HTTCS: Hybridization Technique for Test Case Selection

Adtha LAWANNA

Department of Information Technology, Vincent Marry School of Science and Technology, Assumption University, Samut Prakarn 10540, Thailand

(Corresponding author's e-mail: adtha@scitech.au.edu)

Received: 17 September 2014, Revised: 18 July 2018, Accepted: 12 August 2018

Abstract

One problem found within the process of software maintenance is that the size of the selected test cases is large. This causes the ability of the whole process of software-development life cycle to drop. Particularly, it may be time consuming and cause delays, and the cost may be expensive. The selection of test cases for software maintenance depends more on the criticality of fixing bugs than the criticality of avoiding programming errors. Therefore, selection methods are proposed, such as test-all, random, and regression selection. This includes Technique for Test Case Selection (TTCS) and the improvement of Test Case Selection (TCS). These techniques can provide better results, in particular, giving smaller sizes, reduction rates, and % problem-solving than traditional techniques. However, this paper proposes a new model, which is a combination of using the process of determining an appropriate number of selected test cases regarding TTCS, and TCS with testing-based selection, named the Hybridization Technique for Test Case Selection (HTTCS). Obviously, HTTCS can reduce the size of the selected test cases by about 96.86 - 98.83 %, which is better than TTCS and TCS, by about 0.29 - 16.51 %. Additionally, using HTTCS can increase the % problem-solving by up to 99.98 %, is which higher than others about at most 0.66 %.

Keywords: Software maintenance, selection, coverage, test case, hybridization

Introduction

The Software-Development Life Cycle contains several processes, which include getting requirements, coding, testing, and maintenance [1]. This paper focuses on the process of software maintenance, in which problems can remain after using selection technique for choosing appropriate test cases used in the entire process of testing programs and fixing problems, e.g., bugs or faults [2].

Problems in software maintenance concerns selecting test cases from a test pool, which may depend upon the numbers of function, lines of code, or fault version. Accordingly, there have been many methods proposed for reducing the numbers of selected test cases in order to provide a smaller size of testing and keeping competency. In the past, the retest-all technique was used to maintain and modify programs regarding changes of user requirements, which were business-driven. Unfortunately, this could not deal with the time consumption and cost of retaining codes. During that time, the random technique was first proposed to control the situation. This results in good selection in terms of reducing the numbers of selected test cases, but it cannot guarantee the accuracy of the maintenance. Therefore, regression methods are proposed for a good selection, which are better than traditional methods. They provide smaller numbers of selected test cases. Accordingly, testing time and competency are preserved. However, many techniques have been proposed for improving the ability of selection, e.g., Technique for Test Case Selection (TTCS) and the improvement of Test Case Selection (TCS). Accordingly, using these techniques, the entire performances are improved. In particular, the numbers of selected test cases are small, and the competency is preserved. However, this paper looks for a better method to increase the ability of reducing the numbers of the selected test cases. In the meantime, problem-solving is still preserved. This paper proposes a new technique named Hybridization Technique for Test Case Selection (HTTCS). The objectives of proposing HTTCS are shown as follows;

(1) To provide minimum numbers of the selected test cases compared with the comparative studies.

(2) To provide a higher % of problem-solving in terms of reducing bugs that could occur during the process of testing.

This paper includes the process of software maintenance, which is concerned with conducting change in this part of the software-development life cycle. Developers interact continually with teams, clients, and users in order to detect faults. Testers must be good investigators, testing programs by using a set of selected test cases thoroughly and chasing the sources of the errors [3]. Therefore, the HTTCS is proposed to handle these problems.

Materials and methods

Definition

Table 1 is an explanation of terms used in this paper. There are 3 comparative studies presented, which use TTCS, TCS, and HTTCS, respectively. Additionally, some terms are represented for more understanding.

Table 1 Definitions used in HTTCS.

Symbol	Description
N	Numbers of functions
L	Lines of code
F	Faulty versions
TTCS	Technique for Test Case Selection in Software Maintenance
TCS	Improvement of Test Case Selection
HTTCS	Hybridization Technique for Test Case Selection
T	Numbers of test case regarding *f(N,L,F)*
*T**	Numbers of test case regarding *f(L,F)*
*T***	Numbers of test case regarding *f(F)*
T	Selected test case
Cov(L)	Coverage area of *L*
Cov(N)	Coverage area of *N*
Cov(F)	Coverage area of *F*

Dataset

A summary of the assets of the 7 programs is found in **Table 2**. Accordingly, "*L*" refers to the lines of code. Additionally, "*N*" is the number of functions, and "*F*" the faulty versions. The experiments needed a set of 7 well-known subject programs written in C. They were established by the Siemens suite of programs with hand-scattered bugs or faults, first used by Hutchins *et al*. [4]. These programs have subsequently been modified and extended by other agents, particularly Rothermel and Harrold [5,6] and Graves *et al*. [7]. These programs are chosen because of their historical significance.

Table 2 Average test cases in each test suite by Rothermel and Harrold.

Name	***N***	***L***	***F***	***T***
Print-tokens	18	402	7	4,130
Print-tokens2	19	483	10	4,115
Replace	21	516	32	5,542
Schedule	18	299	9	2,650
Schedule2	16	297	10	2,710
Tcas	9	148	41	1,608
Totinfo	7	346	23	1,052

Methods

TTCS: Technique for Test Case Selection in Software Maintenance [8]

TTCS is test case selection based on determining the numbers of test cases by using the integral technique on factors, e.g., *N*, *L*, and *F*. Particularly, the numbers of selected test cases are equivalent to $\int_0^n \int_0^l \int_0^{\mathrm{f}} f(N,L,F)\,dndld\mathrm{f}$ over the total number of test cases existing in a test pool. There are 2 methods created, which are described as follows;

Method 1: Finding the relationship of $f(N,L,F)$

```
If $\sum T = \int_0^n \int_0^l \int_0^{\mathrm{f}} f(N,L,F)\,dndld\mathrm{f}$    (1)
Then compute
$\sum T = \int_0^n \int_0^{\mathrm{f}} \int_0^{\mathrm{l}} f(N,F,L)\,dnd\mathrm{f}ldl$    (2)
    ElseIf
        $\sum T = \int_0^{\mathrm{f}} \int_0^{\mathrm{n}} \int_0^{\mathrm{l}} f(F,N,L)\,d\mathrm{f}dndl$    (3)
        Then compute
            $\sum T = \int_0^{\mathrm{f}} \int_0^{\mathrm{l}} \int_0^{\mathrm{n}} f(F,L,N)\,d\mathrm{f}dldn$    (4)
            ElseIf
                $\sum T = \int_0^{\mathrm{l}} \int_0^{\mathrm{f}} \int_0^{\mathrm{n}} f(L,F,N)\,dld\mathrm{f}dn$    (5)
                Then compute
                    $\sum T = \int_0^{\mathrm{l}} \int_0^{n} \int_0^{\mathrm{f}} f(L,N,F)\,dldndf$    (6)
            EndIf
    EndIf
End
```

where, over a specific (*N*,*L*), the variable *F* is restricted between *g(*N*,L)* and *h(N,L)* and, for a precise *N*, the variable *L* is restricted between *s(N)* and *t(L)*.

Method 2: Finding the total numbers of test cases

```
If $T = \int_0^n \int_0^l \int_0^{\mathrm{f}} f(N,L,F)\,dndld\mathrm{f}$ where $0 \le N \le n$
Then select $T$
```

```
    ElseIf
        $T^* = \int_1^l \int_1^f f(L,F)\,dl\,df \; where \; 0 \le L \le l$
        Then select $T^*$
                ElseIf $T^{**} = \int_0^f f(F)\,df \; where \; 0 \le F \le f$
                    Then select $T^{**}$
                    EndIf
    EndIf
End
```

Method 3: Selecting the test cases

The algorithm for selecting the test cases have been found through 3 situations, described as follows;

Situation 1: $T = \int_0^n \int_0^l \int_0^f f(N,L,F)\,dn\,dl\,df$

(1) Check frequency of each *T*
(2) Find Frequency$_{max}$
(3) Select the test cases that have Frequency$_{max}$
(4) (Re)find Frequency$_{max}$
(5) Do (2) to (4) until the numbers of selected test cases equivalent to *T*.

Situation 2: $T^* = \int_1^l \int_1^f f(L,F)\,dl\,df$ (7)

(1) Check frequency of each *T**
(2) Find Frequency$_{max}$
(3) Select the test cases that have Frequency$_{max}$
(4) (Re)find Frequency$_{max}$
(5) Do (2) to (4) until the numbers of selected test cases equivalent to *T**.

Situation 3: $T^{**} = \int_0^f f(F)\,df$ (8)

(1) Check frequency of each *T***
(2) Find Frequency$_{max}$
(3) Select the test cases that have Frequency$_{max}$
(4) (Re)find Frequency$_{max}$
(5) Do (2) to (4) until the numbers of selected test cases equivalent to *T***.

TCS: Improvement of Test Case Selection [9]

TCS is the technique of finding selected test cases regarding a designing test case template, creating the details of test cases and testing help, producing a test pool, and selecting test cases. There are several steps, described as follows;

Step 1: Create Test Case Template
Step 2: Assign Details of Test Cases
Step 3: Create Testing Help
Step 4: Find Numbers of Test Cases

$$T = \frac{1}{TotalTestCase} \int_0^n \int_0^l \int_0^f f(N,L,F)\,dn\,dl\,df \tag{9}$$

Step 5: Find coverage value

In this section, the details of the algorithm are created and used to determine the test cases in each test pool due to the subject program.

If coverage area of *L = f(N,F),or Cov(L)* (10)

```
Then    $Cov(L) = \frac{L}{T}$                                                    (11)
    ElseIf  coverage area of $N = f(L,F)$, or $Cov(N)$                          (12)
    Then    $Cov(N) = \frac{N}{T}$                                              (13)
            ElseIf  coverage area of $F = f(L,N)$ or $Cov(F)$                   (14)
            Then $Cov(F) = \frac{F}{T}$                                         (15)
            EndIf
    EndIf
End
```

$$T = \sum (Cov(L) + Cov(N) + Cov(F)) \quad (16)$$

Method 2: Selecting the test cases

```
Algorithm for selecting the test cases
(1) Check coverage of each $T$
(2) Find $\text{coverage}_{max}$
(3) Select the test cases that have $\text{coverage}_{max}$
(4) (Re)find $\text{coverage}_{max}$
(5) Do (2) to (4) until the numbers of selected test cases equivalent to $T$.
```

Accordingly, this algorithm will be applied in order to find the numbers of test cases when the coverage is determined in regards to *N* or *F*.

HTTCS: Hybridization Technique for Test Case Selection

The Overview of HTTCS

HTTCS is the hybridization of using TTCS and testing case technique, which concerns solving failures of the cases. As shown in **Figure 1**, there are 6 steps proposed, as follows;

Process 1: Define factors

In the process of software maintenance, there are many factors to consider, such as the numbers of programmers, the ability of each programmer, testing time, bugs or faults, requirements, function, and code. The first assumption is for a well-defined factor required for characterizing them to be useful in all processes of testing software. For example, objects are proposed regarding their attributes. In this paper, scale data is necessary. Therefore, each factor presents its value.

Process 2: Determine related functions

This means finding the relationships among factors existing in each subject program. On this point, different functions can result in appropriate, or worse, testing. In particular, complexity can occur when the numbers of functions increase. Therefore, it is important to determine what should be the related factors.

Process 3: Use the integral technique

According to the hybridization technique, this step is applied to use method 1 and 2 of TTCS because it gives the appropriate numbers of test cases that need to be tested. The benefit of TTCS is that it can use simple computations but can give high accuracy. All related factors are functioned to perform the size of test cases, which can be created for testers to utilize the efficiency of modifying the whole previous software to be adopted in the process of program maintenance.

Process 4: Test case

Assume that, if testing is done, the statement will be “pass” or “fail”. Accordingly, the test cases are tested and give 2 results, which can be represented by “1” if it passes, or “0” if it fails. In this paper, the "fail" will be important for the next step. This is because it may affect the entire program. Regarding this, bugs or faults exist and need to be fixed.

Process 5: Select test case
Test cases that have a result of "fail" are required and, thus, selected. On the other hand, "pass" test cases are not selected, as these refer to cases with no bugs to be fixed.

Process 6: Fix problem
This step means fixing bugs for test cases that are selected from the test pool. However, this paper does not show the details of fixing the bugs, which occur differently in the test pool.

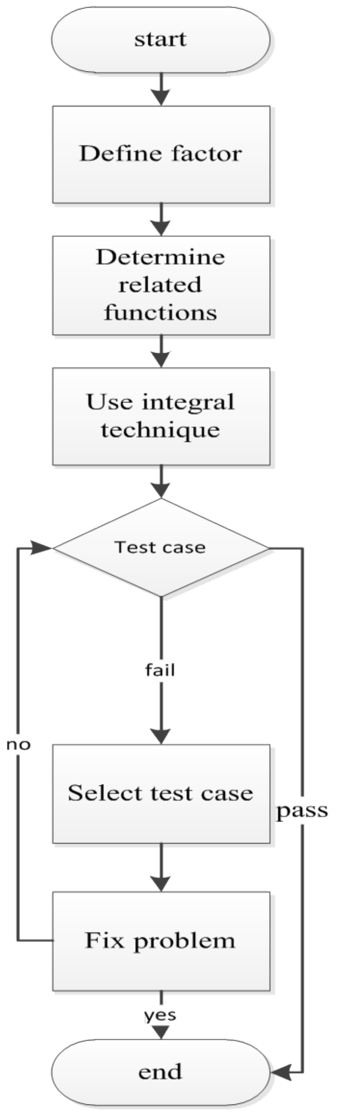

Figure 1 Overview of HTTCS.

The Conceptual Model of HTTCS

This section presents the proposed model, named HTTCS, which gives a higher performance than the comparative studies. It combines 2 concepts, one from using TTCS, and another based on testing.

Step 1: Designing test cases by method 1&2 of TTCS

$$T = \{t_1, t_2, t_3, ..., t_n\} \tag{17}$$

Case I: Retrieve factor *N*, *L*, and *F* respectively

```
If function $= N + L + F$    (18)
    Then compute the numbers of $T$
        $T = \int_0^n \int_0^f \int_0^l (N + L + F)\, dn\, df\, dl$    (19)
    End
Case II: Retrieve factor $F$,$N$, and $L$ respectively
If function $= F + N + L$    (20)
    Then compute the numbers of $T$
        $T = \int_0^f \int_0^n \int_0^l (F + N + L)\, dn\, df\, dl$    (21)
End
Case III: Retrieve factor $L$,$F$, and $N$ respectively
If function $= L + F + N$    (22)
    Then compute the numbers of $T$
        $T = \int_0^f \int_0^l \int_0^n (L + F + N)\, df\, dl\, dn$    (23)
End
```

According to Case I throughout III, the results of using these algorithms will be shown as;

$$T = \frac{N^2LF + NL^2F + NLF^2}{2} \tag{24}$$

In general, when factors are increased, they can be computed simply. For example, if factor time (*TM*) and number of programmers are added, then the computation will be noted as;

$$T = \frac{N^2LF(TM)P + NL^2F(TM)P + NLF^2(TM)P + NLF(TM)^2P + NLF(TM)P^2}{2} \tag{25}$$

However, these can be used for determining only the numbers of the relevant test cases available in the test pool. It does not present the methods of selecting what the selected test cases should be.

Step 2: Testing

Testing cases give 2 results, which are "pass" or "fail", shown as;

$$t = \begin{cases} 0 & t_* = fail \\ 1 & t_{**} = pass \end{cases} \tag{26}$$

```
Do
    Read $t$
Case
    When    $t_* = 0$
                $t_* = fail$
            When   $t_{**} = 1$
            $t_{**} = pass$
EndCase
```

```
Until        $t$ is empty
Write
    $t = t_*$
End
```

The results of testing give 2 answers, which are "pass" or "fail". This algorithm is proposed for selecting those test cases where the answers are "fail" until no cases are found anymore. According to this, the bugs that could occur are solved later.

Step 3: Selecting test case

```
If $t_* = 0$ then
    Select $t_*$
            ElseIf $t_{**} = 1$ then
                    Remove $t_{**}$
        EndIf
    End
```

Accordingly, the numbers of selected test cases are chosen, whereas the results of testing are "fail". This refers to the probability that problems are found in those selected cases. On the other hand, cases which are testing result in "pass" had no problems found in those cases. Therefore, the selection focuses on test cases that have problems to be selected test cases, because they may affect the whole process of testing software.

Results and discussion

The numbers of relevant test cases are reported in **Table 2** by using the concept of integral technique, which needs 3 factors, e.g., *N*, *L*, and *F* [5,6].

For example, determination of the numbers of test cases for a subject program named Print-tokens is explained as follows [8];

By TTCS

$$T = \int_0^n \int_0^l \int_0^f f(N, L, F)\,dndldf \div N \times L \times F$$

$$T = \int_0^n \int_0^l \int_0^f (N + L + F)\,dndldf \div N \times L \times F$$

$$T = \frac{N^2LF + NL^2F + NLF^2}{2} \div N \times L \times F$$

$$T = \frac{18^2(402)(7) + (18)(402)^2(7) + (18)(402)(7)^2}{2} \div (18)(402)(7)$$

$$T = 10{,}814{,}202 \div 50{,}652$$

$$T = 214$$

By TCS

This approach is improved by using the previous method. Therefore, the number of chosen test cases is used for completing a further step, which is to find the test case that has the maximum coverage value. Accordingly, we continue finding the number of selected test cases until it is equivalent to *T*.

According to the experiment, the example of finding *Cov(N), Cov(L),* and *Cov(F)* for the subject program named Print-tokens is shown as follows [9];

Cov(N)= 37
Cov(L)=33
Cov(F)=35

$T= Cov(N)+Cov(L)+Cov(F)$
$T=105$

This means that the numbers of selected numbers equal 105, which can cover all test cases in a test poll (4,130). The rest of the subject programs are experimented with by the same method. The results of finding the selected test cases of the comparative studies are shown in **Table 3**.

By HTTCS

At first, the method of finding *T* is similar to TTCS, which results in 214 picked out test cases. Later, a set of the chosen test cases is evaluated to find "pass" or "fail". According to the experiment, there are 49 test cases resulting in "fail" [9]. These are necessary to be effective selected test cases, required by using the second method of HTTCS, which is explained above.

Table 3 shows the results of finding the numbers of selected test cases by using 3 comparative studies, which are TTCS, TCS, and HTTCS. Obviously, we can see that the number of selected test cases by using HTTCS is smallest, which is relevant to the first objective proposed in this paper. This is because applying HTTCS can provide benefit by trying to use the selection methods regarding the algorithm provided in TTCS and TCS, including choosing the number of selected test cases that resulted in "fail". **Tables 3** and **4** present % size of using HTTCS, which is less than that from applying TTCS and TCS. For example, in Print-tokens, % size of HTTCS is less than TTCS and TCS by 336.73 and 114.29 %, respectively [8,9]. According to this, using HTTCS technique can reduce the complexity of testing when it concerns the number of selected test cases. **Table 5** shows the results of % reduction rate of the comparative studies compared with the size computed by Rothermel and Harrold [5,6]. TTCS can decrease the size by about 80 - 95 %; TCS shows a better reduction of 96 - 99 %. However, when comparing these results using HTTCS, the performances of the comparative studies are less than the proposed model (97 - 99 %). **Table 6** shows the results of determining the % reduction of the 3 techniques when comparing HTTCS versus TTCS and TCS, respectively [8,9]. For example, by using a subject program named Tcas, % reduction rate using HTTCS is greater than that of TTCS and TCS by 22.68 and 0.51 %, respectively. **Table 7** presents the ability of solving problems found in the set of the selected test cases. Obviously, the results shown using the comparative studies run around 99.24 - 99.82, 99.71 - 99.93, and 99.90 - 99.98 % when experimenting with TTCS, TCS, and HTTCS, respectively. **Table 8** is a summary of using HTTCS, which is greater than TTCS by about 0.07 - 0.26 regarding the 7 subject programs. Additionally, HTTCS shows higher % efficiency than TCS by 0.00 - 0.25 %. This may reflect that all % efficiency, in terms of testing programs by using the comparative studies, result in very high value. However, one of the objectives regarding using HTTCS is also reached. The results may not be significant, but to apply the proposed method can give a better performance, shown by the experimental results.

Table 3 Numbers of test cases from several methods.

Name	TTCS	TCS	HTTCS
Print-tokens	214	105	49
Print-okens2	256	60	55
Replace	285	114	65
Schedule	163	69	37
Schedule2	162	78	37
Tcas	316	31	23
Totinfo	99	40	33

Table 4 % size of HTTCS is less than the comparative studies.

Name	TTCS vs HTTCS	TCS vs HTTCS
Print-tokens	336.73	114.29
Print-okens2	365.45	9.09
Replace	338.46	75.38
Schedule	340.54	86.49
Schedule2	337.84	110.81
Tcas	1273.91	34.78
Totinfo	200.00	21.21

Table 5 % reduction rate.

Name	TTCS	TCS	HTTCS
Print-tokens	94.82	97.46	98.81
Print-okens2	93.78	98.54	98.66
Replace	94.86	97.94	98.83
Schedule	93.85	97.40	98.60
Schedule2	94.02	97.12	98.63
Tcas	80.35	98.07	98.57
Totinfo	90.59	96.20	96.86

Table 6 % reduction rate of HTTCS is greater than the comparative studies.

Name	HTTCS vs TTCS	HTTCS vs TCS
Print-tokens	4.21	1.39
Print-okens2	5.21	0.12
Replace	4.18	0.90
Schedule	5.07	1.24
Schedule2	4.91	1.56
Tcas	22.68	0.51
Totinfo	6.93	0.69

Table 7 % Problem-Solving of HTTCS is greater than the comparative studies.

Name	TTCS	TCS	HTTCS
Print-tokens	99.81	99.93	99.98
Print-okens2	99.81	99.88	99.93
Replace	99.91	99.93	99.95
Schedule	99.70	99.85	99.96
Schedule2	99.82	99.82	99.89
Tcas	99.69	99.75	99.94
Totinfo	99.24	99.71	99.90

Table 8 % efficiency of HTTCS is greater than the comparative studies.

Name	**HTTCS-TTCS**	**HTTCS-TCS**
Print-tokens	0.10	0.07
Print-okens2	0.10	0.00
Replace	0.07	0.02
Schedule	0.26	0.15
Schedule2	0.15	0.15
Tcas	0.44	0.25
Totinfo	0.19	0.00

Conclusions

Both TTCS and TCS give better results than the traditional methods, such as retest-all, random, and regression technique, do. However, HTTCS is proposed in order to improve the ability of choosing the selected test cases by hybridizing TTCS and TCS, with the rule of selection regarding testing the test cases which result in "fail". This is because the problems or bugs in a test case that can be occur should be protected. There are 4 benefits of using HTTCS shown, as follows; first, the number of selected test cases provided by applying HTTCS is smaller than that of TTCS and TCS, by about 200 - 1274 and 9 - 114 % respectively. Second, % reduction rate using HTTCS is better than TTCS and TCS, by about 4 - 23 and 0.5 - 1.6 % individually. Third, % problem-solving of HTTCS is about 99.89 - 99.98 %, which is higher than that using TTCS and TCS. Lastly, % efficiency of using HTTCS is higher than TTCS and TCS by about 0.07 - 0.44 and 0.00 - 0.25 %, respectively.

References

[1] A Abran and H Nguyemkim. Analysis of maintenance work categories tough measurement. *In*: Proceedings of the Conference on Software Maintenance. USA, 1991, p. 104-13.

[2] RS Arnold. A road map guide to software re-engineering technology. *In*: Proceedings of the Conference on Software Reengineering. USA, 1993, p. 3-22.

[3] G Alkhatib. The maintenance problem of application software: An empirical analysis. *J. Software Mainten. Res. Pract.* 1992; **4**, 83-104.

[4] M Hutchins, H Foster, T Goradia and T Ostrand. Experiments on the effectiveness of dataflow and control flow-based test adequacy criteria. *In*: Proceedings of the 16th International Conference on Software Engineering. USA. 1994, p. 191-200.

[5] G Rothermel. A safe efficient regression test selection technique. *ACM Trans. Software Eng. Meth.* 1997; **6**, 173-210.

[6] G Rothermel and MJ Harrold. Analyzing regression test selection techniques. *IEEE Trans. Software Eng.* 1996; **22**, 529-51.

[7] TL Graves, MJ Harrold, MJ Kim, A Porter and G Rothermel. An empirical study of regression test selection techniques. *ACM Trans. Software Eng. Meth.* 2001; **10**, 184-208.

[8] A Lawanna. Technique for test case selection in software maintenance. *Walailak J. Sci. & Tech.* 2014; **11**, 69-77.

[9] A Lawanna. The improvement of test case selection for the process of software maintenance. *Inform. Tech. J.* 2014; **10**, 73-81.