2012 IEEE 36th International Conference on Computer Software and Applications Workshops

Distribution-aware Mutation Analysis

Chang-ai Sun[1,3,*], Guan Wang[1], Kai-Yuan Cai[2], Tsong Yueh Chen[4]
[1]School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing, China
[2]Department of Automatic Control, Beijing University of Aeronautics and Astronautics, Beijing, China
[3]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Science, Beijing, China
[4]Faculty of Information and Communication Technologies, Swinburne University of Technology, Melbourne, Australia
e-mails: casun@ustb.edu.cn, jayjaywg@qq.com, kycai@buaa.edu.cn, tychen@swin.edu.au

***Abstract*—Mutation analysis is widely employed to evaluate the effectiveness of various software testing techniques. In most situations, mutation operators are uniformly applied to the original programs, while the faults tend to be clustered in practice. This may result in the inappropriate simulation of faults, and thus cannot deliver the reliable evaluation results. To overcome this, we propose a distribution-aware mutation analysis technique and conducted empirical studies to investigate the impact of the mutation distribution on the effectiveness evaluation of testing techniques. As an illustration, we select the commonly practiced random testing technique and two versions of dynamic random testing techniques and apply them to testing Web services. Results of empirical studies suggest that the mutation distribution significantly affects the evaluation results. This observation further indicates that the effectiveness of testing techniques previously evaluated with the uniform mutation analysis needs further realignments.**

Keywords-mutation analysis; software testing; performance evaluation.

I. INTRODUCTION

Mutation analysis is a fault-based testing technique and has a long history of study [7]. It applies mutation operators to seed various faults into the program under test, and thus generates a set of mutants. If a test case causes a mutant to show a behavior different from the program under test, we say that this test case can "kill" the mutant and thus detects the fault injected into the mutant. It has been pointed out that compared with manually seeded faults, the automatically generated mutants are more similar to the real-life faults [1], and the mutation score is a good indicator for the effectiveness of a testing technique.

Unfortunately, mutation analysis failed to be widely employed in the software development industry although it is a powerful testing technique [18]. Primary reasons include: (1) mutation analysis is a computationally expensive testing technique, (2) the undecidability of mutant equivalence, and (3) the lack of full and economical automated technologies to support it. In recent years, many improvements have been made with an attempt to turn mutation analysis into a practical testing approach, including various cost reduction techniques and equivalent mutant detection techniques [10].

Besides being used to assess the adequacy of a test suite, mutation analysis is often used to evaluate the effectiveness of testing techniques [1]. In the known scenarios, mutation operators are, at least to our best knowledge, uniformly applied to the original programs. For instance, *MuJava* [19] is a widely recognized mutation system for Java programs and uniformly seeds faults into the program under test. On the other hand, it has been observed that in practice the faults tend to be clustered [16]. This means that the effectiveness evaluation results may be not trustworthy, although mutants generated by such a uniform mutation analysis are similar to real-life faults.

In this paper, we propose an improvement on mutation analysis to deliver reliable effectiveness evaluation of testing techniques. The Pareto principle which declares that 80% faults happen to 20% codes is well known in software testing community for a long time [16]. Inspired by the observations that realistic faults tend to be clustered, we introduce the concept of the distribution of faults into mutation analysis to be aware of the likely clustered characteristics, and conducted empirical studies to investigate the impact of the mutation distribution on the effectiveness evaluation of testing techniques. Results of empirical studies suggest that the mutation distribution significantly affects the evaluation results. To our best knowledge, distribution-aware mutation analysis is the first attempt to extend mutation analysis to deliver more realistic simulation of faults in program under test. Such an observation, although derived from the preliminary empirical studies, further indicates the effectiveness of testing techniques previously evaluated with the uniform mutation analysis needs further realignments.

The rest of the paper is organized as follows. Section II introduces the underlying concepts and relevant techniques of distribution-aware mutation analysis. Section III describes two testing techniques which are used in the subsequent experiments. Section IV describes the experiments we have conducted to investigate the impact of the mutation distribution on the effectiveness evaluation of two commonly used testing techniques described in Section III. Section V describes related work and compares them with distribution-aware mutation analysis. Section VI concludes the paper and points out the future work.

II. DISTRIBUTION-AWARE MUTATION ANALYSIS

We first describe the effectiveness evaluation framework using mutation analysis and then discuss how distribution-aware mutation analysis can be implemented.

* Contact author

978-0-7695-4758-9/12 $26.00 © 2012 IEEE
DOI 10.1109/COMPSACW.2012.40

A. An Evaluation Framework using Mutation Analysis

Figure 1 depicts a generic framework where mutation analysis is employed to evaluate the effectiveness of a testing technique. Such a framework is actually an adaptation of the process of mutation analysis as illustrated in [18]. To evaluate the effectiveness of a testing technique *S*, one usually selects a set of subject programs for evaluation. For each program *P*, he then generates a test suite *TS* using *S*, and runs *TS* on *P* and each mutant *P'*, finally calculates mutation score *MS* as a measurement of effectiveness of *S*.

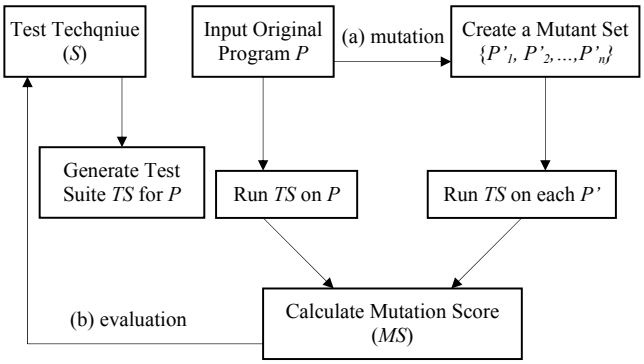

Figure 1. Generic Evaluation Framework with Mutation Analysis

As illustrated in Figure 1, the step from the original program *P* to a set of mutants $\{P'_1, P'_2,...,P'_n\}$ is called mutation. For a specific mutant P_i', it can be derived through applying a mutation operator to program *P*. For instance, *LCR* (*Logical Connector Replacement*) is a common mutation operator; one can apply *LCR* to the occurrences of logical connectors in *P* to achieve a set of mutants. Similarly, one can achieve more mutants by applying other mutation operators. In the known treatments, these mutation operators are uniformly applied to program under test. As a result, the faults simulated by mutants are "uniformly" distributed in the program under test.

B. Implementing Distribution-Aware Mutation Analysis

On the other hands, software testing practitioners have observed that faults are incurred into programs not in an even way, but in a rather clustered way [16]. This means some parts of program may contain more faults, while some parts have fewer ones. This observation is consistent with the Pareto principle which is well-known and widely followed in the software testing practice. To deliver reliable effectiveness evaluation of testing techniques, this observation, in turn, suggests that mutation analysis should be aware of the distribution of faults.

Unfortunately, this has been neglected in the exiting mutation analysis techniques. Traditional mutation analysis is a 5-tuple $E = <P, S, D, L, A>$ [15]. *P* is a given program, *S* is a specification, *D* is a domain of interest, $L = (l_1, l_2, \ldots, l_n)$ is an *n*-tuple of locations in *P*, $A = (A_1, A_2, \ldots, A_n)$ where A_i is an alternative set associated with location l_i, $1 \le i \le n$.

Unlike the traditional mutation analysis (that is, uniform mutation analysis), distribution-aware mutation analysis generates a set of mutants whose associated faults *A* are not always uniformly distributed in program *P*. This can be implemented by introducing the mutation probability *Pr* into the uniform mutation analysis. As a consequence, distribution-aware mutation analysis is a 6-tuple $E = <P, S, D, L, A, Pr>$, where *Pr* is the occurrence probability of each alternative $a_{i,j}$ of A_i, and $0 \le j \le |A_i|$, where A_i is a set of possible alternatives (or faults simulated by applicable mutation operators), and $|A_i|$ denotes the number of applicable mutation operators.

Distribution-aware mutation analysis is an extended version of the uniform mutation analysis. For any given fault distribution model, one can correspondingly achieve a set of mutants. This can be done by randomly selection of mutants generated by the uniform mutation analysis. We propose the following process to implement the distribution-aware mutation analysis.

(1) Set the distribution model *DM* and mutation operators types (namely *A* in *E*).
(2) Apply mutation operators to program *P* as done by the traditional mutation analysis.
(3) Randomly select a "clustered" segment of *P* and keep all mutants within the segment, and randomly remove the mutants out of the segment until the distribution of derived mutants satisfies the specified *DM*. When a mutant (namely $a_{i,j}$ in *E*) is removed, its associated *Pr* is set to 0; otherwise its associated *Pr* is set to 1.

The mutation distribution (or associated faults) can be measured in terms of either *number of statements* or *line of codes*. For example, the following *IF* statement

```
IF (x>5)
   i++;
ELSE
   i--;
```

is calculated as one statement, but it consists of four lines of codes. How to rationally calculate the distribution of mutants against *number of statements* is challenging, because it may involve complex control structures. As an illustration, we describe how distribution-aware mutation analysis can be implemented in terms of *lines of codes*. This implementation is also employed in our empirical study reported later.

III. Two Software Testing Techniques

We briefly describe two testing techniques which are to be examined in the subsequent empirical studies.

A. Random Testing

Random Testing (RT) [8] is one of the most widely and extensively employed black-box testing techniques in practice. RT selects test cases from the entire input domain randomly and independently, and it is easy to use; in the meanwhile, RT may be ineffective in some situation because it does not make use of information about software under test and the test history.

B. Dynamic Random Testing

In recent years, many efforts have been made to improve RT in different ways [5, 13]. Dynamic random testing (DRT) [5] is proposed to improve RT from the perspective of software cybernetics [4] and to select test cases making use of test history information. DRT combines RT and partitioning testing in order to take advantages of the two testing techniques. Assume the test suite *TS* is classified into *m* partitions denoted as $C_1, C_2, \ldots, C_m$, and each partition C_i has K_i distinct test cases, where i=1..m and $K_1+K_2+\ldots+K_m \geq K$ (K is the number of test cases in *TS*). A DRT algorithm is as follows [5]:

Step 1 Initialize the parameter ε, where $0<\varepsilon<1$.

Step 2 Select a partition C_i according to the given probability distribution $\{p_1, p_2, \ldots, p_n\}$, where p_i refers to the probability of C_i being selected, and $\sum_{i=1}^{m} p_i = 1$.

Step 3 Select a test case t from C_i randomly in accordance with the uniform distribution. That is, each distinct test case in C_i has an equal probability of $1/K_i$ being selected.

Step 4 Execute the selected test case t. If a failure is detected by t, increase the probability P_i of test cases in C_i being selected and at the same time decrease the probability P_j of test cases in other partitions as follows:

$$p_j = \begin{cases} p_j - \varepsilon/(m-1); & if\ j \neq i \wedge p_j \geq \varepsilon/(m-1) \\ 0; & if\ j \neq i \wedge p_j < \varepsilon/(m-1) \end{cases}, \text{ and}$$

$p_i = 1 - \sum_{j \neq i} p_j$, and remove the detected defect from the software under test;

Otherwise, take the following actions:

$$p_i = \begin{cases} p_i - \varepsilon; & if\ p_i \geq \varepsilon \\ 0; & if\ p_i < \varepsilon \end{cases}, \text{ and}$$

$$p_j = \begin{cases} p_j + \varepsilon/(m-1); & if\ j \neq i \wedge p_i \geq \varepsilon \\ p_j + p_i/(m-1); & if\ j \neq i \wedge p_i < \varepsilon \end{cases}.$$

Step 5 Check whether the test stopping criteria (such as the required reliability or the limited size of test suite) is satisfied. If it is satisfied, stop testing; otherwise, repeat the execution of Steps 2 to 4.

C. Testing Web Services

Web services must be trustworthy before they can be used. Testing is a major activity to assure that Web services can be trusted. However, the testing of Web services is more challenging than that of traditional software due to the unique features of Service Oriented Architecture. In particular, the lack of source code and the restricted control of services limit the testability of Web services.

In order to address these challenges, researchers have proposed various testing techniques for Web services. For example, in our previous work, we proposed a metamorphic relation-based approach to testing Web services without oracles [23]. Bartolini et al. [3] developed a tool called TAXI that generates test cases for Web services based on WSDL specifications. Bai et al. [2] proposed an ontology-based partition testing approach for Web services. Lenz et al. [14] applied model-driven approaches to the testing of Web services. Many other testing methods for Web services can be found in the literature, such as contract-based Web services testing [9], fault-based Web services testing [20], etc. Besides the above techniques for testing Web services, DRT also presents a good choice. More details are described in [24].

IV. Empirical Studies

In this section, we report empirical studies which have been conducted to investigate the impact of the mutation distribution on the effectiveness evaluation of testing techniques. In our experiments, we examined the two above-mentioned testing techniques on an electronic payment Web service. Random Testing is selected for evaluation because of its wide and easy exercise in practice, while DRT is selected for evaluation because the experimental environments have been developed in our previous studies [24], which greatly save our efforts in this study. As for DRT, we set probability adjusting parameter ε to 0.02 and 0.3 to make the evaluation more comprehensive. Two common metrics and six typical mutation distributions are considered.

A. Subject Program

An electronic payment system is implemented as a Web service and deployed in the Tomcat server [22]. The user and business data are stored in a MySQL database. The system offers several features, such as withdrawal, deposit, transfer, query. Among these features, we select the transfer feature for the evaluation because it is widely used in the electronic payment. The implementation consists of 154 lines of Java codes. For the commission fee charging schema, we refer to Agricultural Bank of China for the calculation rules as shown in Table I. Transfer types I-IV refer to the transfer between two accounts in the same bank and same city, in the same bank but different cities, in the same city but different banks, in different cities and different banks, respectively. These four transfer classes provide the basis to partition the input domain when using DRT.

TABLE I. COMMISSION FEE CALCULATION

	I	II	III	IV
Charge Percentage	0%	0.5%	0.5%	1%
Minimum Commission (¥)	0	1	1	1
Maximum Commission (¥)	0	50	50	50
Limit Per Transfer (¥)	50000	50000	50000	50000

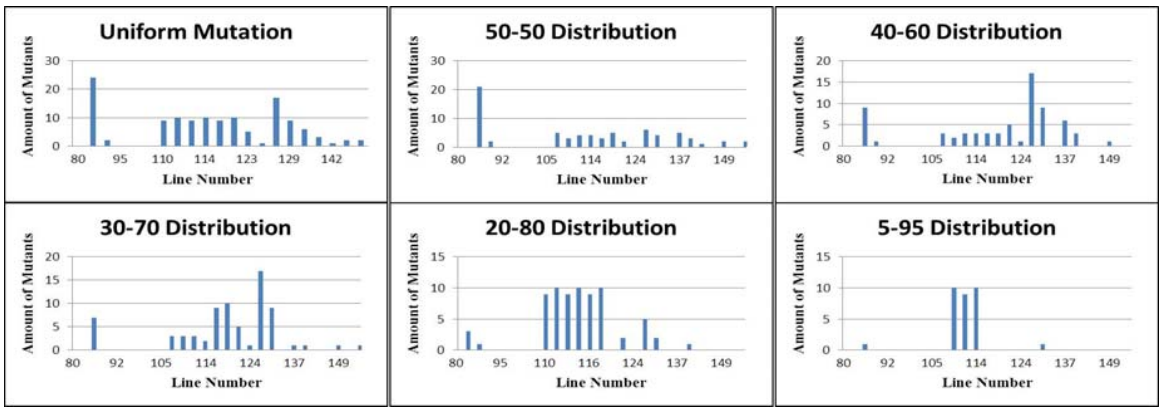

Figure 2. Uniform Mutation and Typical Distribution-aware Mutations

According to the specification of the transfer feature in Table I, we have a partition scheme shown in Table II. Here, *A* and *B* denote the sender and recipient account numbers for the transfer transaction, respectively. They consist of 10 digits; *P* denotes the transfer type. Its value ranges from 0 to 3, corresponding to type I to IV in Table I. Note that the transfer type can be deduced from *A* and *B*. For simplicity, we explicitly specify the type by *P*; *M* denotes the amount of a transfer transaction, ranging from 0 to 50000, inclusive. Since only the transfer type *P* and the transfer amount *M* affect the commission calculation, the partition scheme in Table II only makes partitions on both *P* and *M*. Note that *rate (p)* denotes commission ratio of *p* in Table I.

TABLE II. PARTITION SCHEME FOR ATM SYSTEM

Partition on Input Vectors	
A	N/A
B	N/A
P	0
	1
	2
	3
M	M<=0
	0<M≤1/rate(P)*
	1/rate(P)<M≤50/rate(P)
	50/rate(P)<M≤50000
	M>50000

B. Distribution-aware Mutants Generation

We employ the process described in Section II.B to generate distribution-aware mutants.

- Firstly, we seed faults into the implementation of the transfer feature by uniform mutation analysis. This is automatically done using *MuJava* [19], and a total of 139 mutants are derived. Among them, 10 mutants are equivalent ones and thus excluded from the experiments.
- Secondly, we label all the 129 mutants with line number of code where they occur and sort all mutants via their location values.
- Thirdly, we randomly select a set of the sorted mutants to satisfy the specific mutation distribution, denoted as *X-Y* distribution. In order to follow the convention, we would restrict *X* and *Y* to *X%+Y% =100%*. To generate a specific distribution mutation, we randomly select *X*% of consecutive lines of code as the clustered mutation area M_c, count the number N_1 of all mutants in this area M_c, and count the number N_2 of all mutants outside the area M_{nc}. If $N_1/(N_1+N_2)>Y\%$, we randomly remove mutants in M_c until the number N_1 of the remaining mutants in M_c takes the *Y%* of whole mutants, namely $N_1/(N_1+N_2)=Y\%$; Otherwise, we randomly remove mutants in M_{nc} until the number N_2 of the remaining mutants takes the *(100-Y)%* of whole mutants, namely $N_2/(N_1+N_2)=(100-Y)\%$.

In this study, we examine six classes of mutation distributions as shown in Table III. The *uniform* represents mutants generated by the uniform mutation analysis, *other distributions* represents mutants generated by the distribution-aware mutation analysis. Among them, *20-80* is the well-known Pareto distribution, which means that 80% mutants (or associated faults) distributed in 20% lines of codes. Figure 2 illustrates these mutation distributions with respect to the given subject program in Section IV.A.

TABLE III. NUMBERS OF MUTANTS WITH DIFFERENT DISTRIBUTIONS

Distribution	**Uniform**	**50-50**	**40-60**	**30-70**	**20-80**	**5-95**
Number of mutants	129	72	69	73	71	31

C. Effectiveness Metrics

F-measure and *T-measure* are two common metrics employed to evaluate effectiveness of testing techniques. *F-measure* is defined as the expected number of test cases required to detect the first fault [6]; *T-measure* is defined as the expected number of test cases required to detect all faults [5]. For both two metrics, the smaller the metric results are, the better the effectiveness is.

D. Results and Remarking Discussions

The effectiveness evaluation results of RT and two versions of DRT with respect to two metrics and six classes of mutation distributions are summarized in Tables IV and V, respectively. For all reported experimental results, we repeat the experiments for 50 times and use average values as evaluation results and the "*D*" column denotes the deviation over 50 trials. The "N_F" column in Table IV and the "N_T"

column in Table V denote the evaluation results of *F-Measure* and *T-Measure*, respectively.

TABLE IV. EVALUATION RESULTS WITH F-MEASURE

Distri-bution	*RT*		*DRT(ε=0.02)*		*DRT(ε=0.3)*	
	N_F	*D*	N_F	*D*	N_F	*D*
Uniform	8.77	0.32	7.41	0.17	6.82	0.14
50-50	10.34	0.24	9.77	0.19	9.18	0.29
40-60	7.73	0.53	6.92	0.44	6.03	0.29
30-70	7.16	0.51	5.75	0.36	4.94	0.35
20-80	6.15	0.36	4.77	0.29	3.51	0.22
05-95	6.22	1.18	4.27	0.65	2.74	0.19

From Table IV, we have the following observations. DRT runs fewer test cases to detect the first failure than RT in the context of non-uniform distributions. This means that DRT can detect the first failure faster than RT. The more evidently DRT outperforms RT, more clustered the mutation becomes. The settings of probability adjusting parameter ε have an evident impact the performance of DRT.

TABLE V. EVALUATION RESULTS WITH *T-MEASURE*

Distri-bution	*RT*		*DRT(ε=0.02)*		*DRT(ε=0.3)*	
	N_T	*D*	N_T	*D*	N_T	*D*
Uniform	187.26	10.13	198.68	9.23	197.68	11.9
50-50	179.02	12.62	179.7	10.65	177.15	16.61
40-60	169.14	24.04	169.82	31.43	163.92	30.35
30-70	168.1	27.49	162.2	43.12	155.7	50.85
20-80	161.4	24.6	146.52	45.93	128.96	102.68
05-95	115.08	158.39	90.08	139.51	61.46	142.33

From Table V, we have the following observations. DRT runs fewer test cases to detect all seeded faults than RT for most of non-uniform distributions. This means that DRT can save efforts to improve the quality of software under test than RT. The settings of probability adjusting parameter ε show an evident impact the performance of DRT.

We have the following common observations from Tables IV and V:

- When the two effectiveness metrics are considered, the mutation distribution affects the effectiveness evaluation of both RT and two versions of DRT.
- Even for the same testing technique, the effectiveness evaluation results obtained from the uniform mutation analysis are greatly different from those obtained from the distribution-aware mutation analysis, and such differences may vary nearly three times. For instance, DRT (ε=0.3) has an effectiveness of 2.74 when the 5-95 mutation distribution and *F-Measure* is employed, while its effectiveness becomes 6.82 when the uniform mutation analysis and *F-Measure* is employed (in Table IV).

Through the preliminary empirical studies, we discover that for a testing technique or even for different variants of the same testing technique, the effectiveness results evaluated using the traditional mutation analysis (uniform mutation analysis) are greatly different from those using distribution-aware mutation analysis. This discovery further indicates that the effectiveness reported using the traditional mutation analysis needs to be checked or realigned.

V. RELATED WORK

Various approaches have been investigated to reduce this computation expense. All these techniques can be divided into two types: reduction of the generated mutants and reduction of the execution cost. We next focus on several techniques on reducing the number of mutants, and compare them with the distribution-aware mutation analysis.

A commonly used approach to reducing the number of mutants is via the selective use of mutation operators [17]. The idea behind this approach is that if the mutation operators corresponding to the stronger fault classes are used, those corresponding to the weaker fault classes will not be necessary. This can significantly reduce the number of mutants by using only the mutation operators corresponding to the stronger fault classes. Kuhn [12] investigated the fault hierarchy of Boolean specifications, and the hierarchy can be used to guide the selection of mutation operators. Offutt et al.[17] further refined a subset of five "key" mutation operators. The distribution-aware mutation analysis is different from the above reduction approach because our approach reduces the number of mutants by deleting the mutants outside the clustered mutation area. Furthermore, our approach can be applied to the mutants generated and selected using the above mutant reduction approach. In this way, our approach can get a reduced and clustered set of mutants.

Another simple way to reduce the number of mutants is via sampling only a randomly selected subset of the mutants [25]. That is, given a fixed percentage number (denoted as x), x% mutants are randomly selected. Interestingly, the empirical results show that operator-based mutant selection is not superior to random mutant selection [26]. The above approach is similar to our approach in that mutants are randomly selected in both approaches. However, the random selection process is different in that in our approach mutants are randomly selected to follow a given distribution, while mutants are selected to satisfy a given percentage in the above approach.

Jia and Harman introduced a new form of mutation analysis called *high order mutation* [11]. Their idea is to find the rare but valuable higher order mutants that denote subtle faults. To reduce the number of mutants, a high order mutation can replace several first order mutants. Similar to Jia and Harman's work, Macario Polo et.al [21] used *second order mutants* to replace first order mutants. Since two first order mutants can be replaced by a single second order mutant, the total number of mutants can be reduced to half.

Mutant clustering [10] is a mutant reduction technique which first generates all first order mutants and then classifies them into different clusters. Each mutant in the same cluster is guaranteed to be killed by a similar set of test cases. Note that clustering mutants here are different from our "clustered" mutation in distribution-aware mutation analysis, although both distribution-aware mutation analysis and *mutant clustering* choose a subset of mutants from all first order mutants. Moreover, its motivations and selection criteria are totally different from ours.

VI. CONCLUSIONS AND FUTURE WORK

We have proposed a distribution-aware mutation analysis which is an extension to the existing mutation analysis techniques to deliver reliable effectiveness evaluation of testing techniques. Empirical studies have been conducted to investigate the impacts of the mutation distribution on the effectiveness evaluation of testing techniques. Results of our empirical studies suggest that the mutation distribution significantly affects evaluation results. This indicates that one must be aware of distribution of mutation when mutation analysis is used to measure the effectiveness of some testing techniques.

Distribution-aware mutation analysis proposed in this paper can be also used as a kind of effective mutant reduction technique. This is because that (1) mutants in distribution-aware mutation analysis are randomly selected and the resulting mutants are significantly fewer than the original mutants, particularly in case of densely clustered mutations; and (2) it has been validated that operator-based mutant selection is not superior to random mutant selection [26]. Furthermore, one can achieve a reduced and clustered set of mutants when distribution-aware mutation analysis is used in combination with operator-based mutant selection.

In our future work, we would like to include more testing techniques for empirical studies and report the realignments of their effectiveness.

ACKNOWLEDGMENT

This research is supported by the National Natural Science Foundation of China (Grant Nos. 60903003, 60973006), the Beijing Natural Science Foundation of China (Grant Nos. 4112037, 4112033), the Fundamental Research Funds for the Central Universities (Grant No. FRF-SD-12-015A), the Open Funds of the State Key Laboratory of Computer Science of Chinese Academy of Science (Grant No. SYSKF1105), and a discovery grant of the Australian Research Council (Grant No.DP0771733).

REFERENCES

[1] J. H. Andrews, L. C. Briand, Y. Labiche. "Is mutation an appropriate tool for testing experiments?", *Proceedings of 27th International Conference on Software Engineering (ICSE 2005)*, 2005, pp402-411.

[2] X. Bai, S. Lee, W.-T. Tsai, Y. Chen, "Ontology-based test modelling and partitioning testing of web services", *Proceedings of 6th International Conference on Web Services* (ICWS 2008), 2008, pp465-472.

[3]C. Bartolini, A. Bertolino, E. Marchetti, A. Polini. "WS-TAXI: A WSDL-based testing tool of web services", *Proceedings of the 2nd International Conference on Software Testing Verification and Validation (ICST2009)*, 2009, pp326-335.

[4] K.Y. Cai, J.W. Cangussu, R.A. DeCarlo, A.P. Mathur. "An Overview of Software Cybernetics", *Proceeding of the Eleventh Annual International Workshop on Software Technology and Engineering Practice (STEP'03)*, 2003, pp77-86.

[5] K.Y. Cai, H. Hu, C. Jiang, F. Ye. "Random Testing with Dynamically Updated Test Profile", *Proceedings of ISSRE 2009*.

[6] T.Y. Chen, R. Merkel. "An upper bound on software testing effectiveness". *ACM Transactions on Software Engineering and Methodology*, 2008, 17(3):1-27.

[7] R. A. DeMillo, R. J. Lipton, F. G. Sayward. "Hints on test data selection: Help for the practicing programmer", *IEEE Computer*,1978, 1(4):31-41.

[8] R. Hamlet. Random Testing. Encyclopedia of Software Engineering. John Wiley & Sons, New York, NY. 2002.

[9]R. Heckel, M. Lohmann. "Towards contract-based testing of web services", *Proceedings of the 2004 International Workshop on Test and Analysis of Component Based Systems (TACoS2004)*, 2004, pp145-456.

[10] Y. Jia, M. Harman. "An analysis and survey of the development of mutatio testing", *IEEE Transactions on Software Engineering*, 2011, 37(5): 649-678.

[11]Y. Jia, M. Harman. "Constructing Subtle Faults Using Higher Order Mutation Testing," *Proceedings of the 8th International Working Conference on Source Code Analysis and Manipulation (SCAM'08)*, 2008, pp.249–258.

[12]D. R. Kuhn. "Fault classes and error detection capability of specification-based testing", *ACM Transactions on Software Engineering and Methodology*, 1999, 8 (4): 411–424.

[13]F.-C. Kuo, K.Y. Sim, C. Sun, S.F. Tang, Z.Q. Zhou. "Enhanced Random Testing for Programs with High Dimensional Input Domains", *Proceedings of 19th International Conference on Software Engineering and Knowledge Engineering (SEKE 2007),* 2007, pp135-140.

[14]C. Lenz, J. Chimiak-Opoka, R. Breu, "Model driven testing of SOA-based software", *Proceedings of the Workshop on Software Engineering Methods for Service-Oriented Architecture (SEMSOA2007)*, 2007, pp99-110.

[15]L.J. Morell. "A theory of fault-based testing". *IEEE Transactions on Software Engineering*, 1990, 16(8): 844-857.

[16]G.J. Myers. "The art of software testing". 2nd ed., New York: John Wiley and Sons, 2004, 10-20.

[17]A.J. Offutt, A. Lee, G. Rothermel, R. Untch, and C. Zapf. "An experimental determination of sufficient mutation operators", *ACM Transactions on Software Engineering Methodology*, 1996, 5(2):99-118.

[18]J.Offutt, R.H. Untch. "Mutation 2000: Uniting the orthogonal", *Proceedings of mutation testing for the new century*, 2001, 24:34-44.

[19]J. Offutt, Y.S. Ma, Y.R. Kwon. "*An experimental mutation system for Java*", *ACM SIGSOFT Software Engineering Notes*, 2004, 29(5):1-4.

[20]J. Offutt, W. Xu. "Generating test cases for web services using data perturbation", *ACM SIGSOFT Software Engineering Notes*, 2004, 29(5):1-10.

[21]M. Polo, M. Piattini, G. Rodríguez. "Decreasing the cost of mutation testing with second-order mutants". *Software Testing, Verification and Reliability*, 2009,19(2): 111–131.

[22]C. Sun, G. Wang, B. Mu, H. Liu, Z.S. Wang, T.Y. Chen. "Metamorphic Testing for Web Services: Framework and a Case Study", *Proceedings of 9th International Conference on Web Services (ICWS 2011),* 2011, pp283-290.

[23]C. Sun, G. Wang, B. Mu, H. Liu, Z.S. Wang, T.Y. Chen. "A Metamorphic Relation-Based Approach to Testing Web Services without Oracles", *International Journal on Web Services Research*, 2012, 9(1):51-73.

[24]C. Sun, G. Wang, K.Y. Cai, T.Y. Chen. "Towards Dynamic Random Tesing for Web Services", *Proceedings of 36th International Computer Software and Applications Conference (COMPSAC 2012)*, to appear.

[25]W. E. Wong, A. P. Mathur. "Reducing the cost of mutation testing: An empirical study", *Journal of Systems and Software*, 1995, 31(3):185–196.

[26]L. Zhang, S.S. Hou, J. J. Hu, T. Xie, H. Mei. "Is Operator-Based Mutant Selection Superior to Random Mutant Selection?", *Proceedings of 32nd International Conference on Software Engineering (ICSE 2010)*, 2010, pp435-444.