3-2012

Pair-Wise Time-Aware Test Case Prioritization for Regression Testing

Prem Parashar
Sheridan College, prem.parashar@sheridancollege.ca

Arvind Kalia
Himachal Pradesh University

Rajesh Bhatia
Deen Bandhu Chotu Ram University

Pair-Wise Time-Aware Test Case Prioritization for Regression Testing

Prem Parashar[1], Arvind Kalia[1], and Rajesh Bhatia[2]

[1] Computer Science Department, Himachal Pradesh University, Shimla, India
[2] Computer Science Department, Deen Bandhu Chotu Ram University, Murthal, India
{prem.parashar,arvkalia,rbhatiapatiala}@gmail.com

Abstract. After maintenance, software requires regression testing for its validation. Prioritization of test cases for regression testing is required as software is tested under strict time and other constraints. A Pair-wise time-aware Test Case Prioritization (PTCP) technique has been proposed in this paper that determines the effectiveness of a test case on the basis of total number of faults present in software, number of faults detected till time, and the time of execution of different test cases. It selects that test case which determines maximum new faults, not yet detected, within minimum time. Thus prioritized test suite contains those test cases which are effective and tend to minimize repetitive faults detection. Through two comparative studies, it has been observed that with least wastage of time, the proposed technique performed equally well as other two parallel prioritizing techniques, Average Percentage of Fault Detection (APFD) based prioritization, and Optimal Test Case Prioritization (OTCP).

Keywords: Regression testing, prioritization, fault detection, redundancy, random selection.

1 Introduction

Software testing is one of the very expensive stages of software development life cycle [22]. It is an important part of the software development irrespective of the programming paradigm used. It is a broad term containing a wide spectrum of different activities. It starts from the testing of small unit of software to the post implementation and maintenance of software. The main activities involved in software testing are preparing test cases and their respective test oracles [12]. Since both these tasks are very tedious, therefore even after preparing test cases and test oracles with utmost care, some bugs remain uncovered. Software may malfunction due to uncovered bugs. The consequences of such errors may be nominal or catastrophic depending upon the type of software application [13, 14, 15, 22]. In order to avoid such situations, the system is tested with sufficient number of test cases and the gap between expected value and actual value is observed. Any difference between these values declares the system as erroneous.

S. Dua et al. (Eds.): ICISTM 2012, CCIS 285, pp. 176–186, 2012.

Regression testing has always remained a challenge for maintenance team. As maintenance is not a regular or periodic activity for most of the software[13], it is very difficult to make prediction about software maintenance time. For some software systems like business applications, frequency of maintenance may be very high whereas in other types like scientific application, it may be low. When software is brought for maintenance , it is very difficult to reconstitute the same team which had developed and tested it. The formation of new maintenance team makes this process complex.

One of the most important constraints in regression testing is time budget [6]. With the maintenance of software, the size of test cases usually grows making it difficult to execute all the existing test cases and new test cases with in the specified time limit. Prioritization of test cases is even significant if the time budget and total time for the execution of a test suite are equal. After modification, when new changes are incorporated in existing software, software need to be tested well by using a test suite that covers change impact set (all modified components of software and the other components affected by this change). Generally, due to limited time constraints, it is impossible to execute all test cases of a test suite [11, 20, 22]. Under such circumstances, a test case prioritization technique is required which can generate an optimum subset of test cases that covers all or almost all changes. It is not an easy task to construct such subset of given test suite, especially if the number of changes made is very large. Researchers have developed different types of test case prioritization techniques. In this paper it is evident that those techniques also prove to be effective under given time constraints. Pair-wise Test Case Prioritization (PTCP) technique has been presented in this paper, which prioritizes test cases such that test cases selected from test suite (i) will always run within given time limit, (ii) will have the highest potential for fault detection, and (iii) will have the minimum wastage of time.

2 Review of Literature

Rothermel [16,17] et al. described different test case prioritization techniques and compared their relative results through empirical studies. The average percentage of fault detection (APFD) and total fault-exposing potential (TFEP) measures proposed by them have become benchmarks for making comparisons with other techniques. Dalal [3, 4] et al. proposed that automatic test cases that are generated by using combinatorial approach of pair wise interaction between the input fields are highly effective in detecting the failures as compared to traditional approach of software testing. By using this approach, number of test cases required for testing software can be reduced considerably.

Zang [23] et al. used integer linear programming (ILP) method for prioritizing test cases. They represented the test case prioritization problem as an integer linear programming of operations research constructing objective function with the help of number of faults detected by test cases and constraints by taking time of execution of test cases into consideration. The optimum solution of ILP gives the set of feasible test cases for the software and they are prioritized by implementing any of the

available test case prioritization techniques. Test cases can be prioritized on the basis of total coverage techniques such as total statement coverage, additional statement coverage, total fault expose potential (TFEP), total function coverage, and additional function coverage. The success of a test case does not only depend on whether it covers the statements that contain faults but also on its ability to surface out the possible faults [17, 19, 22]. In latest studies, it has been observed that various prioritization techniques like prioritization based on random ordering, optimum rate of fault detection, number of branches covered by a test case, and number of methods covered by a test case, are playing significant role in reducing the cost of software testing [5]. The main advantage of all these techniques is that they are general prioritization techniques and can be applied to any type of software application.

Arcuri A.,and Yao Xin [1] have proposed a memetic algorithm that tests container classes of object-oriented software. The main objective of this algorithm is to reduce the search space for object-oriented software. The objective has been achieved by dynamically eliminating those functions from search space that have already been covered. They compared their algorithm to a Hill climbing and a Genetic Algorithm through empirical study and showed that memetic algorithm outperforms the other two types of algorithms.

Corel [2] et al. proposed automatic test data generation for regression testing. The chaining method for test data generation has been implemented in the proposed approach. This test data generation has mainly focused on the common functionality of old and the new system. In this method a pair of equivalent output variables has been identified from old and new system. A software system is erroneous if for a given input its equivalent output variables produce different outputs. The empirical studies conducted by them on systems of different domain have shown that this method of regression testing is most successful in version-specific software. A similar study conducted by Xie and Notkin [21] considered value spectra to see the difference between old and new version of software. They also performed root cause analysis of the deviation by using some heuristics, the main variants of which have been deviation follower and deviation container. The study conducted has shown encouraging results to locate the deviation in the behavior of system at regression testing. Li [9] et al. considered greedy algorithm, additional greedy algorithm, 2-optimal algorithm, hill climbing algorithm, and genetic algorithm for prioritizing test cases. The results of empirical study conducted by them showed that though genetic algorithms do not produce best results all the times but they perform better as compared to greedy algorithms and optimal algorithms most of the times. Jiang [7] et al. proposed a genetic time aware test case prioritizing algorithm in which all permutations of the test cases of a test suit that fit in the given time constraints have been considered. Those test cases have been selected for final execution which exposed maximum faults within minimum time. The algorithm has also kept an eye on the redundancy of the test cases.

Software operational profile plays vital role in prioritizing test cases. While prioritizing the test cases, the operation which is most important is given the highest priority in software testing irrespective of its probability of occurrence [5, 8]. In continuous integration, the developers integrate the software artifacts with the

continuous integration agent frequently. Usually, the continuous integrated agent gets a number of modules for integration at a time due to different teams of developers. The main objective of continuous integration is that a developer gets bug report at early stage of software development and can be fixed at earliest. By conducting various empirical studies on different types of software, it has been observed that coverage-based strategies for prioritizing test cases of a test suite outperforms the other strategies in continuous integration testing [7].

Mei [10] et al. proposed a technique for prioritization of test cases for regression testing of business-oriented application. This study has taken into account the erroneous attempts performed by process engineers unknowingly while maintaining some of the artifacts of a system [18]. These faults are unexpected for a test manager and hence are not covered by traditional test suite due to its limited vision. In such cases, coverage based testing is needed. In this study, series of such prioritizing techniques have been implemented. The results of experimental study have shown that the importance of an artifact plays significant role in test case prioritization.

3 Research Methodology

One of the main objectives of regression testing is to test the software within given time budget. The PTCP technique proposed in this paper, is based on two assumption,(i) the reordering of test cases in a test suite is an acceptable ordering i.e. the test cases are independent from one another and hence can be executed in any order, and (ii) reordering of test cases does not make any change in their behavior i.e. the test case will take same time for the execution and will reveal the same faults irrespective of its position in the test suite.

Let time budget assigned to test a software for regression testing is TB, and there are *n* test cases in a test suite T with times of execution $t_1,t_2,\ldots,t_n$ respectively. Even in ideal time conditions, the prioritization of test cases is required,i.e, if the following inequality holds true, still there is need of prioritization, either for saving time or for additional testing.

$$\sum_{i=1}^{n} t_i <= TB$$

In PTCP, while selecting a test case, number of new faults detected (fd) by it, its time of execution(t), and number of repetitive faults detected(rfd) by it are taken into consideration. The effectiveness (et) of a test case has been calculated as:

$$et = \frac{fd}{t * rfd} \qquad if\ rfd \neq 0\ and$$

$$et = \frac{fd}{t} \qquad if\ rfd == 0$$

A test case that has maximum value of *et* is considered as most effective. A tie is broken arbitrarily and one of the test cases is included in the prioritized list if two or more test cases have same maximum et value. The algorithm used for the purpose is shown in Table A.

3.1 PTCP Algorithm

Input: A test suite (T), permissible time budget (TB), set of faults not yet detected (TF) and set of faults detected (ft_i) after execution of test case T_i.
Output: a) P: Prioritized list of test cases. b) FD: set of faults detected

In Table A, different mathematical set operations i.e. A U B, A ∩ B, |A|, and A-B have been considered to make proposed algorithm simple and understandable. Test suite T stores test case number. For test case T_1, it stores 1, for T_2, it stores 2, and so on.

After implementing the algorithm given in Table A, it has been observed that number of random selections of test cases in PTCP is always lesser than or equal to APFD based and OTCP techniques. *et*, computed in this algorithm, stores the effective potential of a test case to detect the new faults.

Table A. PTCP Algorithm

```
1.  max=0;
2. for i=1 to n
3. {if ((|ft_i |/t_i  ≥  max) and (t_i ≤ TB)
4. {i) max=| ft_i |/ t_i;   ii)    p=i;}
5. }
6. i) t=t_p;          ii) FD=FD U ft_p;
7. iii)TF=TF-ft_p;      iv)P=P U T_p;
8.  v) T_p=0;
9.while ((TF ≠ Φ) and ((t ≤ TB))
10.{      max=0;
11.       for i=1 to n
12.       {if ((T_i ≠ 0) and ( t+ t_i  ≤ TB))
13.          {fd=|TF∩ ft_i | ; //pairing( for new faults)
14.               if (fd>0)
15.             {rfd=|FD∩ ft_i|     //pairing(redundancy)
16.                 if (rfd≠0)
17.                     et=fd/(t_i*rfd);
18.                else
19.                     et=fd/t_i;
20.              }
21.             if (et>max)
22.             {
23.                   i)  p=i;   ii)  max=et;
24.             }
25.       }
26.    }
27. t=t+t_p;            ii)      FD=FD U ft_p;
28. iii) TF=TF-ft_p;    iv)      P=P U T_p;
29. vi)  T_p=0;
30.}
```

4 Objectives

The broad objective of the comparative study conducted in this section is to identify those factors which are associated with time-aware test case prioritization. The specific objectives of the comparative study are :

1. To analyze which technique makes optimum utilization of permissible time budget for regression testing.
2. To Analyze which technique minimizes the overlapping of faults detected.

5 Major Findings

The results of the comparative study are based on two examples of test suites tabulated in Table 1 and Table 4 respectively. It has been supported from the results of study that PTCP performs equally well as other two techniques (APFD,OTCP) with least wastage of time. In proposed study,the time taken to detect repetitive fault(s) has been considered a wastage, which should be minimized in regression testing if not avoided. This time can be utilized to test some other modules (which are not included in change impact set) which will boost the confidence in software testing. It has also been found that PTCP technique has strong tendency to minimize repetitive fault detection. The evaluation metrics considered in the study have played significant role in comparative study.

5.1 Analysis

A comparative study of proposed technique has been conducted with two parallel techniques APFD and OTCP proposed by different researchers. The comparative study has been conducted by taking two examples.In Example 1, a test suite(Table 1) with six test cases(T_1,T_2,…,T_6) and ten known faults (f_1,f_2,…,f_{10}), has been taken. Column T of Table 1 contains the respective times of execution(in second) of test cases. Let time limit allowed for regression testing be 20 seconds. The evaluation metrics proposed in this study are:

(i) Total Faults Exposed(TFE): It represents the total number of faults exposed by prioritized test suite. It also includes the repetitive faults detected by the test suite, if any. The value of TFE may be greater than the total number of distinct faults present in the software.

(ii) Distinct Faults Detected(DFD): It represents the distinct faults detected by prioritized test suite. If T_1,T_2 are two test cases that constitute the prioritized test suite and detect faults f_1, f_2, f_3, f_5 and f_2,f_5, f_6 respectively, the DFD value for the test suite will be 5. DFD ignores the repetitive faults detected by prioritized test suite.

(iii) Effective Time for execution of test cases(ET): It represents the total time taken to execute all test cases of prioritized test suite.

(iv) Percentage of Test Suite Failure (TSF): It represents the percentage of faults not detected by the prioritized test suite. for example. if the total number of faults present is 10 and the prioritized test suite detects 9 faults , the value of TSE will be 10.

(v) Percentage of Time Wastage (TW): It represents the percentage of total time wasted in detecting repetitive faults. If repetitive faults take *m* units of time and total time allowed is *n* units, then TW will be calculated as *(m/n*100)*.

(vi) Percentage of Time Saving (TS): The detection of all faults before given time limit has been considered as time saving. It is calculated as (m/n*100), where m represents the units of time saved and n represents time budget.

Table 1. Test suite and list of faults exposed (Example 1)

'X' represents fault detected

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	T
T_1	X	X	X				X	X			6
T_2		X	X				X		X		6
T_3	X			X	X					X	5
T_4						X					3
T_5		X			X		X		X		6
T_6	X		X		X			X		X	6

First row of Table 1 represents that test case T_1 exposes 5 faults(f_1,f_2,f_3, f_7 and f_8) in 6 seconds. The remaining rows of Table 1 can be interpreted in similar way.

After implementing the PTCP algorithm (Example 1), the results obtained are tabulated in Table 2.

Table 2. Prioritized set of test cases(Example 1)

Time Limit: 20 sec.	
APFD	T_1,T_6,T_3,T_4
OTCP	T_1,T_3,T_2,T_4
PTCP	T_1,T_3,T_4,T_5

In Table 2, for APFD, the test cases selected are T_1,T_3,T_4,T_6 and their order of execution is T_1, T_6, T_3, T_4. The results for remaining techniques can be interpreted in similar way.

In Table 3, the results of evaluation metrics for different techniques are tabulated.

Table 3. Metrics values analysis (Example 1)

	TFE	**DFD**	**TSF (%)**	**ET**	**TW(%)**	**TS(%)**
APFD	15	9	10	20	40	0
OTCP	14	10	0	20	28.6	0
PTCP	14	10	0	20	28.6	0

In Table 3, for APFD, total number of faults exposed is 15 (5+5+4+1),distinct number of faults detected is 9 (f_1,f_2,f_3,f_4,f_5,f_6,f_7,f_8,f_{10}), effective time is 20 seconds (6+6+5+3), test suite failure is 10% (f_9 is undetected), time wastage is 40% (6 faults are detected more than once in 20 seconds), and there is no time saving. The results for other techniques tabulated in Table 3 can be interpreted in similar manner.

In Example 2, a less organized test suite has been taken into consideration. The test suite contains test cases T_1, and T_3, that reveal the same faults and total faults exposed by all its test cases are approximately three times the number of distinct faults present. Therefore, there is a strong possibility of overlapping in fault detection. The time allowed in this case has been considered as 12 seconds. The example is tabulated in Table 4.

Table 4. Test suite and list of faults exposed (Example 2)

'X' represents fault detected

	f_1	f_2	f_3	f_4	f_5	f_6	T
T_1	X		X			X	5
T_2		X		X	X		7
T_3	X		X			X	6
T_4		X			X		3
T_5				X	X		4
T_6	X	X	X		X	X	5

The prioritized test suites generated by applying different techniques have been tabulated in Table 5 and can be interpreted in similar way as for Table 2.

Table 5. Prioritized set of test cases (Example 2)

Time Limit: 12 sec.	
APFD	T_6,T_4,T_5
OTCP	T_6,T_2
PTCP	T_6,T_5

The evolution metrics results for Example 2 have been tabulated in Table 6.

Analysis of both examples revealed that PTCP technique performs equally well as other two techniques, APFD and OTCP with least number of repeated faults.

Table 6. Metrics values analysis for different techniques (Example 2).

	TFE	DFD	TSF (%)	ET	TW(%)	TS(%)
APFD	9	6	0	12	33	0
OTCP	8	6	0	12	25	0
PTCP	6	6	0	9	0	25

5.2 Results and Discussion

The results of Example 1 and Example 2 are shown graphically in Figure 1 and Figure 2 respectively.

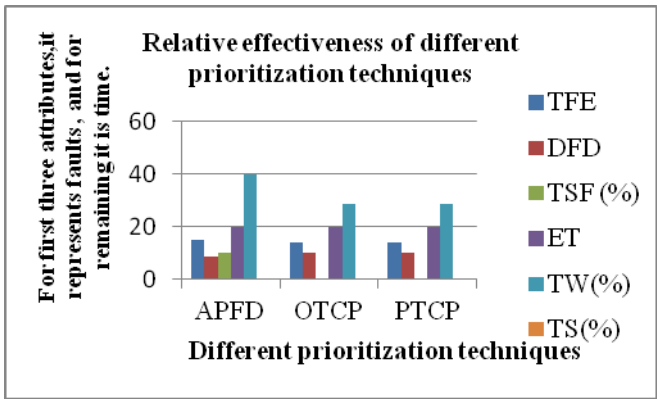

Fig. 1. Relative effectiveness of various techniques (Example 1)

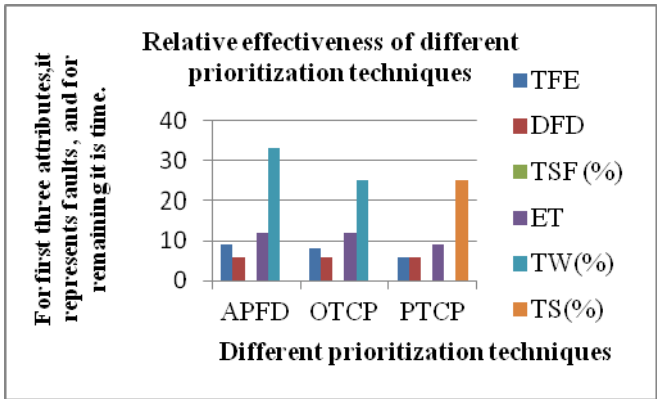

Fig. 2. Relative effectiveness of various techniques (Example 2)

From the results of two comparative studies, it is evident that by taking time and number of effective faults detected by a test case together into consideration, time budget can be managed efficiently. After comparing the results of Figure 2 and Figure 3, it has been observed that PTCP technique is very effective if the test suite is not properly organized. Though results of OTCP are very close to that of PTCP but still PTCP technique is better than OTCP as the number of random selections of test cases required in former is always less than or equal to the later. In case of OTCP, an optimal selection of set of test cases requires the knowledge of all permutations of test cases. It becomes complex if the size of test suite is very large. In such situation, the

proposed technique will certainly help the maintenance team to perform regression testing efficiently. Since time effectiveness and reduction in redundancy of faults detection are given highest priority while forming prioritized set of test cases, the proposed technique outperforms under tight time constraints.

5.3 Threats to Validity

Internal Validity. One of the main threats to internal validity is that this study is based upon two main assumptions (i) the test cases of a test suite are independent from each other and (ii) their positions in the test suite do not make any difference in their behaviors. The assumption in all the cases may not be true. The correlation to find the effectiveness of a test case requires further verifications and validations.

External Validity. Threats to external validity are that the study is based on the results of two examples that contain test cases with known faults. In real practice it is difficult to predict test cases and their respective associated faults. When the proposed technique is implemented on subject examples, the results are satisfactory, but in order to generalize it, more experimental studies are needed.

Construct Validity. The evaluation metrics considered in this study may be threats to its construct validity. The mentioned evaluation metrics have appeared in very few researches. Though, through the comparative study, it has been concluded that these metrics play vital role in prioritization, still their relevance to the objectives requires more exploration.

6 Conclusion and Future Work

The proposed technique for the prioritization of test cases has shown encouraging results in maximizing fault detection and minimizing wastage of time as compared to other techniques (APFD and OTCP). The proposed algorithm reduces the possibility of random selection of test cases for execution. PTCP technique can be extended further by conducting empirical study on different software. Although, the metrics proposed in this paper are very effective, still, their generalization needs exhaustive studies of different software domains. The present study can be extended further by considering the behavior of faults instead of test cases.

References

1. Arcuri, A., Xin, Y.: A memetic algorithm for test data generation of object oriented software. In: IEEE Congress on Evolutionary Computation (CEC) (2007)
2. Corel, B., Al-Yami, A.M.: Automated Regression test Generation. In: ISSTA (1998)
3. Dalal, S.R., Jain, A., Karunanithi, N., Leaton, J.M., Lott, C.M., Horowitz, B.M.: Model-Based Software Testing of highly programmable system. In: ISSRE 1998 (1998)
4. Dalal, S.R., Jain, A., Karunanithi, N., Leaton, J.M., Lott, C.M., Horowitz, B.M.: Model-Based Testing in Practice. In: ICSE 1999. ACM Press (1999)

5. Sebastian, E., Malishevsky, A.G., Gregg, R.: Test case prioritization: A family of empirical studies. IEEE Transactions on Software Engineering 28(2) (2002)
6. Graves, T.L., Harrold, M.J., Kim, J.M., Porter, A., Rothermel, G.: An empirical study of regression test selection techniques. ACM Trans. On Software Engineering (2001)
7. Jiang, B., Zhang, Z., Tse, T.H., Chen, T.Y.: How well do test case prioritization techniques support statistical fault localization. In: ICSAC 2009 (2009)
8. Kumar, K.S., Babu, M.R.: Software Operational Profile Based test Case Allocation Using Fuzzy Logic. International Journal of Automation and Computing 04(4), 388–395 (2007)
9. Li, Z., Mark, H., Hierons, R.M.: Search algorithms for regression test case prioritization. IEEE Transaction on Software Engineering 33(4) (2007)
10. Mei, L., Zhang, Z., Chan, W.K., Tse, T.H.: Test case prioritization for regression testing of service –oriented business applications. In: WWW 2009 (2009)
11. Memon, A., Banerjee, I., Hashmi, N., Nagarajan, A.: DART: A framework for regression testing "nightly/daily builds" of GUI applications. In: ICSM (2003)
12. Bertrand, M., Ilinca, C., Andreas, L., Lisa, L.: Automatic testing of object-oriented Software. In: SOFSEM (2007)
13. Parashar, P., Bhatia, R., Kalia, A.: Change Impact Analysis: A Tool for Effective Regression Testing. In: Dua, S., Sahni, S., Goyal, D.P. (eds.) ICISTM 2011. CCIS, vol. 141, pp. 160–169. Springer, Heidelberg (2011)
14. Parashar, P., Kalia, A., Bhatia, R.: Fault-based time-aware test case prioritization for regression testing. In: The Proc. Indian Science Congress 2011, pp. 94–103 (2010)
15. Philippe, G., Segla, K., Filippo, R., Giuliano, A.: Evolution and Search Based Metrics to Improve Defects Prediction, pp. 23–32. IEEE Society (2009)
16. Gregg, R., Untch, R.H., Chu, C., Jean, H.M.: Prioritizing test cases for regression testing. IEEE Transactions on Software Engineering (2001)
17. Gregg, R., Untch, R.H., Chu, C., Jean, H.M.: Test case prioritization: An Empirical study. In: ICSE 1999 (1999)
18. Mark, S., Mike, L., Laurie, W.: Prioritization of regression tests using singular value decomposition with empirical change records. In: ISSRE (2007)
19. Voas, J.: PIE: A dynamic failure-based technique. IEEE Transaction on Software Engineering, 717–727 (1992)
20. Walcott, K.R., Kapfhammer, G.M., Soffa, M.L., Roos, R.S.: Time-aware test suite prioritization. In: ISSTA 2006 (2006)
21. Xie, T., David, N.: Checking Inside the Black Box: regression testing by Comparing Value Spectra. IEEE Transactions on Software Engineering 31(10) (2005)
22. Yoo, S., Harman, M.: Regression Testing Minimization, Selection, and Prioritization: A Survey. Softw. Test. Verif. Reliab. (2007)
23. Zang, L., Hou, S.S., Guo, C., Xie, T., Mei, H.: Time-Aware Test –Case Prioritization using Integer Linear Programming. In: ISSTA 2009 (2009)