Application of Mahalanobis-Taguchi Method and 0-1 Programming Method to Cost-Effective Regression Testing

Hirohisa Aman*, Yuta Tanaka†, Takashi Nakano†, Hideto Ogasawara† and Minoru Kawahara*
*Center for Information Technology, Ehime University, Matsuyama, Japan 790–8577
†Toshiba Corporation, Kawasaki, Japan 212–8585

***Abstract*—To enhance the cost effectiveness of regression testing, this paper proposes a method for prioritizing test cases. In general, a test case can be evaluated from various different points of view, therefore whether it is worth it to re-run should be discussed using multi criteria. This paper shows that the Mahalanobis-Taguchi (MT) method is a useful way to successfully integrate different evaluations of a test case. Moreover, this paper proposes to use the 0-1 programming method together with the MT method in order to take into account not only the priority of a test case but also its cost to run. The empirical study with 300 test cases for an industrial software system shows that the combination of the MT method and the 0-1 programming method is more cost-effective than other conventional methods.**

I. Introduction

A continuous release of quality products with enhanced functionalities is required for business success in the software industry. All of the software development organizations have endeavored to maintain a high quality of their products. To find as many hidden faults as possible prior to a release, software testings have been performed at various developmental phases [1], [2]. A system testing is one of crucial testing activities since it is performed to check the target system's successful behavior involving their functionalities, and to be the final check before a release. A thorough performance of a system test is desired for a successful new release. Moreover, test engineers must test not only upgraded functionalities but also other functionalities again because unexpected failures (regressions) might occur. That is to say, it is ideal to re-run all test cases whenever their product is upgraded.

However, many organizations have also tackled challenges other than quality assurance: reducing cost and shortening the time it takes to release. Since a system testing by hand requires much cost and time, it is hard to re-run all test cases (regression testing) at every upgrade because of realistic restrictions on the budget, resources and the time it takes to release. Thus, test engineers have to plan a cost-effective regression testing under such limitations. One of useful ways to increase the cost-effectiveness of regression testing is to prioritize test cases [3], [4], [5], [6], [7]. By assessing a test case's expectation to find a failure, we can preferentially select promising test cases for regression testing. While there have been some criteria to assess test cases from different points of view regarding their properties, there is a risk of overlooking regressions when we evaluate test cases by focusing only on a specific criterion. To reduce such a risk, we apply the Mahalanobis-Taguchi (MT) method [8] to reasonably combine different assessments for a better prioritization in this paper.

Even if we have a good way to evaluate priorities of test cases, we need to consider a reasonable way to select them as well. While it is simple to select test cases in decreasing order of priority, that is not optimal algorithm. Hence, we use the 0-1 programming method together with the MT method for a better regression testing in this paper.

The key contributions of this paper are: 1) to propose an application of the MT method to a better test case prioritization, and 2) to show the cost effectiveness of the combination of the MT method and the 0-1 programming method for a regression testing of an industrial software system.

II. Mahalanobis-Taguchi Method-Based Test Case Prioritization

A. Mahalanobis Distance

In general, an object such as a test case can have some different aspects. Thus, the features of an object can be observed as a multi-dimensional vector, and we would decide a similarity between two different objects by using the distance between them. For example, let us consider a two-dimensional example shown in Fig.1, where we have two groups A and B. Now, object X which is plotted at $(4.00, 3.00)$ seems to be closer to the center of group A than that of group B. Such a distance is referred to as the Euclid distance: let $\boldsymbol{x} = (x_1, \ldots, x_n)^T$ and $\boldsymbol{a} = (a_1, \ldots, a_n)^T$ be n-dimensional position vectors of object X and the center of group A, respectively[1]. The square of the Euclid distance between $\boldsymbol{x}$ and $\boldsymbol{a}$, $d_E(\boldsymbol{x}, \boldsymbol{a})$, is obtained by the following equation.

$$d_E(\boldsymbol{x}, \boldsymbol{a}) = (\boldsymbol{x} - \boldsymbol{a})^T(\boldsymbol{x} - \boldsymbol{a}) \ . \tag{1}$$

We can also compute $d_E(\boldsymbol{x}, \boldsymbol{b})$ by replacing $\boldsymbol{a}$ with $\boldsymbol{b}$ in the above equation. The Euclid distance is often used as a distance metric, but not always appropriate for deciding which group is closer to an object. In Fig.1, there is a significant difference between groups in their dispersions of data. While the data of group A is concentrated in a narrower region, the data of group B is scattered wider. By considering those dispersions, it would be more natural to classify object X into group B. There

[1]Superscript T signifies the transposition of vector.

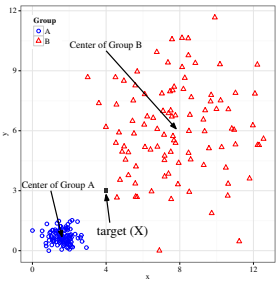


Fig. 1. An example of two-dimensional data groups.

is another notion of distance considering data-dispersion, the Mahalanobis distance. The square of the Mahalanobis distance $d_M(\boldsymbol{x}, \boldsymbol{a})$ is computed by the following equation.

$$d_M(\boldsymbol{x}, \boldsymbol{a}) = (\boldsymbol{x} - \boldsymbol{a})^T S_A^{-1} (\boldsymbol{x} - \boldsymbol{a}) \ , \tag{2}$$

where S_A is the variance-covariance matrix of data belong to group A, and S_A^{-1} is its inverse matrix. We can also obtain the square of the Mahalanobis distance from group B, $d_M(\boldsymbol{x}, \boldsymbol{b})$, by replacing $\boldsymbol{a}$ with $\boldsymbol{b}$ and S_A with S_B, respectively.

To give an intuitive interpretation, let us consider the case of a single dimension: $\boldsymbol{x} = x$ and $\boldsymbol{a} = a$, where non-bold-faced symbols x and a are scalar. By specializing Eq.(2) to the case of a single dimension, $d_M(x, a)$ is computed as follows.

$$d_M(x, a) = \frac{(x - a)^2}{\sigma_A^2} \ , \tag{3}$$

where σ_A^2 is the variance of the data in group A. In simplified terms, the Mahalanobis distance is a normalized distance which is the Euclid distance divided by the dispersion of data. Equation (2) is a generalized form of Eq.(3) to any dimension. In Fig.1, we have $\boldsymbol{x} = (4.00, 3.00)^T$, $\boldsymbol{a} = (1.61, 0.71)^T$, $\boldsymbol{b} = (7.81, 6.13)^T$,

$$S_A^{-1} = \begin{bmatrix} 4.001 & 0.502 \\ 0.502 & 8.058 \end{bmatrix}, \text{ and } S_B^{-1} = \begin{bmatrix} 0.197 & -0.005 \\ -0.005 & 0.192 \end{bmatrix}.$$

We can obtain $d_M(\boldsymbol{x}, \boldsymbol{a}) = 70.73$ and $d_M(\boldsymbol{x}, \boldsymbol{b}) = 4.62$, so $d_M(\boldsymbol{x}, \boldsymbol{a}) > d_M(\boldsymbol{x}, \boldsymbol{b})$: $\boldsymbol{x}$ is closer to $\boldsymbol{b}$ with the Mahalanobis distance. Therefore, we can classify object X into group B in Fig.1 when we adopt the Mahalanobis distance.

B. Mahalanobis-Taguchi Method

The Mahalanobis distance has been utilized in the statistical discriminant analysis to decide the group which an object X should be classified into. That notion can be applied to a quality control as well. Let us regard group A as the set of normal objects which are working properly. Then, we can assess the quality of an object X by using the distance from the center of A: if X is farther from group A, then X seems to be abnormal, i.e., something wrong in terms of its feature, and it would be better to be reviewed, repaired or replaced preferentially. The Mahalanobis distance of X from the center of group A can be a measure of X's degree of abnormality. This is a fundamental idea of the Mahalanobis-Taguchi (MT) method [8].

TABLE I
AN EXAMPLE OF TEST HISTORY

Test case	Version								
	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9
T_1	–	–	–	–	–	–	–	P	–
T_2	P	–	–	–	–	–	–	–	–
T_3	F	–	P	–	–	–	–	–	–
T_4	–	P	F	–	–	–	P	–	–
T_5	–	–	F	–	F	P	–	–	–

C. Test Case Prioritization

The aim of this study is to single out test cases which are likely to find failures (regressions). This kind of study is categorized as "test case prioritization" for regression testing and there have been many studies on it. Many of them are based on source code analyses [3], [4], [5]. Indeed, it is a promising way to associate a test case with a source code, and we can decide which test cases we should re-run after a source code modification by tracing their associations. However, there might be accidental associations causing unexpected failures at another functionality whose source code has not been changed. Moreover, there are also cases where test engineers are independent of programmers, and they may not get sufficient information in regard to the source code modifications [7]. Therefore, this paper studies a test case prioritization by focusing only on the test history which consists of test case IDs, versions of products and test results. This is not to say that we omit the information on "which functionalities are upgraded" at the current version. Of course, we should re-run the test cases related to those upgraded functionalities, and we naturally re-run test cases such that they failed at their last runs as well. After those re-runs are completed, our test case selection is started. Thus, under the condition that "all test cases passed at their last runs," we consider to run test cases again as many as possible in order to find unexpected faults which might be hidden in the system.

Table I is a simple example of test history which we can get. In this table, we have five test cases $T_1, \ldots, T_5$ and some of them were run for nine versions of the products $V_1, \ldots, V_9$. Labels "P" and "F" signify whether the corresponding test case run was "pass" or "fail," respectively; Label "–" denotes the corresponding test case was not run at the version.

We have empirically used the following two criteria (metrics) to evaluate test cases—GLC and FR.

1) Gap between the Last run version and the Current version (GLC):
   GLC(T) is the number of consecutive versions at which test case T was not run until the current version. For example, GLC(T_1) $= 1$ and GLC(T_2) $= 8$ in Table I.
2) Failure Rate (FR):
   FR(T) is the failure rate of test case T. That is to say, it is the number of versions failed by T, divided by the number of versions at which T was run. For example, FR(T_3) $= 1/2$ and FR(T_4) $= 1/3$ in Table I.

A test case having a greater GLC value has not been run for more versions after its last run. Such a test case has a higher

risk of overlooking regressions. Under our condition "all test cases passed at their last runs," GLC$= 0$ is the least value to re-run, because it is already re-run at the current version.

A test case having a higher FR value has a better track record for finding a failure in the past. Such a test case may test a part which is fault-prone in the product through maintenance, and we might expect a higher ability to find a regression.

On both GLC and FR, zero means that the corresponding test case has the least expectation in finding a failure. Thus, a (squared) Mahalanobis distance from 0 can be a measure of the test case's worth to re-run in terms of each metric's aspect. Table II shows their metric values and squared Mahalanobis distances when their variances are $\sigma^2_{\mathrm{GLC}} = 9.00$ and $\sigma^2_{\mathrm{FR}} = 0.0625$. In Table II, two different metrics may provide different evaluations for the same test case. Those differences are expected results because we successfully observe a test case from different points of view. Thus, a use of a single metric causes a higher risk of overlooking regressions. To reduce the risk, we propose to combine different metrics by the MT method. That is to say, we consider the Mahalanobis distance from $\mathbf{0} = (0, 0)^T$ to be a combined evaluation of a test case. Table II also shows the combined evaluations in the column annotated by "$d_{\mathrm{GLC\&FR}}$" where we computed those values with the following variance-covariance matrix S:

$$S = \begin{bmatrix} 9.00 & -0.225 \\ -0.225 & 0.0625 \end{bmatrix}, \text{ and } S^{-1} = \begin{bmatrix} 0.122 & 0.440 \\ 0.440 & 17.58 \end{bmatrix}.$$

III. 0-1 Programming Method-Based Test Case Selection

A. Greedy Method

The simplest way to select test cases is to single out test cases in decreasing order of evaluated value. This method is intuitive and easy-to-implement. For example, we can select five test cases shown in Table II by focusing on the evaluation "$d_{\mathrm{GLC\&FR}}$" in the following order: T_3, T_5, T_2, T_4 and T_1. Now we remember the cost to run a test case. The cost required to run different test cases is not always the same. Table III presents examples of costs. Moreover, we usually have an upper limit of effort which can go into our testing. Thus, we would have to give up running all test cases according to our limitations. If our upper limit is 20, we may select T_3 and T_5 in decreasing order of evaluation: the total value is 22.09.

The above method is simple, and has been known as "greedy method." However, the greedy method cannot always provide the optimal solution [9] in maximizing the total value. That is to say, the greedy method would not be a proper way to ensure a high cost-effectiveness in our regression testing.

B. 0-1 Programming Method

To get a higher expectation to find more failures, we aim to maximize the total value under our limitation of cost. This problem comes down to the "0-1 programming problem" which is formalized as follows.

TABLE II
METRIC VALUES AND SQUARED MAHALANOBIS DISTANCES OF TEST CASES SHOWN IN TABLE I

Test case	GLC	FR	d_{GLC}	d_{FR}	$d_{\mathrm{GLC\&FR}}$
T_1	1	0	0.11	0.00	0.12
T_2	8	0	7.11	0.00	7.81
T_3	6	1/2	4.00	4.00	11.42
T_4	2	1/3	0.44	1.78	3.03
T_5	3	2/3	1.00	7.11	10.67

TABLE III
AN EXAMPLE OF TEST CASE EVALUATION AND COST TO RUN

Test case	Evaluation ($d_{\mathrm{GLC\&FR}}$)	Cost
T_1	0.12	1
T_2	7.81	7
T_3	11.42	8
T_4	3.03	4
T_5	10.67	12

Given N test cases T_i $(i = 1, \ldots, N)$. Let P_i and C_i be the evaluated value of T_i and the cost to run T_i, respectively. Then,

$$\text{maximize} \quad \sum_{i=1}^{N} P_i \, x_i \ , \tag{4}$$

$$\text{subject to} \quad \sum_{i=1}^{N} C_i \, x_i \leq L, \text{ and } x_i \in \{\, 0, 1 \,\}, \tag{5}$$

where $x_i = 1$ means that we select T_i; $x_i = 0$ denotes that we do not select T_i. L signifies the upper limit of total cost which we can put into the regression testing.

□

For example, we can formulate the case shown in Table III with $L = 20$ as follows.

$$\begin{aligned} \text{maximize} \quad & 0.12x_1 + 7.81x_2 + 11.42x_3 + 3.03x_4 + 10.67x_5, \\ \text{subject to} \quad & 1x_1 + 7x_2 + 8x_3 + 4x_4 + 12x_5 \leq 20, \\ & \text{and } x_i \in \{\, 0, 1 \,\}. \end{aligned}$$

By solving this 0-1 programming problem, we obtain the optimal solution $(x_1, x_2, x_3, x_4, x_5) = (1, 1, 1, 1, 0)$ corresponding to when we select T_1, T_2, T_3 and T_4 with the total value as 22.38. Remember the total value by the greedy method was 22.09: the 0-1 programming method can always provide a solution which is the same as or superior to the greedy method. We can easily solve such 0-1 programming problems by using software tools such as lp_solve[2].

In reality, we also have to consider associations between test cases. That is to say, there may be a situation that: whenever we run test case T_i, it would be better to run T_j as well. Such an association can be considered by the simple inequality constraint: $x_i \leq x_j$. If $x_i = 1$ then we have $x_j \geq 1$ under the constraint. Since $x_j \in \{0, 1\}$, it leads to $x_j = 1$. Therefore, the above inequality constraint represents "$x_i = 1 \Longrightarrow x_j = 1$," so we have to select T_j whenever we select T_i.

[2] http://lpsolve.sourceforge.net/5.5/

TABLE IV
METHODS USED IN TEST CASE SELECTION

Symbol	Evaluation Method	Selecting Method
GLC-g	d_{GLC}	greedy
GLC-z	d_{GLC}	0-1 programing
FR-g	d_{FR}	greedy
FR-z	d_{FR}	0-1 programing
GLC-FR-g	$d_{\mathrm{GLC\&FR}}$	greedy
GLC-FR-z	$d_{\mathrm{GLC\&FR}}$	0-1 programing

TABLE V
NUMBER OF REGRESSION-FINDING TEST CASES BY METHOD

	Method					
Effort	GLC-g	GLC-z	FR-g	FR-z	GLC-FR-g	GLC-FR-z
60	20	20	2	2	2	19
90	20	20	2	2	22	22
120	20	20	2	2	22	22
150	20	20	2	21	22	22
180	20	20	2	22	22	22
210	20	20	2	22	22	22
240	20	20	2	22	22	22

IV. EMPIRICAL STUDY

A. Aim and Experimental Objects

The aim of this study is to examine the combination of the MT method and the 0-1 programming method in terms of the cost-effectiveness in the regression testing. Our experimental objects are 300 test cases for an industrial software system maintained by a certain company. We have the test history on their 13 versions. In order to clearly discuss the cost-effectiveness of the regression testing, we ran all test cases for the current version: the total running cost was 539 man-minutes, and 22 test cases found failures. The better test-case-selection method selects those 22 test cases with less effort.

B. Procedure

This empirical study is conducted in accordance with the following procedure.

1) Collect metric (GLC and FR) values and costs to run for all test cases. Let a test case's cost to be the actual man-minutes required for its last running.
2) Compute the evaluated values of test cases in terms of GLC, FR and their combination.
3) Extract associations between test cases.
   If test case T_j has always been run whenever T_i was run, the testing by T_i might be related with the one by T_j. In the authors' experience, sometimes both of their runnings would fail at the same subsequent version. Therefore, in order to reduce the risk of overlooking a regression, if we select T_i, then we also select T_j. To exclude coincidental co-runs, we focus only on test cases that have been run two or more times.
4) Compare six combinations of methods shown in Table IV in terms of the number of failure-finding test cases included, for 60, 90 120, 150, 180, 210 and 240 man-minutes as the upper limits of effort—these limits correspond to about 10%, 15%, 20%, ..., 45%. Use lp_solve version 5.5 as our 0-1 programming problem solver.

C. Results and Discussion

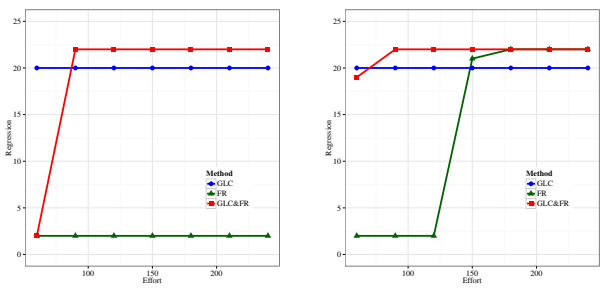
The variances and co-variance of metric values were $\sigma^2_{\mathrm{GLC}} = 8.85$, $\sigma^2_{\mathrm{FR}} = 0.021$ and $\sigma_{\mathrm{GLC,FR}} = -0.21$. Using these parameters, we obtained values of each test case: d_{GLC}, d_{FR} and $d_{\mathrm{GLC\&FR}}$. Because of space limitations, we will omit the table showing all of the evaluated values.

By checking a co-run of the test cases in the test history, we found 30 associations between them: we will also omit the list of them due to limitations of space. We used those 30 associations during our test case selection as follows.

- In the greedy method, we select test cases in decreasing order of their values. If "T_i associates with T_j" and we selected T_i, then we also select T_j. We iterated such a chained selection for all transitive relationships such that if T_i associates with T_j and T_j associates with T_k then we select T_k as well.
- In the 0-1 programming method, we add the inequality constraint corresponding to the association. If "T_i associates with T_j," we add constraint "$x_i \leq x_j$" to our formulation. Notice that possible transitive associations are also supported by satisfying all inequality constraints.

Table V shows the number of failure-finding test cases included in the set of selected test cases. The method having a larger number of those test cases with less effort is the more cost-effective method for regression testing. Notice that the maximum number of the target test cases is 22.

We compare ways of evaluating test cases by focusing only on the results by the greedy method (see Fig.2(a)) and (see Fig.2(b)) on the ones by the 0-1 programming method, respectively. For both of two methods, the best performances were made by "$d_{\mathrm{GLC\&FR}}$" except for the case of the least effort 60 man-minutes. While GLC contributed to select 20 of the target test cases with less effort, it failed to select the remaining two test cases and those two were picked up by FR rather than GLC. In this study, GLC seems to be useful to find failures but it misses some of them. On the other hand, although FR is not good at quickly finding failures, it recovers something missed by GLC. While they have different advantages and disadvantages, we succeeded in combining



(a) greedy (b) 0-1 programming

Fig. 2. Results of the greedy method and the 0-1 programming methods.

them to produce a more useful evaluation method by using the MT method.

We also compare the results from another perspective: "the greedy method vs. the 0-1 programming method." While they did not show significant differences in Table V, the 0-1 programming method always gives the results at the same with or superior to the greedy method for all settings.

From these results, we conclude that the combination of the MT method and the 0-1 programming method is useful in selecting test cases for a cost-effective regression testing.

D. Threats to Validity

Since the 0-1 programming problem is a NP-hard problem, it may take much time to solve with an increase in the number of test cases. However, we got the most solutions in less than one second with an average personal computer[3]. Although a part of the computation (FR-z with efforts of 180, 210 and 240) required a timeout processing with 30 seconds threshold, those suboptimal solutions succeeded in finding all failures in reality. Therefore, their computational complexity would not be a serious threat to validity in our empirical study.

While our empirical data consists of 300 test cases, we have many more test cases and their test histories. Those 300 test cases were selected from the past maintenance records without any bias, i.e., we adopted them without any intention. Thus, our data selection may not be a threat to validity.

V. Related Work

Various studies on the test case prioritization for regression testing [3], [4], [5] have been utilized the static code analysis techniques and the repository mining techniques. Their approaches are based on an association between a test case and a source program. While such an association can be powerful to decide which test cases should be re-run when a part of the source file was changed, we tackle the test case prioritization by using only test history which is easier to collect.

Fisher [10] focused on the 0-1 programming method to minimize the number of test cases for a regression testing. It proposed to select the minimal set of test cases such that they retest all elements affected by their code changes. While it is a significant previous work, our work focuses on a test history rather than code change information.

Kim et al. [6] proposed a method by focusing only on the test history. Their method computes a "selection probability" of a test case as its priority by using the exponential weighted moving average and the exponential smoothing. While that method is a useful related work matching our situation, they did not consider the testing cost. In this paper, we take into account not only the test case's priority but also its cost.

The literatures [11], [12] are our previous work. Although they proposed to apply the 0-1 programming method on the test case selection, they are based on a single metric. As mentioned above, a use of a single metric may lead to a higher risk of overlooking regressions. This paper proposes to successfully combine two or more metrics by using the MT method and to apply it with the 0-1 programming method.

[3]CPU: Intel Core i5, 3.2GHz; Memory: 16GB; OS: Linux 3.4.110

VI. Conclusion

We proposed to apply the Mahalanobis-Taguchi (MT) method into a test case prioritization in a regression testing. Since a test case has some different aspects in general, we expressed the feature of a test case as a multi-dimensional vector, and computed the Mahalanobis distance of the test case from the special test case which has the least expectation to find a failure. The computed distance can be an index of worth to re-run the test case. Moreover, in order to consider not only the priority of test cases but also the cost to run, we applied the 0-1 programming method to our test case selection.

We conducted an empirical study with 300 test cases for 13 versions of an industrial software product, and examined that the combination of the MT method and the 0-1 programming method shows the best performance of cost-effectiveness in regression testing. We can expect that the proposed method contributes to a more cost-effective regression testing and a more efficient quality management of software products.

Our future work include: 1) to apply our method to many other products from other domains to examine the generality of its cost-effectiveness; 2) to compare with other methods in order to discuss the superiority of our method.

Acknowledgment

The authors would like to thank the anonymous reviewers for their helpful comments.

References

[1] R. S. Pressman, *Software Engineering: a practitioner's approach*, 6th ed. Columbus, OH: McGraw-Hill, 2005.

[2] A. P. Mathur, *Foundations of Software Testing*. Delhi, India: Pearson Education, 2008.

[3] D. Jeffrey and N. Gupta, "Test case prioritization using relevant slices," in *Proc. 30th Annual Int'l Computer Softw. and Appl. Conf.*, vol. 1, Sept. 2006, pp. 411–420.

[4] M. Sherriff, M. Lake, and L. Williams, "Prioritization of regression tests using singular value decomposition with empirical change records," in *Proc. 18th Int'l Symp. Softw. Reliab. Eng.*, Nov. 2007, pp. 81–90.

[5] S. Mirarab and L. Tahvildari, "A prioritization approach for software test cases based on bayesian networks," in *Proc. 10th Int'l Conf. Fundamental Approaches to Softw. Eng.*, Mar. 2007, pp. 276–290.

[6] J.-M. Kim and A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments," in *Proc. 24th Int'l Conf. Softw. Eng.*, May 2002, pp. 119–129.

[7] M. J. Arafeen and H. Do, "Test case prioritization using requirements-based clustering," in *Proc. 6th Int'l Conf. Softw. Testing, Verification and Validation*, Mar. 2013, pp. 312–321.

[8] G. Taguchi, S. Chowdhury, and Y. Wu, *The Mahalanobis-Taguchi System*. New York, NY: McGraw-Hill, 2001.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 2009.

[10] K.F. Fischer, "A test case selection method for the validation of software maintenance modifications," in *Proc. Int'l Computer Softw. and Appl. Conf.*, Nov. 1977, pp. 421–426.

[11] H. Aman, M. Sasaki, K. Kureishi, and H. Ogasawara, "Application of the 0-1 programming model for cost-effective regression test," in *Proc. 37th Int'l Computer Softw. and Appl. Conf.*, July 2013, pp. 721–722.

[12] H. Aman, M. Sasaki, T. Nakano, H. Ogasawara, T. Sasaki, and M. Kawahara, "An efficient regression testing based on test case clustering and 0-1 programming model," *J. Japan Society for Softw. Sc. & Tech.*, vol. 32, no. 3, pp. 111–125, Aug. 2015 (in Japanese).