Dynamic Random Testing of Web Services: A Methodology and Evaluation

Chang-ai Sun, *Senior Member, IEEE,* Hepeng Dai, Guan Wang, Dave Towey, *Member, IEEE,* Kai-Yuan Cai, *Member, IEEE,* and Tsong Yueh Chen, *Member, IEEE,*

Abstract—In recent years, Service Oriented Architecture (SOA) has been increasingly adopted to develop distributed applications in the context of the Internet. To develop reliable SOA-based applications, an important issue is how to ensure the quality of web services. In this paper, we propose a dynamic random testing (DRT) technique for web services, which is an improvement over the widely-practiced random testing (RT) and partition testing (PT). We examine key issues when adapting DRT to the context of SOA, including a framework, guidelines for parameter settings, and a prototype for such an adaptation. Empirical studies are reported where DRT is used to test three real-life web services, and mutation analysis is employed to measure the effectiveness. Our experimental results show that, compared with the two baseline techniques, RT and Random Partition Testing (RPT), DRT demonstrates higher fault-detection effectiveness with a lower test case selection overhead. Furthermore, the theoretical guidelines of parameter setting for DRT are confirmed to be effective. The proposed DRT and the prototype provide an effective and efficient approach for testing web services.

Index Terms—Software Testing, Random Testing, Dynamic Random Testing, Web Service, Service Oriented Architecture.

1 Introduction

Test result verification is an important part of software testing. A test oracle [1] is a mechanism that can exactly decide whether the output produced by a programs is correct. However, there are situations where it is difficult to decide whether the result of the software under test (SUT) agrees with the expected result. This situation is known as oracle problem [2], [3]. In order to alleviate the oracle problem, several techniques have been proposed, such as N-version testing [4], metamorphic testing (MT) [5], [6], assertions [7], and machine learning [8]. Among of them, MT obtains metamorphic relations (MRs) according to the properties of SUT. Then, MRs are used to generate new test cases called follow-up test cases from original test cases known as source test cases. Next, both source and follow-up test cases are executed and their result are verified against the corresponding MRs.

The fault-detecting efficiency of MT relies on the quality of MRs and the source test cases. There are astronomically large number of studies to investigate generate good MRs [9]–[13] or the source test cases [14]–[16]. However, Researchers ignore the impact of test execution on the efficiency of MT. Random testing (RT) that randomly selects test cases from input domain (which refers to the set of all possible inputs of SUT), which is most commonly used technique in traditional MT [17]. Although RT is simple to implement, RT does not make use of any execution information about SUT or the test history. Thus, traditional MT may be inefficient in some situations.

In contrast to RT, partition testing (PT) attempts to generate test cases in a more systematic way, aiming to use fewer test cases to detect more faults. When conducting PT, the input domain of SUT is divided into disjoint partitions, with test cases then selected from each and every one. Each partition is expected to have a certain degree of homogeneity, that is, test cases in the same partition should have similar software execution behavior: If one input can detect a fault, then all other inputs in the same partition should also be able to detect a fault. RT and PT have their own characteristics. Therefore, it is a nature thought to investigate the integration of them for developing new testing techniques. Sun et al. [18] proposed adaptive partition testing (APT) that takes advantages of testing information to control the testing process, with the goal of benefitting from the advantages of RT and PT.

In this study, we investigate how to make use of feedback information in the previous tests to control the execution process of MT. As a result, an adaptive metamorphic testing framework is proposed to improve the fault-detecting efficiency of MT. An empirical study was conducted to evaluate the efficiency of the proposed technique. Main contributions made in this paper are the following:

The rest of this paper is organized as follows. Section 2 introduces the underlying concepts for DRT, web services and mutation analysis. Section **??** presents the DRT framework for web services, guidelines for its parameter settings, and a prototype that partially automates DRT. Section **??** describes an empirical study where the proposed DRT is

A preliminary version of this paper was presented at the 36th Annual IEEE Computer Software and Applications Conference (COMPSAC 2012) [?].
C.-A. Sun, H. Dai, and G. Wang are with the School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China. E-mail: casun@ustb.edu.cn.
D. Towey is with the School of Computer Science, University of Nottingham Ningbo China, Ningbo 315100, China. E-mail: dave.towey@nottingham.edu.cn
K.-Y. Cai is with the School of Automation Science and Electrical Engineering, Beihang University, Beijing 100191, China. E-mail: kycai@buaa.edu.cn.
T.Y. Chen is with the Department of Computer Science and Software Engineering, Swinburne University of Technology, Hawthorn VIC 3122, Australia. Email: tychen@swin.edu.au.

used to test three real-life web services, the results of which are summarized in Section 5. Section 6 discusses related work and Section **??** concludes the paper.

2 BACKGROUND

In this section, we present some of the underlying concepts for DRT, web services, and mutation analysis.

2.1 Metamorphic Testing (MT)

MT is a novel technique to alleviate the oracle problem: Instead of applying an oracle, MT uses a set of MRs (corresponding to some specific properties of the SUT) to verify the test results. MT is normally conducted according to the following steps:

Step1. Identify an MR from the specification of the SUT.
Step2. Generate the source test case stc using the traditional test cases generation techniques.
Step3. Derive the follow-up test case ftc from the stc based on the MR.
Step3. execute stc and ftc and get their outputs O_s and O_f.
Step4. Verify stc, ftc, O_s, and O_f against the MR: If the MR does not hold, a fault is said to be detected.

The above steps can be repeated for a set of MRs.

Let us use a simple example to illustrate how MT works. For instance, consider the mathematic function $f(x, y)$ that can calculate the maximal value of two integers x and y. There is a simple yet obvious property: the order of two parameters x and y does not affect the output, which can be described as the follow metamorphic relation (MR): $f(x, y) = f(y, x)$. In this MR, (x, y) is source test case, and (y, x) is considered as follow-up test case. Suppose `P` denotes a program that implements the function $f(x, y)$, `P` is executed with a test cases $(1, 2)$ and $(2, 1)$. Then we check $P(1, 2) = P(2, 1)$: If the equality does not hold, then we consider that `P` at least has one fault.

2.2 Adaptive Partition Testing (APT)

DRT combines RT and PT [31], with the goal of benefitting from the advantages of both. Given a test suite *TS* classified into m partitions (denoted $s_1, s_2, \ldots, s_m$), suppose that a test case from s_i ($i = 1, 2, \ldots, m$) is selected and executed. If this test case reveals a fault, $\forall j = 1, 2, \ldots, m$ and $j \neq i$, we then set

$$p'_j = \begin{cases} p_j - \dfrac{\varepsilon}{m-1} & \text{if } p_j \geq \dfrac{\varepsilon}{m-1} \\ 0 & \text{if } p_j < \dfrac{\varepsilon}{m-1} \end{cases}, \quad (1)$$

where ε is a probability adjusting factor, and then

$$p'_i = 1 - \sum_{\substack{j=1 \\ j \neq i}}^{m} p'_j. \quad (2)$$

Alternatively, if the test case does not reveal a fault, we set

$$p'_i = \begin{cases} p_i - \varepsilon & \text{if } p_i \geq \varepsilon \\ 0 & \text{if } p_i < \varepsilon \end{cases}, \quad (3)$$

and then for $\forall j = 1, 2, \ldots, m$ and $j \neq i$, we set

$$p'_j = \begin{cases} p_j + \dfrac{\varepsilon}{m-1} & \text{if } p_i \geq \varepsilon \\ p_j + \dfrac{p'_i}{m-1} & \text{if } p_i < \varepsilon \end{cases}. \quad (4)$$

Algorithm 1 describes DRT. In DRT, the first test case is taken from a partition that has been randomly selected according to the initial probability profile $\{p_1, p_2, \ldots, p_m\}$ (Lines 2 and 3 in Algorithm 1). After each test case execution, the testing profile $\{\langle s_1, p_1 \rangle, \langle s_2, p_2 \rangle, \ldots, \langle s_m, p_m \rangle\}$ is updated by changing the values of p_i: If a fault is revealed, Formulas 1 and 2 are used; otherwise, Formulas 3 and 4 are used. The updated testing profile is then used to guide the random selection of the next test case (Line 8). This process is repeated until a termination condition is satisfied (Line 1). Examples of possible termination conditions include: "testing resources have been exhausted"; "a certain number of test cases have been executed"; and "a certain number of faults have been detected".

Algorithm 1 DRT

Input: $\varepsilon, p_1, p_2, \ldots, p_m$
1: **while** termination condition is not satisfied
2: Select a partition s_i according to the testing profile $\{\langle s_1, p_1 \rangle, \langle s_2, p_2 \rangle, \ldots, \langle s_m, p_m \rangle\}$.
3: Select a test case t from s_i.
4: Test the software using t.
5: **if** a fault is revealed by t
6: Update p_j ($j = 1, 2, \ldots, m$ and $j \neq i$) and p_i according to Formulas 1 and 2.
7: **else**
8: Update p_j ($j = 1, 2, \ldots, m$ and $j \neq i$) and p_i according to Formulas 3 and 4.
9: **end_if**
10: **end_while**

As can be seen from Formulas 1 to 4, updating the testing profile involves m simple calculations, thus requiring a constant time. Furthermore, the selection of partition s_i, and subsequent selection and execution of the test case, all involve a constant time. The execution time for one iteration of DRT is thus a constant, and therefore the overall time complexity for DRT to select n test cases is $O(m \cdot n)$.

2.3 Adaptive Random Testing (ART)

3 SDMT

In this section, we describe a framework for applying DRT to web services, discuss guidelines for DRT's parameter settings, and present a prototype that partially automates DRT for web services.

3.1 Motivation

3.2 Framework

1 *Source Test Case Selection*.
2 *candidate MRs Selection*.
3 *MR Selection*.
4 *Follow-up Test Case Generation*.

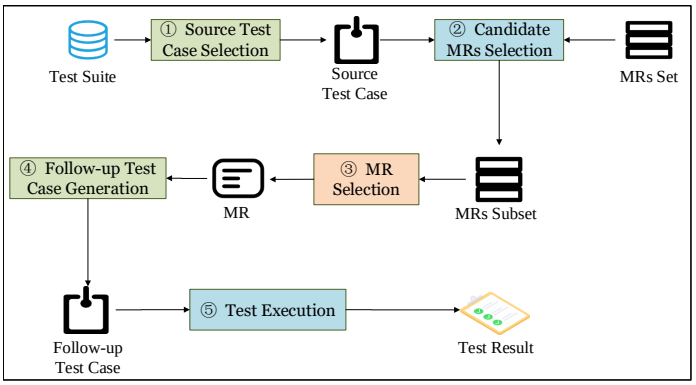

Fig. 1. framework

5 *Test Case Execution*. The relevant DRT component receives the generated test case, converts it into an input message, invokes the web service(s) through the SOAP protocol, and intercepts the test results (from the output message).

3.3 Updating Test Profile Strategies

3.4 Selecting MR Strategies

3.5 Partition Based SDMT

3.6 Random Based SDMT

4 Empirical Study

We conducted a series of empirical studies to evaluate the performance of DRT.

4.1 Research Questions

In our experiments, we focused on addressing the following three research questions:

RQ1 **How efficient is SDMT at detecting faults?** Fault-detection efficiency is a key criterion for evaluating the performance of a testing technique. In our study, we chose three real-life programs, and applied mutation analysis to evaluate the fault-detecting efficiency.

RQ2 **What is the actual test case selection overhead when using the M-AMT strategy?** We evaluate the test case selection overhead of M-AMT and compare with traditional MT in detecting software faults.

4.2 Object Programs

In order to address our research questions, we chose to study four sets of object programs: three laboratorial programs that were developed according to corresponding specifications, seven programs from the Software-artifact Infrastructure Repository (SIR) [19], the regular expression processor component of the larger utility program `GUN grep`, and a Java library developed by Alibaba, which can be used to convert Java Objects into their JSON representation, and convert a JSON string to an equivalent Java object. The details of twelve programs are summarized in Table 1.

There four sets of programs present complementary strengths and weaknesses as experiment objects. The Laboratorial programs implement simple functions and their interfaces are easy to understand. The test engineers can easily generate source test cases, and identify MRs for testing laboratorial programs. However, these programs are small and there are a limited number of faulty versions available for each programs. The SIR repositories provides object programs, including a number of pre-existing versions with seeded faults, as well as a test suite in which test cases were randomly generated. It is a challenge task to partition the input domain of those programs. The `grep` program is a much larger system for which mutation faults could be generated. The `FastJson` is also a much larger system, and the faults of that are obtained on GitHub. We provide further details on each of those sets of object programs next.

4.2.1 Laboratorial Programs

Based on three real-lift specifications, We developed three systems, respectively. `China Unicom Billing System (CUBS)` provides an interface through which customers can know how much they need to pay according to plans, month charge, calls, and data usage. The details of two cell-phone plans are summarized in Tables 2 and 3. `Aviation Consignment Management System (ACMS)` aims to help airline companies check the allowance (weight) of free baggage, and the cost of additional baggage. Based on the destination, flights are categorized as either domestic or international. For international flights, the baggage allowance is greater if the passenger is a student (30kg), otherwise it is 20kg. Each aircraft offers three cabins classes from which to choose (economy, business, and first), with passengers in different classes having different allowances. The detailed price rules are summarized in Table 4, where $price_0$ means economy class fare. `Expense Reimbursement System (ERS)` assists the sales Supervisor of a company with handling the following tasks: i) Calculating the cost of the employee who use the cars based on their titles and the number of miles actually traveled; ii) accepting the requests of reimbursement that include airfare, hotel accommodation, food and cell-phone expenses of the employee.

TABLE 2
Plan A

Plan details		Month charge (CNY)			
		$option_1$	$option_2$	$option_3$	$option_4$
Basic	Free calls (min)	50	96	286	3000
	Free data (MB)	150	240	900	3000
Extra	Dialing calls (CNY/min)	0.25	0.15	0.15	0.15
	Data (CNY/KB)	0.0003			

TABLE 3
Plan B

Plan details		Month charge (CNY)			
		$option_1$	$option_2$	$option_3$	$option_4$
Basic	Free calls (min)	120	450	680	1180
	Free data (MB)	40	80	100	150
Extra	Dialing calls (CNY/min)	0.25	0.15	0.15	0.15
	Data (CNY/KB)	0.0003			

4.2.2 SIR Programs

The seven object programs selected from the SIR [24] repository were `printtokens`, `printtokens2`, `replace`,

TABLE 1
Twelve Programs as Experimental Objects

programs	Source	Language	LOC	All Faults	Used Faults	Number of MRs	Number of Partitions
CUBS	Laboratory	Java	107	187	4	184	8
ACMS	Laboratory	Java	128	210	9	142	4
ERS	Laboratory	Java	117	180	4	1130	12
grep	GUN	c	10,068	20	20	12	–
printtokens	SIR	c	483	7	7	3	–
printtokens2	SIR	c	402	10	10	3	–
replace	SIR	c	516	31	31	3	–
schedule	SIR	c	299	9	9	3	–
schedule2	SIR	c	297	9	9	3	–
tcas	SIR	c	138	41	41	3	–
totinfo	SIR	c	346	23	23	3	–
FastJson	Alibaba	Java	204125	6	6	–	–

TABLE 4
ACMS Baggage Allowance and Pricing Rules

	Domestic flights			International flights		
	First class	Business class	Economy class	First class	Business class	Economy class
Carry on (kg)	5	5	5	7	7	7
Free checked-in (kg)	40	30	20	40	30	20/30
Additional baggage pricing (kg)	$price_0 * 1.5\%$			$price_0 * 1.5\%$		

schedule, schedule2, tcas, and totinfo. These programs were originally compiled by Hutchins et al. [20] at Siemens Corporate Research. We used these programs for several reasons:

1 Faulty versions of the programs are available.
2 The programs are of manageable size and complexity for an initial study.
3 All programs and related materials are available from the SIR, and the MRs of each program are defined in [21].

The printtokens and printtokens2 are independent implementations of the same specification. They each implement a lexical analyzer. Their input files are split into lexical tokens according to a tokenizing scheme, and their output is the separated tokens. The replace program is a command-line utility that takes a search string, a replacement string, and an input file. The search string is a regular expression. The replacement string is text, with some metacharacters to enable certain features. The replace program searches for occurrences of the search string in the input file, and produces an output file where each occurrence of the search string is replaced with the replacement string. The schedule and schedule2 programs are also independent implementations of the same specification. They each implement a priority scheduler that takes three non-negative integers and an input file. The integers indicate the number of initial processes at the three available scheduling priorities. The schedule and schedule2 programs then take as input a command file that specifies certain actions to be taken. The tcas program is a small part of a much larger aeronautical collision avoidance system. It performs simple analysis on data provided by other parts of the system and returns an integer indicating what action, if any, the pilot should take to avoid collision.

4.2.3 GUN grep

The used version of the grep programs is 2.5.1a [22]. This program searches one or more input files for lines containing a match to a specified pattern. By default, grep prints the matching lines. We chose grep for our study for several reasons:

1 The grep program is wide used in Unix system, providing a opportunity to demonstrate the real world relevance of our techniques.
2 The grep program, and its input format, are of greater complexity than the the programs in the other test sets, but still manageable as a target for automated test case generation.

The inputs of the grep were categorize into three components: options, which consist of a list of commands to modify the searching process, pattern, which is the regular expression to be searched for, and files, which refers to the input files to be searched.

The scope of functionality of this program is larger, which leads to construct test infrastructure to test all of functionality would have been impractical. Therefore, we restricted our focus to the regular expression analyzer of grep.

4.2.4 Real-World Popular Programs

The FastJson

4.3 Variables

4.3.1 Independent Variables

The independent variable is the testing technique, DRT. RPT and RT were used as baseline techniques for comparison.

4.3.2 Dependent Variables

The dependent variable for RQ1 is the metric for evaluating the fault-detection effectiveness. Several effectiveness metrics exist, including: the P-measure [23] (the probability of at least one fault being detected by a test suite); the E-measure [**?**] (the expected number of faults detected by a

test suite); the F-measure [18] (the expected number of test case executions required to detect the first fault); and the T-measure [**?**] (the expected number of test cases required to detect all faults). Since the F- and T-measures have been widely used for evaluating the fault-detection efficiency and effectiveness of DRT-related testing techniques [**?**], [**?**], [24]–[27], they are also adopted in this study. We use F and T to represent the F-measure and the T-measure of a testing method. As shown in Algorithm 1, the testing process may not terminate after the detection of the first fault. Furthermore, because the fault detection information can lead to different probability profile adjustment mechanisms, it is also important to see what would happen after revealing the first fault. Therefore, we introduce the F2-measure [18] as is the number of additional test cases required to reveal the second fault after detection of the first fault. We use $F2$ to represent the F2-measure of a testing method, and $SD_{measure}$ to represent the standard deviation of metrics (where $measure$ can be F, $F2$, or T).

An obvious metric for RQ3 is the time required to detect faults. Corresponding to the T-measure, in this study we used T-$time$, the time required to detect all faults. F-$time$ and $F2$-$time$ denote the time required to detect the first fault, and the additional time needed to detect the second fault (after detecting the first), respectively. For each of these metrics, smaller values indicate a better performance.

4.4 Experimental Settings

4.4.1 Partitioning

4.4.2 Initial Test Profile

4.4.3 Constants

4.5 Experimental Environment

4.6 Experiment Architecture

4.7 Threats To Validity

4.7.1 Internal Validity

4.7.2 External Validity

4.7.3 Construct Validity

4.7.4 Conclusion Validity

5 Experimental Results

6 Related Work

In this section, we describe related work from two perspectives: related to testing techniques for web services; and related to improving RT and PT.

Acknowledgment

This research is supported by the National Natural Science Foundation of China (Grant Nos. 61872039 and 61872167), the Beijing Natural Science Foundation (Grant No. 4162040), the Aeronautical Science Foundation of China (Grant No. 2016ZD74004), and the Fundamental Research Funds for the Central Universities (Grant No. FRF-GF-17-B29).

References

[1] E. J. Weyuker, "On testing non-testable programs," *The Computer Journal*, vol. 25, no. 4, pp. 465–470, 1982.

[2] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 507–525, 2015.

[3] K. Patel and R. M. Hierons, "A mapping study on testing non-testable systems," *Software Quality Journal*, vol. 26, no. 4, pp. 1373–1413, 2018.

[4] S. S. Brilliant, J. C. Knight, and P. Ammann, "On the performance of software testing using multiple versions," in *Proceedings of the 20th International Symposium on Fault-Tolerant Computing (FTCS'90)*, 1990, pp. 408–415.

[5] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: A new approach for generating next test cases," Department of Computer Science , Hong Kong University of Science and Technology, Hong Kong, Tech. Rep. HKUST-CS98-01, Tech. Rep., 1998.

[6] T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. Tse, and Z. Q. Zhou, "Metamorphic testing: A review of challenges and opportunities," *ACM Computing Surveys*, vol. 51, no. 1, p. 4, 2018.

[7] K. Y. Sim, C. S. Low, and F.-C. Kuo, "Eliminating human visual judgment from testing of financial charting software," *Journal of Software*, vol. 9, no. 2, pp. 298–312, 2014.

[8] W. K. Chan, S.-C. Cheung, J. C. Ho, and T. Tse, "Pat: A pattern classification approach to automatic reference oracles for the testing of mesh simplification programs," *Journal of Systems and Software*, vol. 82, no. 3, pp. 422–434, 2009.

[9] T. Y. Chen, D. Huang, T. Tse, and Z. Q. Zhou, "Case studies on the selection of useful relations in metamorphic testing," in *Proceedings of the 4th lberoAmerican Symposium on Software Engineering and Knowledge Engineer-ing (JIISIC'04)*, 2004, pp. 569–583.

[10] Y. Cao, Z. Q. Zhou, and T. Y. Chen, "On the correlation between the effectiveness of metamorphic relations and dissimilarities of test case executions," in *Proceedings of the 13th International Conference on Quality Software (QSIC'13)*, 2013, pp. 153–162.

[11] C.-A. Sun, G. Wang, B. Mu, H. Liu, Z. Wang, and T. Y. Chen, "Metamorphic testing for web services: Framework and a case study," in *Proceedings of the 9th IEEE International Conference on Web Services (ICWS'11)*, 2011, pp. 283–290.

[12] T. Chen, P. Poon, and X. Xie, "Metric: Metamorphic relation identification based on the category-choice framework," *Journal of Systems and Software*, vol. 116, pp. 190–14, 2014.

[13] X. Xie, J. Li, C. Wang, and T. Y. Chen, "Looking for an mr? try metwiki today," in *Proceedings of the 1st International Workshop on Metamorphic Testing (MET'16), Co-located with the 38th International Conference on Software Engineering (ICSE'16)*, 2016, pp. 1–4.

[14] T. Y. Chen, F.-C. Kuo, Y. Liu, and A. Tang, "Metamorphic testing and testing with special values." in *Proceedings of the 5th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD'04)*, 2004, pp. 128–134.

[15] G. Batra and J. Sengupta, "An efficient metamorphic testing technique using genetic algorithm," in *International Conference on Information Intelligence, Systems, Technology and Management*, 2011, pp. 180–188.

[16] G. Dong, T. Guo, and P. Zhang, "Security assurance with program path analysis and metamorphic testing," in *Proceedings of the 4th IEEE International Conference on Software Engineering and Service Science (ICSESS'13)*, 2013, pp. 193–197.

[17] S. Segura, G. Fraser, A. B. Sanchez, and A. Ruiz-Cortés, "A survey on metamorphic testing," *IEEE Transactions on Software Engineering*, vol. 42, no. 9, pp. 805–824, 2016.

[18] C.-A. Sun, H. Dai, H. Liu, T. Y. Chen, and K.-Y. Cai, "Adaptive partition testing," *IEEE Transactions on Computers*, 2018.

[19] H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empirical Software Engineering*, vol. 10, no. 4, pp. 405–435, 2005.

[20] 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 16th International conference on Software engineering*. IEEE, 1994, pp. 191–200.

[21] X. Xie, W. E. Wong, T. Y. Chen, and B. Xu, "Metamorphic slice: An application in spectrum-based fault localization," *Information and Software Technology*, vol. 55, no. 5, pp. 866–879, 2013.

[22] T. G. Project, "Grep home page. http://www.gnu.org/software/grep," 2006.
[23] J. W. Duran and S. C. Ntafos, "An evaluation of random testing," *IEEE transactions on Software Engineering*, no. 4, pp. 438–444, 1984.
[24] K.-Y. Cai, B. Gu, H. Hu, and Y.-C. Li, "Adaptive software testing with fixed-memory feedback," *Journal of Systems and Software*, vol. 80, no. 8, pp. 1328–1348, 2007.
[25] K. Cai, H. Hu, C.-h. Jiang, and F. Ye, "Random testing with dynamically updated test profile," in *Proceedings of the 20th International Symposium On Software Reliability Engineering (ISSRE'09)*, 2009, pp. 1–2.
[26] J. Lv, H. Hu, and K.-Y. Cai, "A sufficient condition for parameters estimation in dynamic random testing," in *Proceedings of the 35th IEEE Annual International Computer Software and Applications Conference Workshops (COMPSACW'11)*, 2011, pp. 19–24.
[27] Z. Yang, B. Yin, J. Lv, K. Cai, S. S. Yau, and J. Yu, "Dynamic random testing with parameter adjustment," in *Proceedings of the 6th IEEE International Workshop on Software Test Automationthe, Co-located with the 38th IEEE Annual International Computer Software and Applications Conference (COMPSAC14)*, 2014, pp. 37–42.

Chang-ai Sun is a Professor in the School of Computer and Communication Engineering, University of Science and Technology Beijing. Before that, he was an Assistant Professor at Beijing Jiaotong University, China, a postdoctoral fellow at the Swinburne University of Technology, Australia, and a postdoctoral fellow at the University of Groningen, The Netherlands. He received the bachelor degree in Computer Science from the University of Science and Technology Beijing, China, and the PhD degree in Computer Science from Beihang University, China. His research interests include software testing, program analysis, and Service-Oriented Computing.

Hepeng Dai is a PhD student in the School of Computer and Communication Engineering, University of Science and Technology Beijing, China. He received the master degree in Software Engineering from University of Science and Technology Beijing, China and the bachelor degree in Information and Computing Sciences from China University of Mining and Technology, China. His current research interests include software testing and debugging.

Guan Wang is a masters student at the School of Computer and Communication Engineering, University of Science and Technology Beijing. He received a bachelor degree in Computer Science from University of Science and Technology Beijing. His current research interests include software testing and Service-Oriented Computing.

Dave Towey is an associate professor in the School of Computer Science, University of Nottingham Ningbo China. He received his BA and MA degrees from The University of Dublin, Trinity College, PgCertTESOL from The Open University of Hong Kong, MEd from The University of Bristol, and PhD from The University of Hong Kong. His current research interests include technology-enhanced teaching and learning, and software testing, especially metamorphic testing and adaptive random testing. He is a member of both the IEEE and the ACM.

Tsong Yueh Chen is a Professor of Software Engineering at the Department of Computer Science and Software Engineering in Swinburne University of Technology. He received his PhD in Computer Science from The University of Melbourne, the MSc and DIC from Imperial College of Science and Technology, and BSc and MPhil from The University of Hong Kong. He is the inventor of metamorphic testing and adaptive random testing.

Kai-Yuan Cai received the BS, MS, and PhD degrees from Beihang university, Beijing, China, in 1984, 1987, and 1991, respectively. He has been a full professor at Beihang University since 1995. He is a Cheung Kong Scholar (chair professor), jointly appointed by the Ministry of Education of China and the Li Ka Shing Foundation of Hong Kong in 1999. His main research interests include software testing, software reliability, reliable flight control, and software cybernatics.