Front. Comput. Sci., 2016, 10(5): 769–777
DOI 10.1007/s11704-016-6112-3

REVIEW ARTICLE

Test-case prioritization: achievements and challenges

Dan HAO (✉)[1,2], Lu ZHANG[1,2], Hong MEI[1,2]

1 Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing 100871, China

2 Institute of Software, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China

Abstract Test-case prioritization, proposed at the end of last century, aims to schedule the execution order of test cases so as to improve test effectiveness. In the past years, test-case prioritization has gained much attention, and has significant achievements in five aspects: prioritization algorithms, coverage criteria, measurement, practical concerns involved, and application scenarios. In this article, we will first review the achievements of test-case prioritization from these five aspects and then give our perspectives on its challenges.

Keywords test-case prioritization, achievements, challenges

1 Introduction

Test-case prioritization is firstly proposed by Wong et al. [1] in 1997, to deal with the trade-off between what ideal regression testing should do and what is affordable by scheduling the execution order of test cases. Later, in 1999 Rothermel et al. [2] presented an industrial case (i.e., a product containing about 20 000 lines of code consumes seven weeks on running its entire test suite) to show the necessity of test-case prioritization and further proposed nine techniques, which usually serve as the control techniques in the evaluation of novel test-case prioritization techniques. Since then, a large number of work contributes to this domain.

Test-case prioritization is initially proposed in the scenario of regression testing, and thus mostly used to improve the effectiveness of regression testing. In regression testing, the test cases designed for an old version are usually reused to test its latter versions to reduce the cost on test case generation. Furthermore, to reveal faults in the latter versions as early as possible, the reused test cases should be executed in some specified order, which is the aim of test-case prioritization. That is, regression test-case prioritization (usually abbreviated as RTP) targets at scheduling the execution order of test cases designed for an early version so as to detect faults in a latter version as early as possible.

Besides the scenario of regression testing, test-case prioritization is also applied to other testing scenarios where test cases are not designed for an old version but for the current version, which is called initial testing in this paper. That is, test-case prioritization in initial testing (abbreviated as ITP in this paper) targets at scheduling the execution order of test cases designed for the current version so as to detect faults in the current version as early as possible. Due to the characteristics of ITP, its techniques are usually applicable to regression testing, whereas the techniques of the latter may not be applicable for initial testing.

Given any test suite T, test-case prioritization is formalized [3] to find a permutation T' of T satisfying $f(T') \geqslant f(PT)$, where PT represents any permutation of T and f is a function defined to map permutations of T to real numbers. Considering the ultimate goal of test-case prioritization, the number of detected bugs in any given time tends to be regarded as the goal, but it is not due to its infeasibility. Therefore, many alternative goals like structural coverage are used to guide test-case prioritization instead. The difference be-

Received February 24, 2016; accepted April 12, 2016

E-mail: haodan@pku.edu.cn

tween these alternative goals and the ultimate goal weakens the effectiveness of test-case prioritization, which becomes an inherent problem in existing test-case prioritization techniques. Besides, even taking these alternative goals, test-case prioritization is also an NP-hard problem [4]. Therefore, test-case prioritization also suffers from the efficiency issue. Furthermore, it is also challenging to measure the effectiveness of a test-case prioritization technique, which is actually related to the function f.

Similar to previous work [5], we also classify the existing work on test-case prioritization into five categories: prioritization algorithms, coverage criteria, measurement, practical issues involved, and application scenarios. In the remaining of this paper, we will first review the achievements of test-case prioritization in these categories in Section 2, and then present our perspectives on the challenges of test-case prioritization in Section 3.

2 Achievements

Test-case prioritization is proposed as a process to identify a permutation of test cases so that test cases executed in such an order may detect faults early. However, whether a test case detects faults is unknown until the test case is executed. That is, such information cannot be actually used to guide test-case prioritization. Therefore, many alternative/intermediate goals like structural coverage (e.g., statement coverage, branch coverage) are studied and used to guide test-case prioritization. Section 2.1 summarizes the achievements of test-case prioritization falling in this group. However, as none of these intermediate goals are equivalent to the ultimate goal of test-case prioritization, test-case prioritization approaches based on such intermediate goals can hardly produce the best prioritization result.

Taking any of these intermediate goals to define the function f, test-case prioritization is still an NP-hard problem. That is, it is costly to produce the best prioritization result in terms of an intermediate goal. Therefore, many heuristics or strategies have been proposed in the literature so as to produce better prioritization results. Section 2.2 summarizes the achievements of test-case prioritization falling in this group. Interestingly, due to the difference between ultimate goals and intermediate goals, some heuristic test-case prioritization algorithms on intermediate goals sometimes produce good prioritization results, which are even good enough in terms of some ultimate goals like fault-detection [6].

Besides, it is also challenging to measure the effectiveness of test-case prioritization. Test-case prioritization was initially proposed to address the effectiveness issue of software testing, especially when testing resources (e.g., time and human-efforts) are limited. That is, in practice, some of the scheduled test cases may be not executed at all due to the limited testing resources. Therefore, the results of test-case prioritization should be measured by considering such a constraint. Section 2.3 presents the typical measurements used in test-case prioritization.

Furthermore, test-case prioritization is a practical problem, and thus many practical issues should be considered in the work of test-case prioritization. Section 2.4 summarizes the practical issues in test-case prioritization and presents how existing work addresses these practical issues.

Although test-case prioritization is first proposed in regression testing, it can also be applied in other testing scenarios. Section 2.5 summaries the achievements of test-case prioritization considering its application scenarios.

In general, we classify the work on test-case prioritization into five categories similar to previous researchers did [5,7,8], shown by Table 1 [2,4,5,9–32]. We will present the typical work within each category in the following five subsections.

Table 1 Classification on test-case prioritization

Category	Typical work
Coverage criterion	code-based coverage [9–11], model-based coverage [12–14], fault-related coverage [2,15,16], input alone coverage [17] ...
Algorithm	greedy algorithms [2,4], search-based algorithm [4,17,18], ILP based algorithms [5,19] ...
Measurement	$APFD$ [2], $APFD_C$ [20], $APXC$ [4], normalized $APFD$ [21] ...
Constraint	time budget [5,22–25], fault severity [26,27] ...
Scenario	general regression [2,15], specific regression [12,13,28–30], initial testing [11,17,31,32] ...

2.1 Coverage criteria

In principle, test-case prioritization can use any test adequacy criterion as the underlying coverage criterion. In fact, the coverage criteria used by most prioritization approaches are basic code-based coverage criteria, such as statement coverage, branch coverage [2], function/method coverage [33], block coverage [34], and modified condition/decision coverage [35]. These coverage criteria actually serve as the intermediate goal of test-case prioritization (i.e., the function f), because the fault-detection capability of test cases (i.e., the ultimate goal of test-case prioritization) is unknown before test-case execution and can hardly serve as a goal to guide test-case prioritization.

Furthermore, researchers extend these basic code-based coverage criteria and thus propose a series of extended code-

based coverage criteria, such as definition-use association coverage, ordered sequence of program entities [36], and dataflow testing criteria [37]. Besides the coverage at the implementation level, the coverage at the model level is also studied in the literature of test-case prioritization. In particular, Korel et al. [12] presented a prioritization approach based on the state-based model for the system under test, which assumes the modification in software evolution occurs on both the system and the model and prioritizes test cases based on the execution information of the model. Later, Korel et al. [13] further extended their approach but the extended approach is also based on model-based coverage criteria. Similarly, Xu and Ding [14] proposed transition coverage and round-trip coverage based on state models to prioritize aspect tests.

The preceding coverage criteria are related to the ultimate goal of test-case prioritization, fault detection, but these criteria can hardly replace the latter due to their inherent difference. In particular, these code-based coverage criteria measure the fault-detection capability of test cases to some extent according to the PIE theory [38]. Therefore, many other fault-related coverage criteria are proposed to simulate fault-detection capability of test cases. In particular, in the early work of test-case prioritization, Rothermel et al. [2] proposed to prioritize test cases based on their fault-exposing-potential, which is defined as the accumulative mutation score of mutants [39] of a test case on every statement. Similarly, Elbaum et al. [15] further proposed the fault index, which is calculated by analyzing a set of measurable attributes obtained from the source code, and combined the fault-exposing-potential with the fault index. Ma and Zhao [16] combined fault-prone (which is obtained through program analysis) and importance of module (e.g., method) as the guide of test-case prioritization. Besides, some researchers [40–42] proposed fault-based logic coverage, which is defined as the ratio of detected faults to hypothesized faults, and investigated types of logic faults and fault classes in such coverage.

Recently, Jiang and Chan [17] proposed to schedule the execution order of test cases based on the difference between test inputs. In other words, the coverage criterion used by this technique is actually test inputs themselves. Although test inputs are not a new test adequacy criterion, which has already been used in test generation, it is still novel in test-case prioritization. Moreover, different from the other coverage criteria, test inputs require no extra cost on coverage information collection at all.

2.2 Prioritization algorithms

The prioritization algorithms in this section refer to what algorithms are used in test-case prioritization. Besides random selection, greedy strategies are the earliest prioritization algorithms in the literature of test-case prioritization. In particular, the first prioritization technique [1] schedules the execution order of test cases based on their additional coverage per unit cost, the following techniques proposed by Rothermel et al. [2] schedule the execution order by total coverage or additional coverage (e.g., structural coverage, fault-exposing-potential), which are actually the widely-used total strategy and additional strategy. Using statement coverage criterion, the total strategy is to prioritize test cases based on their total number of covered statements, whereas the additional strategy is to prioritize test cases based on the number of statements covered by each test case and not covered by existing selected test cases. Similarly, Li et al. [4] proposed 2-optimal algorithm, which selects the next two test cases together, which have the largest coverage. As all these techniques belong to greedy algorithms, they cannot guarantee to produce the best prioritization results.

As test-case prioritization is an NP-hard problem, Li et al. [4] investigated to apply metaheuristic search techniques [43] to test-case prioritization so as to improve the prioritization results “at a reasonable computation cost”. These researchers applied the hill climbing algorithm and the genetic algorithm to test-case prioritization. In particular, these search-based algorithms encode the execution order of test cases and try to find the optimal solution through specified fitness function. Different from the greedy strategy, these search-based algorithms explore the solution space in a specified way guided by the fitness function and thus are expected to be effective. However, these search-based algorithms are inferior without significant difference to the additional strategy in terms of APXC, which measures the average percentage of structural coverage of the prioritized test suite. Besides, Saha et al. [44] presented a new approach REPiR which transfers the problem of test-case prioritization to a standard information retrieval problem. Tonella et al. [45] presented a machine learning based approach that incorporates user knowledge into test-case prioritization. Arafeen and Do [46] presented to cluster test cases based on requirements to enhance existing code-based prioritization techniques.

To learn the optimal prioritization result, Hao et al. [19] modeled test-case prioritization using an Integer Linear Programming model so as to generate the optimal prioritization results in terms of APXC. However, the prioritized test suite produced by such an “optimal” approach is usually less effective than the additional strategy in terms of fault-detection capability. That is, due to the difference between intermediate

goals (like statement coverage) and ultimate goals (like fault-detection capability), "it may not be worthwhile to pursue the optimality in test-case prioritization" [19].

Besides these novel prioritization algorithms, researchers in this domain may also use random prioritization, optimal prioritization, original prioritization, and reverse prioritization in comparison. In particular, random prioritization is to randomly order test cases; optimal prioritization[1] is to prioritize test cases based on their actual revealed faults; original prioritization is to keep the original order of test cases in a test suite; and reverse prioritization is to reverse the original order of test cases in a test suite. These algorithms serve as the comparison base for a novel test-case prioritization algorithm. Moreover, optimal prioritization is actually not a practical technique, because the information whether a test case reveals a fault is unknown before executing the test case itself.

2.3 Measurement

To learn whether a test-case prioritization technique is better, it is necessary to present a measurement on test-case prioritization. Therefore, some work focuses on measuring the effectiveness of test-case prioritization. This problem is first solved by Rothermel et al. [2], which presented a measurement called weighted Average of the Percentage of Faults Detected (abbreviated as APFD). In particular, APFD measures how quickly a prioritized test suite by drawing a curve for the prioritized test suite on 2-dimension space and calculating the area under such a curve. Formally, the APFD can be defined by Eq. (1)[2], where m represents the number of faults detected by the given test suite, n represents the total number of test cases in the given test suite, and TF_j represents the first test case in the prioritized test suite that detects fault j. Such a measurement is widely used in assessing the effectiveness of test-case prioritization.

$$\mathrm{APFD} = 1 - \frac{\sum_{j=1}^{m} TF_j}{nm} + \frac{1}{2n}. \tag{1}$$

As this measurement ignores the influence of test costs and fault severity, Elbaum [20] further proposed a new metric to measure the effectiveness of test-case prioritization considering their influence. In particular, the proposed cost-cognizant weighted average percentage of faults detected (usually abbreviated as $\mathrm{APFD_C}$) is adopted based on APFD, shown by Eq. (2), where $t_1, t_2, \ldots, t_n$ represent the test cases to be prioritized and $f_1, f_2, \ldots, f_m$ represent the severity of each fault detected by the given test cases.

$$\mathrm{APFD_C} = \frac{\sum_{j=1}^{m} (f_i \times (\sum_{i=TF_j}^{n} t_i - \frac{1}{2} * t_{TF_j}))}{\sum_{j=1}^{n} t_j \times \sum_{j=1}^{m} f_j}. \tag{2}$$

Besides these two measurements, researchers [4,19] latter proposed to use the average percentage of some structural coverage (abbreviated as APXC) as a measurement because the value of such a measurement is available before test-case execution. Moreover, the formula for APXC is the same as Eq. (1), whose only difference lies in the definition of m and TF_j. In APXC, m represents the total number of structural units (e.g., statements or methods) covered by the given test suite and TF_j represents the first test case in the prioritized test suite that covers the structural unit j. Based on the general definition of APXC, we may have APSC for statement coverage, APMC for method coverage, APBC for block coverage. As these measurements are defined based on intermediate goals, they are not widely used as APFD and $\mathrm{APFD_C}$.

On the other hand, some researchers modified APFD by considering some practical constraints in test-case prioritization. Qu [21] presented a normalized APFD to measure the effectiveness of test-case prioritization by addressing the two problems (i.e., various m and n in practice) in the existing APFD. This measurement solves the similar problem raised by Walcott et al. [22], which addressed this problem by assigning a penalty to the missing faults (i.e., various m). Furthermore, Do and Rothermel [47,48] presented more comprehensive cost-benefits models to assess regression testing techniques, including test-case prioritization.

2.4 Constraints

As test-case prioritization is a practical problem, some researchers focus on investigating its practical constraints. The first practical constraint raised in test-case prioritization is time budget. Ideally, all the test cases to prioritized are expected to be executed so as to guarantee the software quality. However, sometimes the time allowed to run test cases may be insufficient (e.g., only test in night) so that some of the test cases prioritized may not be executed. That is, test-case prioritization in practice may suffer from the time constraint. To solve this practical problem, Walcott et al. [22] proposed the first prioritization approach considering a testing time budget, which specifies the total execution time of test cases, and they further solved such a time-aware test-case prioritization problem through genetic algorithms. Similar, Alspaugh et

[1] The optimal prioritization technique, firstly proposed by Rothermel et al. [2], actually tries to optimize the prioritization results, but does not guarantee to produce the optimal results, because some faults may be revealed by multiple test cases

[2] Another alternative formula for APFD [8] is equivalent to this formula

al. [23], Zhang et al. [5], Bharti Suri and Shweta Singhal [24] addressed the same time constraints through knapsack solvers, integer linear programming, and ant colony optimization, respectively. In particular, these approaches take the execution time of each test case as input and solve time-aware test-case prioritization by specifying the time budget, which actually serves as a constraint. Besides these approaches, the existing additional and total algorithms are also adopted to time-aware test-case prioritization by specifying the time budget [5]. Moreover, Do et al. [25] studied the impact of the time constraints on existing test-case prioritization techniques rather than time-aware techniques, and demonstrated that the time constraints actually significantly influenced the cost-effectiveness of these techniques.

Besides time constraints, another widely studied constraints in test-case prioritization are fault severity. In particular, Park et al. [26] presented a cost-cognizant approach by incorporating the time cost and fault severity of test cases, which are obtained through the history of these test cases. Kim and Porter [49] presented another history-based approach to address the constraints resulting from severe time and resource constraints. Huang et al. [50] addressed the same test cost [27] and fault severity constraints by presenting a modified cost-cognizant approach. Furthermore, Hou et al. [51] addressed the request quotas of web service (e.g., the maximum number of requests a user can send in any given time range) in service-centric system.

2.5 Application scenarios

Although test-case prioritization is first proposed in regression testing, it may be also applied to other testing scenarios.

In regression testing, test cases for an early version tend to be reused to test a latter version so that a large number of test cases may aggregate. In such a scenario, the execution information of test cases on an early version may be reused to help determine the execution order of test cases for a latter version. However, due to the difference between the early version and the latter version, regression test-case prioritization techniques may be classified into general techniques and version-specific techniques [33]. Some techniques ignore such difference and utilize only the information of the early version, which are called general test-case prioritization, because the prioritized test cases are applicable to all the latter versions rather than some specific version. Most of the existing prioritization techniques belong to this category. On the other hand, some techniques utilize the information of the latter version, especially the change between the two versions (i.e., the early version and the latter version) [34], which are called version-specific prioritization, because the prioritized test cases are applicable to some specific subsequent version. For example, Srivastava and Thiagarajan [28] presented a binary code based approach which prioritizes test cases based on the change at the basic block granularity. Korel et al. [12,13] presented a model-based technique, which prioritizes test cases based on the system model and modification between versions. Lou et al. [30] presented a mutation based approach for software evolution, which simulates faults occurred in software evolution by mutants on the change and prioritizes test cases based on their killing information on these simulation faults. Furthermore, some researchers [15,33] also investigated the use of general technique for version-specific scenarios.

In regression testing, the extra execution information of test cases on an early version can be used to aid prioritize test cases on latter versions. However, such information is not available for many scenarios. For example, the first version of a software has no early versions at all. To distinguish from the regression testing scenario, we call the testing scenario without any execution information of the early version as initial testing. In initial testing, it is hard to apply most of the proceeding prioritization techniques due to the lack of coverage information of test cases. However, the input-based techniques [17,31] and the static analysis based techniques [11,32] are applicable because they rely on only the test cases themselves. Note that although test-case prioritization techniques specific to regression testing cannot be applied to initial testing, the latter techniques are applicable to the former scenario.

3 Challenges

In this section, we point out the challenges in the current research of test-case prioritization from several aspects.

3.1 Well-designed techniques

Test-case prioritization has been studied for long, and a large number of prioritization techniques have been proposed and investigated in the literature. However, most of the prioritization techniques are less effective than the simple greedy algorithm, the additional algorithm, resulting from the difference between the ultimate goal and the intermediate goal of test-case prioritization. In particular, as the ultimate goal of test-case prioritization can hardly serve to guide prioritization, existing prioritization techniques actually use an intermedi-

ate goal instead, and thus these "well-designed" prioritization techniques do not optimize the execution order of test cases in terms of the ultimate goal. In recent years, researchers in test-case prioritization started to notice this fact [4] and investigated this fact [19]. Unfortunately, no work in the literature actually solves this problem, and it becomes a fundamental challenge for test-case prioritization. In the future, researchers can investigate other intermediate goals (e.g., detection of mutation faults or detection of similar real faults), which have closer relationship with the ultimate goal rather than the existing intermediate goals (e.g., structural coverage).

3.2 Measurement

Most of the existing test-case prioritization techniques are evaluated in terms of their effectiveness by using APFD. However, in practice, this measurement suffers from many practical constraints. For example, different test cases may have various execution time and the faults detected by test cases may have various severity. $APFD_C$ is proposed to alleviate only two practical constraints. Therefore, another general measurement considering more than two practical constraints is needed.

Besides the effectiveness, efficiency is also another important issue influencing the usage of test-case prioritization techniques. In the past, the efficiency of test-case prioritization is mostly evaluated through complexity analysis rather than the actual prioritization time. Complexity analysis may aid to estimate the actual time cost of a prioritization algorithm, but the complexity of some prioritization algorithms (e.g., genetic algorithm [4]) is hard to estimate. Furthermore, although the time complexity of some algorithms (e.g., integer linear programming based algorithm [5]) is large, their actual prioritization time may be acceptable for some specific programs. On the other hand, the efficiency of test-case prioritization is also related to the execution time of test cases. That is, considering the aim of test-case prioritization, it is unbearable when the prioritization time is close to the time spent on test case execution. Therefore, it is necessary to study the efficiency of test-case prioritization techniques considering their actual prioritization time and test-case execution time.

Furthermore, most prioritization techniques require extra information besides test cases (e.g., structural coverage) for test-case prioritization. Apparently, obtaining such information may occur extra cost. In particular, some prioritization techniques require structural coverage on some early version [2], some techniques require static coverage of the current version [11], and some techniques require the mutation execution information on some early version [30]. Although such information can be collected before prioritization, it increases the cost of test-case prioritization. Furthermore, such information is available in some testing scenarios (e.g., regression testing), not all testing scenarios. As extra information collection is another dimension of cost, it is also challenging to measure a prioritization technique combining this dimension with other dimensions like prioritization time.

3.3 Thoughtful evaluation

In the literature, existing empirical studies investigated the various factors (e.g., programming languages [52], coverage granularity and type [11,52], fault type [39,53], test granularity [11,30], constraints [54]) that may influence the effectiveness and efficiency of test-case prioritization.

Besides, there are other factors that may influence the effectiveness and efficiency of test-case prioritization. Some of these factors are recognized as threats in the past, e.g., representative of subjects and test cases, and measurement. In particular, the early work of test-case prioritization, especially the papers published around 2 000, mostly used the seven small programs (whose number of lines of code is smaller than 600) in Siemens as the subjects in evaluation. Such threats are reduced to some extent by later work on test-case prioritization through more larger projects, e.g., *grep* and *gzip* whose number of lines of code is about 10 000.

Besides these well-recognized threats, there are also some other factors that influence the effectiveness and efficiency of test-case prioritization. For example, recently Lu et al. [55] identified another one important flawed setting in the existing evaluation, evolution of source code and test cases. That is, previous work on test-case prioritization is usually evaluated based on the source code and test cases without any change, which is not practical at all. Lu et al. [55] investigated the influence of this factor on the effectiveness of many existing prioritization techniques, and found that changes on source code do not have much influence on the effectiveness of test-case prioritization, but changes on test cases do have.

3.4 Practical concerns

Test-case prioritization is a practical problem raised from industry, and thus it is important to study test-case prioritization in the practical scenarios. Unfortunately, several practical concerns occur in the existing work of test-case prioritization.

The first practical concern of test-case prioritization lies in its application scenario. In general, test-case prioritization aims to facilitate fault detection in software testing, and thus it brings more benefit when the time spent on test-case execution is not ignorable (e.g., several days or months). In other words, when the total execution time of all test cases is small (e.g., several minutes), it does not matter so much whether a fault is detected by the first test case or the last test case. However, to our knowledge, most of the existing work is actually evaluated on the subjects whose total execution time of test cases is not large at all. That is, the existing techniques are not evaluated in its most possible application scenario.

Besides, test-case prioritization may have variants besides its default setting. Traditionally, test-case prioritization aims to address the test effectiveness problem when the total execution time of test cases are big. However, in practice, it may be costly to run an individual test case. In particular, a test suite may consist of only several test cases, each of which consumes long execution time. Therefore, it is also necessary to optimize the execution of test cases due to the cost of individual test cases rather than the total cost. Apparently, such test-case prioritization is different from the existing prioritization problem, and thus a totally new method for this problem is needed.

4 Conclusion

In this article, we briefly introduce the history of test-case prioritization and its representation, and quickly review the existing work of test-case prioritization from five aspects: coverage criteria, prioritization algorithms, measurement, constraints, and application scenarios. Through the analysis, we present the challenges in the existing work, including the approaches, measurement, evaluation, and practical concerns.

Acknowledgements This work was supported by the National Basic Research Program of China (2015CB352201), the National Natural Science Foundation of China (Grant Nos. 61421091, 61225007, 61522201, 61272157, and 61529201).

References

1. Wong W E, Horgan J R, London S, Agrawal H. A study of effective regression testing in practice. In: Proceedings of the 8th IEEE International Symposium on Software Reliability Engineering. 1997, 264–274
2. Rothermel G, Untch R H, Chu C Y, Harrold M J. Test case prioritization: an empirical study. In: Proceedings of the IEEE International Conference on Software Maintenance. 1999, 179–188
3. Rothermel G, Untch R H, Chu C Y, Harrold M J. Prioritizing test cases for regression testing. IEEE Transactions on Software Engineering, 2001, 27(10): 929–948
4. Li Z, Harman M, Hierons R M. Search algorithms for regression test case prioritization. IEEE Transactions on Software Engineering, 2007, 33(4): 225–237
5. Zhang L, Hou S S, Guo C, Xie T, Mei H. Time-aware test-case prioritization using integer linear programming. In: Proceedings of the 18th ACM International Symposium on Software Testing and Analysis. 2009, 213–224
6. Zhang J, Wang X Y, Hao D, Xie B, Zhang L, Mei H. A survey on bug-report analysis. SCIENCE CHINA Information Sciences, 2015, 58(2): 1–24
7. Zhang L M, Hao D, Zhang L, Rothermel G, Mei H. Bridging the gap between the total and additional test-case prioritization strategies. In: Proceedings of the 35th IEEE International Conference on Software Engineering. 2013, 192–201
8. Hao D, Zhang L M, Zhang L, Rothermel G, Mei H. A unified test case prioritization approach. ACM Transactions on Software Engineering and Methodology, 2014, 10: 1–31
9. Jones J A, Harrold M J. Test-suite reduction and prioritization for modified condition/decision coverage. IEEE Transactions on Software Engineering, 2003, 29(3): 195–209
10. Zhang L M, Zhou J, Hao D, Zhang L, Mei H. Prioritizing JUnit test cases in absence of coverage information. In: Proceedings of the IEEE International Conference on Software Maintenance. 2009, 19–28
11. Mei H, Hao D, Zhang L M, Zhang L, Zhou J, Rothermel G. A static approach to prioritizing junit test cases. IEEE Transactions on Software Engineering, 2012, 38(6): 1258–1275
12. Korel B, Tahat L H, Harman M. Test prioritization using system models. In: Proceedings of the 21st IEEE International Conference on Software Maintenance. 2005, 559–568
13. Korel B, Koutsogiannakis G, Tahat L H. Application of system models in regression test suite prioritization. In: Proceedings of the IEEE International Conference on Software Maintenance. 2008, 247–256
14. Ding J H, Xu D X. Prioritizing state-based aspect tests. In: Proceedings of the 3rd IEEE International Conference on Software Testing, Verification and Validation. 2010, 265–274
15. Elbaum S, Malishevsky A G, Rothermel G. Test case prioritization: a family of empirical studies. IEEE Transactions on Software Engineering, 2002, 28(2): 159–182
16. Ma Z K, Zhao J J. Test case prioritization based on analysis of program structure. In: Proceedings of the 15th Asia-Pacific Software Engineering Conference. 2008, 471–478
17. Jiang B, Chan W K. Input-based adaptive randomized test case prioritization: a local beam search approach. Journal of Systems and Software, 2015, 105: 91–106
18. Li S H, Bian N W, Chen Z Y, You D J, He Y C. A simulation study on some search algorithms for regression test case prioritization. In: Proceedings of the 10th IEEE International Conference on Quality Software. 2010, 72–81
19. Hao D, Zhang L, Zang L, Wang Y B, Wu X X, Xie T. To be optimal or not in test-case prioritization. IEEE Transactions on Software Engineering, 2015
20. Elbaum S, Malishevsky A, Rothermel G. Incorporating varying test

costs and fault severities into test case prioritization. In: Proceedings of the 23rd International Conference on Software Engineering. 2001, 329–338

21. Qu X, Cohen M B, Woolf K M. Combinatorial interaction regression testing: a study of test case generation and prioritization. In: Proceedings of the IEEE International Conference on Software Maintenance. 2007, 255–264
22. Walcott K R, Soffa M L, Kapfhammer G M, Roos R S. Time-aware test suite prioritization. In: Proceedings of the 2006 ACM International Symposium on Software Testing and Analysis. 2006, 1–12
23. Alspaugh S, Walcott K R, Belanich M, Kapfhammer G M, Soffa M L. Efficient time-aware prioritization with knapsack solvers. In: Proceedings of the 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies. 2007, 17–31
24. Suri B, Singhal S. Analyzing test case selection & prioritization using ACO. ACM SIGSOFT Software Engineering Notes, 2011, 36(6): 1–5
25. Do H, Mirarab S, Tahvildari L, Rothermel G. The effects of time constraints on test case prioritization: a series of controlled experiments. IEEE Transactions on Software Engineering, 2010, 36(5): 593–617
26. Park H, Ryu H, Baik J. Historical value-based approach for cost-cognizant test case prioritization to improve the effectiveness of regression testing. In: Proceedings of the 2nd IEEE International Conference on Secure System Integration and Reliability Improvement. 2008, 39–46
27. Zhang X F, Nie C H, Xu B W, Qu B. Test case prioritization based on varying testing requirement priorities and test case costs. In: Proceedings of the 7th IEEE International Conference on Quality Software. 2007, 15–24
28. Srivastava A, Thiagarajan J. Effectively prioritizing tests in development environment. ACM SIGSOFT Software Engineering Notes, 2002, 27(4): 97–106
29. Hao D, Zhao X, Zhang L. Adaptive test-case prioritization guided by output inspection. In: Proceedings of the 37th IEEE Annual Computer Software and Applications Conference. 2013, 169–179
30. Lou Y L, Hao D, Zhang L. Mutation-based test-case prioritization in software evolution. In: Proceedings of the 26th IEEE International Symposium on Software Reliability Engineering. 2015, 46–57
31. Chen J J, Bai Y W, Hao D, Xiong Y F, Zhang H Y, Zhang L, Xie B. A text-vector based approach to test case prioritization. In: Proceedings of the IEEE International Conference on Software Testing, Verificationa and Validation. 2016, 266–277
32. Thomas S W, Hemmati H, Hassan A E, Blostein D. Static test case prioritization using topic models. Empirical Software Engineering, 2014, 19(1): 182–212
33. Elbaum S, Malishevsky A G, Rothermel G. Prioritizing test cases for regression testing. In: Proceedings of the 2000 ACM SIGSOFT International Symposium on Software Testing and Analysis. 2000, 102–112
34. Wong W E, Horgan J R, London S, Agrawal H. A study of effective regression testing in practice. In: Proceedings of the 8th IEEE International Symposium on Software Reliability Engineering. 1997, 230–238
35. Jones J A, Harrold M J. Test-suite reduction and prioritization for modified condition/decision coverage. IEEE Transactions on Software Engineering, 2003, 29(3): 195–209
36. Fang C R, Chen Z Y, Wu K, Zhao Z H. Similarity-based test case prioritization using ordered sequences of program entities. Software Quality Journal, 2014, 22(2): 335–361
37. Mei L, Chan W K, Tse T H. Data flow testing of service-oriented workflow applications. In: Proceedings of the 30th ACM/IEEE International Conference on Software Engineering. 2008, 371–380
38. Voas J M. PIE: a dynamic failure-based technique. IEEE Transactions on Software Engineering, 1992, 18(8): 717–727
39. Do H, Rothermel G. On the use of mutation faults in empirical assessments of test case prioritization techniques. IEEE Transactions on Software Engineering, 2006, 32(9): 733–752
40. Chen T Y, Lau M E, Yu Y T. MUMCUT: a fault-based stategy for testing boolean specifications. In: Proceedings of the 6th Asia Pacific Software Engineering Conference. 1999, 606–613
41. Chen Z Y, Chen T Y, Xu B W. A revisit of fault class hierachies in general boolean specifications. ACM Transactions on Software Engineering and Methodology (TOSEM), 2011, 20(3): 13
42. Fang C R, Chen Z Y, Xu B W. Comparing logic coverage criteria on test case prioritization. SCIENCE CHINA Information Sciences, 2012, 55(12): 2826–2840
43. Reeves C R. Modern Heuristic Techniques for Combinatorial Problems. Oxford: Blackwell Scientific Publications, 1993
44. Saha R K, Zhang L M, Khurshid S, Perry D E. REPiR: an information retrieval based approach for regression test prioritization. In: Proceedings of the 37th IEEE International Conference on Software Engineering. 2015, 268–279
45. Tonella P, Avesani P, Susi A. Using the case-based ranking methodology for test case prioritization. In: Proceedings of the 22nd IEEE International Conference on Software Maintenance. 2006, 123–133
46. Arafeen M J, Do H. Test case prioritization using requirements based clustering. In: Proceeding of the 6th IEEE International Conference on Software Testing, Verification and Validation. 2013, 312–321
47. Do H, Rothermel G. An empirical study of regression testing techniques incorporating context and lifetime factors and improved cost-benefit models. In: Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering. 2006, 141–151
48. Do H, Rothermel G. Using sensitivity analysis to create simplified economic models for regression testing. In: Proceedings of the 2008 ACM International Symposium on Software Testing and Analysis. 2008, 51–62
49. Kim J M, Porter A. A history-based test prioritization technique for regression testing in resource constrained environments. In: Proceedings of the 24th IEEE International Conference on Software Engineering. 2002, 119–129
50. Huang Y C, Peng K L, Huang C Y. A history-based cost-cognizant test case prioritization technique in regression testing. Journal of Systems and Software, 2012, 85(3): 626–637
51. Hou S S, Zhang L, Xie T, Sun J. Quota-constrained test-case prioritization for regression testing of service-centric systems. In: Proceedings of the IEEE International Conference on Software Maintenance. 2008, 257–266
52. Do H, Rothermel G, Kinneer A. Prioritizing JUnit test cases: an empirical assessment and cost-benefits analysis. Empirical Software Engineering, 2006, 11(1): 33–70
53. Do H, Rothermel G. A controlled experiment assessing test case prioritization techniques via mutation faults. In: Proceedings of the 21st

IEEE International Conference on Software Maintenance. 2005, 411–420

54. Do H, Mirarab S, Tahvildari L, Rothermel G. An empirical study of the effect of time constraints on the cost-benefits of regression testing. In: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering. 2008, 71–82
55. Lu Y F, Lou Y L, Chen S Y, Zhang L M, Hao D, Zhou Y F, Zhang L. How does regression test prioritization perform in real-world software evolution? In: Proceedings of the 38th International Conference on Software Engineering. 2016

Dan Hao is an associate professor at the School of Electronics Engineering and Computer Science, Peking University, China. She received her PhD in computer science from Peking University in 2008, and the BS in computer science from the Harbin Institute of Technology, China in 2002. She is the awardee of the NSFC Excellent Young Scholars Program in 2015, and a senior member of ACM. Her current research interests include software testing and debugging.

Lu Zhang is a professor at the School of Electronics Engineering and Computer Science, Peking University, China. He received his PhD and BSc from Peking University in 2000 and 1995, respectively. He was a postdoctoral researcher in Oxford Brookes Universityand University of Liverpool, UK. He has served on the editorial boards of Journal of Software Maintenance and Evolution: Research and Practice and Software Testing, Verification and Reliability. He also served on the program committees of many major conferences. His current research interests include software testing, software analysis, program comprehension, software maintenance, software reuse, and service computing.

Hong Mei is a professor at the School of Electronics Engineering and Computer Science, Peking University, China. He received his PhD in computer science from Shanghai Jiaotong University, China in 1992. He is a senior member of IEEE. He was a program co-chair of many software conferences, including COMPSAC2005, QSIC2006, COMPSAC2007, ICSR2008, ICSM2008, and ICWS2008. He also serves on the editorial board of IEEE Transactions on Service Computing and International Journal of Web Services Research. His current research interests include software engineering and software engineering environment, software reuse and software component technology, distributed object technology, and programming language.