation, S_i is statement coverage of test case, F_i is fault exposed by test case t_i , RP_i is requirement priority value covered by a test case, R_i is risk exposure covered by a test case. The decision variable t_i takes only 0 and 1 value. If test case i is selected then its value will be 1 otherwise 0. Hence the constraint is given below:

$$\sum_1^n t_{ei}t_i \leq T$$

where t_{ei} is test case execution time for test case t_i , T is total time allocated for testing the project.

3 A Case Study

Let us consider a small software program as a case study for cash withdrawal from bank. Here we considered a case of cash withdrawal from ATM which uses fingerprint along with the pin number to provide additional security. Code of software contains a method of cash withdrawal which takes four parameter namely Pin number, Fingerprint, Account type and Amount of withdrawal. The Pin number is 1234, Account type is savings and maximum withdrawal limit is 50,000 and Fingerprint is fingerprint (OK). Here fingerprint is taken as correct input while wfingerprint (wrong fingerprint) for unauthorized access. Different requirements of this software are listed below:

- Req1: Appropriate message should be display if user enters wrong pin number;
- Req2: A message of fingerprint not matched
- Req3: Appropriate message should appear on the screen if user does not select correct account type
- Req4: A message of insufficient amount available should be displayed if user enters amount of withdrawal more than the available balance
- Req5: Appropriate message should appear if user tries to withdraw amount more than withdrawal limit
- Req6: Appropriate message of withdrawal should appear after successful withdrawal

For the above, a code in JAVA is given below:

```

1 void withdraw (PIN, FINGERPRINT, ACC_TYPE, AMOUNT){
2 if (PIN!='1234')
3 {System.out.print("Please enter correct pin");}
4 else{
5 if( FINGERPRINT !='Ok')
6 {System.out.print("Fingerprint not matched");}
7 else{
8 if( ACC_TYPE!='Saving')
9 {System.out.print("Please enter correct account type");}
10 else{
11 if( AMOUNT>balance)
12 { System.out.print("Not sufficient amount");}
13 else if( AMOUNT<Balance && AMOUNT> 50000)
14 { System.out.print("Enter amount should be less than 500000");}
15 else{
16 System.out.print("Balnce"+Balance-AMOUNT); } } } }
```


In the above program, there are six possible outcomes hence, there are six equivalence classes. These six equivalence classes are used for test case generations which are as follows:

- C1= {<PIN, FINGERPRINT, ACC_TYPE, AMOUNT>, Pin is wrong}
- C1= {<PIN, FINGERPRINT, ACC_TYPE, AMOUNT>, Fingerprint not matched}
- C3= {<PIN, FINGERPRINT, ACC_TYPE, AMOUNT>, Please eneter correct account type}
- C4= {<PIN, FINGER_PRINT, ACC_TYPE, AMOUNT>, Not Sufficient Amount}
- C5= {<PIN, FINGER_PRINT, ACC_TYPE, AMOUNT>, Enter amount should be less than 50000}
- C6= {<PIN, FINGERPRINT, ACC_TYPE, AMOUNT>, Balance}




On the basis of above equivalence classes, various test cases are generated and represented below in Table 3.

Table 3. Test case creation from JAVA code

| Test case | Input data | Expected output |
|----------------|-------------------------------------|------------------------------------|
| T ₁ | (12,'Fingerprint','Saving',5) | Please enter correct Pin number |
| T ₂ | (1234,'wFingerprint','Current',5) | Please scan your Fingerprint |
| T ₃ | (1234,'Fingerprint','Current',5) | Please choose correct Account type |
| T ₄ | (1234,'Fingerprint','Saving',6000) | Not Sufficient balance available |
| T ₅ | (1234,'Fingerprint','Saving',65000) | Amount should be less than limit |
| T ₆ | (1234,'Fingerprint','Saving',5) | Shows available balance |

In the above program, authors introduced three bugs to make mutants. Bugs inserted are shown below as  in the same program.

```

1 void withdraw(PIN, FINGERPRINT, ACC_TYPE, AMOUNT){
2 If (PIN  "==" "" 1234'){
3 System.out.println("Please enter correct pin number"); }
4 else{
5 If( FINGERPRINT !='OK'){
6 System. out. print (" Fingerprint not matched");}
7 else {
8 If (ACC_TYPE  "==" 'Saving')
9 {System. out. println ("Please enter correct account type");}
10 else {
11 if (AMOUNT>Balance)
12 {System. out. println ("Not sufficient amount"); }
13 else if (AMOUNT  ">" Balance && AMOUNT>50000)
14 {System. out. Println ("Enter amount should be less than 50000");}
15 else
16 {System.out.print("Balance"+Balance-AMOUNT); } } } }

```

Total mutant inserted are=3

Mutant exposed by test cases are (1, 1, 2, 2, 2) respectively and statement coverage of test cases is (3, 5, 7, 9, 10, 11), respectively. Now, requirement priority of test cases are recorded in the following Table 4.

Table 4. Setting the priority factor for requirements

| Requirement | PF value (1 to 10) | | | Total |
|----------------|--------------------|-----------|---------|-------|
| | Customer | Developer | Manager | |
| R ₁ | 5 | 5 | 5 | 15 |
| R ₂ | 5 | 5 | 5 | 15 |
| R ₃ | 6 | 6 | 6 | 18 |
| R ₄ | 7 | 7 | 7 | 21 |
| R ₅ | 8 | 8 | 8 | 24 |
| R ₆ | 9 | 9 | 9 | 27 |

Let us introduce the risk factor for the requirements and it is given below in the following Table 5.

Table 5. Requirement with their risk factor and priority factor value

| Requirement | Risk | PF value | | | | Total of risk exposure |
|----------------|------|----------|-----|-----|----|------------------------|
| | | LP | CFD | UUD | ST | |
| R ₁ | P | 2 | 2 | 2 | 3 | 30 |
| | S | 4 | 5 | 3 | 2 | |
| | RE | 8 | 10 | 6 | 6 | |
| R ₂ | P | 3 | 4 | 5 | 4 | 37 |
| | S | 4 | 2 | 3 | 5 | |
| | RE | 12 | 8 | 8 | 9 | |
| R ₃ | P | 3 | 3 | 4 | 2 | 40 |
| | S | 4 | 4 | 2 | 4 | |
| | RE | 12 | 12 | 8 | 8 | |
| R ₄ | P | 2 | 3 | 2 | 6 | 35 |
| | S | 4 | 3 | 3 | 2 | |
| | RE | 8 | 9 | 6 | 12 | |
| R ₅ | P | 2 | 4 | 3 | 5 | 41 |
| | S | 5 | 3 | 3 | 2 | |
| | RE | 10 | 12 | 9 | 10 | |
| R ₆ | P | 3 | 4 | 2 | 4 | 43 |
| | S | 5 | 3 | 4 | 2 | |
| | RE | 15 | 12 | 8 | 8 | |

The following Table 6 represents the statement and test case coverage criteria for the statements taken from JAVA code from line numbers 1-16.

Table 6. Statement and test case coverage taken from JAVA code

| Statement/Test cases | T ₁ | T ₂ | T ₃ | T ₄ | T ₅ | T ₆ |
|----------------------|----------------|----------------|----------------|----------------|----------------|----------------|
| 1 | × | × | × | × | × | × |
| 2 | × | × | × | × | × | × |
| 3 | × | | | | | |
| 4 | | × | × | × | × | × |
| 5 | | × | × | × | × | × |
| 6 | | × | | | | |
| 7 | | | × | × | × | × |
| 8 | | | × | × | × | × |
| 9 | | | × | | | |
| 10 | | | | × | × | × |
| 11 | | | | × | × | × |
| 12 | | | | × | | |
| 13 | | | | | × | × |
| 14 | | | | | × | |
| 15 | | | | | | × |
| 16 | | | | | | × |

From Tables 4, 5 and 6 and mutant exposed as (1, 1, 2, 2, 2, 2) are considered to find statement coverage, fault exposed, requirement priority and risk represented in following table. For the execution time, it is considered as a twice the statement covered by the test case. It is recorded in the last row of the table.

Table 7. Test case with statement, fault, requirement and risk coverage

| Test case | T ₁ | T ₂ | T ₃ | T ₄ | T ₅ | T ₆ |
|----------------------|----------------|----------------|----------------|----------------|----------------|----------------|
| Statement coverage | 3 | 5 | 7 | 9 | 10 | 11 |
| Fault exposed | 1 | 1 | 2 | 2 | 2 | 2 |
| Requirement priority | 15 | 15 | 18 | 21 | 24 | 27 |
| Risk | 30 | 37 | 40 | 35 | 41 | 43 |
| Total (T) | 49 | 58 | 67 | 67 | 77 | 83 |
| Execution time (ms) | 6 | 10 | 14 | 18 | 20 | 22 |

On the basis of above, let us formulate the 0-1 integer linear programming problem which is given below. Total time of executing test cases is considered to be 70 ms.

$$\text{Max } Z=49T_1+58T_2+67T_3+67T_4+77T_5+83T_6$$

Subject to:

$$6T_1+10T_2+14T_3+18T_4+20T_5+22T_6 \leq 70 \tag{3}$$

4 Results and Discussion

For the solution of the problem 3, authors used LINGO 15.0 software to solve the formulated optimization problem. Different types of equations like linear, nonlinear, quadratic, integer optimization problems can be easily solved by LINGO. Optimization models can be easily expressed by using integrated package of LINGO. It provides an environment for building, editing problems and solves them with the available build-in solvers. In the reduced test suite, we find T₁, T₂, T₃, T₄ and T₆ as test cases and these should run in the given time constraint to cover maximum requirement, risk, statement coverage and faults. For selected test cases by the software, authors calculated different factors recorded in the following Table 8.

Table 8. Percentage coverage of different factors

| Test cases | T ₁ | T ₂ | T ₃ | T ₄ | T ₆ | % covered |
|---------------------|----------------|----------------|----------------|-------------------------|---------------------------------|--------------------------------------|
| Fault identified | 1 | 1 | 2 | 2 | 2 | 2/2=100% |
| Statement covered | 1,2,3 | 1,2,4,5,6 | 1,2,4,5,7,8,9 | 1,2,4,5,7,8, 10, 11, 12 | 1,2,4,5,7,8, 10, 11, 13, 15, 16 | 1,2,3,4,5,6,7,8, 9,10,11,12,13,15,16 |
| Requirement covered | 15 | 15 | 18 | 21 | 27 | 96/120=80.0% |
| Risk covered | 30 | 37 | 40 | 35 | 43 | 185/226=81.85% |

From the above table, prioritized order of test cases based on requirement and risk is T₆, T₃, T₄, T₂ and T₁ which is in the decreasing order of requirement and risk. This is the order of test case which covers maximum statements within the given time. Here test cases which have higher coverage of requirement and have probability of exposing more number of errors are tested first. It is recorded in the following Table 9.

Table 9. Prioritized order of test cases

| Test case | Requirement+Risk | Total |
|----------------|------------------|-------|
| T ₁ | 15+30 | 45 |
| T ₂ | 15+37 | 52 |
| T ₃ | 18+40 | 58 |
| T ₄ | 21+35 | 56 |
| T ₆ | 27+43 | 70 |

5 Conclusions

In the presented work, authors have proposed a test case selection and prioritization technique using 0-1 integer programming based on requirement priority and risk severity and statement coverage. The first test cases are selected from test suite based on given time constraint. Selected test cases cover maximum faults, statements, requirement and risk. Then test cases are prioritized based on requirement and risk values. A small JAVA method for cash withdrawal is considered for validation of proposed approach. Here 0-1 integer programming is used as each decision variable can have only two values 0 or 1. For 1 a decision variable is selected and for 0 a decision variable is not selected. In case of software testing, minimized test cases can be selected from large number of test cases. If a test case is selected then value of corresponding decision variable will be 1 otherwise 0. In the present work, requirement of cash withdrawal are written manually, statement covered by test cases and fault exposed by test cases are also calculated manually so authors considered simple example of cash withdrawal from ATM. In future we may consider complex and large software for test case prioritization and automated tool for statement coverage and fault exposing potential may also be considered.

Competing Interests

Authors have declared that no competing interests exist.

References

- [1] Jorgensen PS. Software testing: A craft's man approach, 4th edition. CRC Presses Taylor and Francis Group; 2013.
- [2] Pressman RS. Software engineering: A practitioner's approach. McGraw Hill; 2013.
- [3] Srikanth H, Williams L. On the economics of requirements based test case prioritization. Proceeding of the seventh international workshop on Economics-driven software engineering research. New York, USA; 2005;1-3.
- [4] Srivastava PR. Test case prioritization. Journal of Theoretical and Applied Information Technology. 2008;4(2):178-181.
- [5] Krishnamoorthi R, Mary SASA. Factor oriented requirement coverage based system test case prioritization of new and regression test cases. Information and Software Technology. 2009;51(4):799-808.
- [6] Do H, Mirarab S, Tahvildari L, Rothermel G. The effects of time constraints on test case prioritization: A series of controlled experiments. IEEE Transaction on Software Engineering. 2010;26(5):593-617.
- [7] Yoon M, Lee Y, Song M, Choi Bi. A test case prioritization through correlation of requirement and risk. Journal of Software Eng. Appl. 2012;5(10):823-835.
- [8] Stallbaum H, Metzger A, Pohl K. An automated technique for risk based test case generation and prioritization. In: Proceedings of the 3rd International Workshop on Automation of Software Test, New York, USA. 2008;67-70.
- [9] Srikanth H, Williams L, Osborne J. System test case prioritization of new and regression test cases. In: Proceedings of International Symposium on Empirical Software Engineering. 2005;64-73.

- [10] Arafeen MJ, Do H. Test case prioritization using requirements based clustering. Proceedings of IEEE Sixth International Conference of Software Testing Verification and Validation (ICST), Washington, USA. 2013;312-321.
- [11] Elbaum S, Malishevsky AG, Rothermel G. Test case prioritization: A family of empirical studies. IEEE Transaction on Software Engineering. 2002;28(2):159-182.
- [12] Mudgal AP. A proposed model for minimization of test suite. Journal of Nature Inspired Computing. 2013;1(2):34-37.
- [13] Tamres L. Introducing software testing. 1st Edition, Addison Wesley; 2006.
- [14] Williams HP. Model building in mathematical programming. John Wiley, New York; 1993.

© 2016 Rhmann et al.; This is an Open Access article distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/4.0>), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Peer-review history:

The peer review history for this paper can be accessed here (Please copy paste the total link in your browser address bar)

<http://sciencedomain.org/review-history/13030>