2014 14th International Conference on Quality Software

Bottom-up Integration Testing with the Technique of Metamorphic Testing

Xiaoyuan Xie [1,2], Jingxuan Tu [2], Tsong Yueh Chen [1], Baowen Xu [2]

[1] Department of Computer Science and Software Engineering, Swinburne University of Technology
{xxie, tychen}@swin.edu.au

[2] Department of Computer Science and Technology, Nanjing University
dg1233024@smail.nju.edu.cn, bwxu@nju.edu.cn

***Abstract*—In previous studies on the application of Metamorphic Testing (MT), testing was usually conducted at the system level directly, without any well-planned integration strategy. This could lead to a difficulty in constructing Metamorphic Relations (MRs), as well as tracking and debugging the faults. On the other hand, traditional integration testing requires extra cost. In this paper, we combine the bottom-up integration approach with MT to address all the above problems. We use the Feature Selection (FS) system to illustrate our method and a 2-phase MT is conducted. With this 2-phase MT method, we can ease the MR identification and provide a clear hierarchy to easily track and isolate the faults for a complex system. Moreover, since such an integration approach is logically achieved with MRs, it does not need to have those specific set-ups as required in traditional integration testing.**

***Keywords*—Metamorphic testing, bottom-up integration testing, feature selection, fault localization, debugging**

I. Introduction

Metamorphic testing (MT), proposed by Chen et al. [1] as an alleviation of oracle problem, has been successfully applied in many application domains [2], [3]. In MT, some necessary properties are required to verify the relations between multiple test inputs and their outputs, which is known as the "metamorphic relation" (MR). In previous applications of MT, MRs are usually constructed based on the necessary properties of the entire system and then MT is directly conducted on the whole system level only. In fact, it has been widely accepted that in traditional testing, instead of directly conducting the test on the entire system, we should first perform a step-by-step integration testing. Such a procedure can definitely facilitate tracking and isolating the fault in a clear hierarchy for a complex system [4]. Similarly, in MT, it is reasonable to adopt such an idea.

However, simply applying the traditional integration testing strategy involves many tasks, such as physically decomposing the system and integrating the sub-components (i.e. in terms of the source code), implementing the test drivers and test stubs, generating test cases for each individual sub-component, etc. These extra tasks could decrease the testing efficiency and increase the testing cost.

Therefore, in this paper, we propose an innovative bottom-up integration MT method, combining MT and traditional integration testing. Our method borrows the idea of bottom-up integration procedure but differs from the traditional idea in the following way: Instead of physically decomposing the whole system, testing and integrating the sub-components, it utilizes MRs that focus on components at various levels to achieve such goals logically. With our method, we can ease the MR identification and provide a clear hierarchy to track and isolate the faults for a complex system. Moreover, no extra task is needed. We use the filter Feature Selection (FS) algorithm as the motivation example to illustrate our method. Experimental studies are conducted to demonstrate its feasibility and effectiveness.

The contributions in this paper are summarized as follows.

1. Proposing a bottom-up integration MT method.
2. Demonstrating the feasibility of our method with FS algorithms.
3. Conducting an experimental study on the effectiveness of the proposed MRs, which provides supportive evidence to confirm our conjectures and justify the rationale of the bottom-up method.

II. Background

A. Metamorphic testing

In MT, the metamorphic relations (MR) that are identified from some necessary properties of the algorithm under test are known as the most essential part. Each MR binds multiple inputs and defines a relation between them and their corresponding outputs of the target application. Based on the MRs, some test cases obtained from any other testing strategies are used as source test cases, from which the corresponding follow-up test cases are generated. MT executes both the source and follow-up test cases, and then examines their output relation against the MR instead of verifying their correctness individually. A violation found in the MR indicates some faults in the program.

B. Feature Selection

Feature selection (FS) has been playing a critical role in many data-intensive areas. In this section, we will use the machine learning as the representative application domain to introduce FS. The training data set used in machine learning are usually represented by a list of features (denoted as $\mathbb{F}=<F_1, F_2, \ldots, F_n>$) and a set of class labels (denoted as $\mathbb{L}=\{L_1, L_2, \ldots, L_k\}$). Each sample s_i consists of an n-dimension vector $<f_1^i, f_2^i, \ldots, f_n^i>$ and a class label C_i, where

1550-6002/14 $31.00 © 2014 IEEE
DOI 10.1109/QSIC.2014.29

f_j^i is the value of feature F_j in sample s_i, and $C_i \in \mathbb{L}$ is the class of s_i. FS aims at reducing the dimensionality of the training sample set, removing irrelevant feature information, so as to increase the learning accuracy and improve result comprehensibility [5], [6]. There are three categories of FS algorithms, namely, embedded, wrapper and filter methods. In this study, we will focus on the filter method, which evaluates the relevance of features by studying the intrinsic property of the training data.

1. Search strategies.

Greedy Hill Climbing Search (denoted as GCS) is a local search algorithm. There are three directions in the search: forward selection, backward elimination and stepwise bi-directional search. Given a direction, this search strategy iteratively considers all the possible neighbours of the current starting feature subset in the specified directions, measures them with the adopted evaluation strategy, and then selects the best neighbour as a new starting subset. This search procedure iterates until there is no further improvement [7].

Best First Search (denoted as BFS) is a backtracking search strategy which explores search paths by changing the most promising feature subset with three directions. It differs from greedy hill climbing search in allowing the backtrack to a more promising candidate feature subset if all possible neighbour subsets result in no improvement. Normally, the stopping criterion involves limiting the times of backtracking to a more promising candidate feature subset.

In this paper we will focus on the forward direction for both greedy hill climbing search and best first search.

2. Evaluation strategies.

Let $S_F^{All}=\{F_1, F_2, \ldots, F_n\}$ denote the set containing all the n features and S_F^i denote any subset of S_F^{All}.

Pearson Correlation Based Evaluation (denoted as PCE) computes the merit of S_F^i as $\frac{h \times \overline{r_{fc}}}{\sqrt{h+h(h-1)\times \overline{r_{ff}}}}$, where $h=|S_F^i|$ (i.e. the size of the subset S_F^i), $\overline{r_{fc}}$ is the average feature-class correlation, and $\overline{r_{ff}}$ is the average feature-feature inter-correlation [8].

Consistency Based Evaluation (denoted as CSE). Suppose the number of all possible values for the t^{th} feature in feature subset S_F^i is n_t. Then, there are in total $P_{S_i}=\prod_{t=1}^{|S_F^i|} n_t$ possible combinations of values for feature subset S_F^i. Each combination is known as a *pattern* (denoted as p). Given a training set with m samples, a pattern p of feature subset S_F^i may appear for τ_p times in this training set, where $0 \leq \tau_p \leq m$. Note that these τ_p samples with pattern p may have different class labels. Suppose samples with p and of class L_{max} is the largest group, whose population is denoted as $\tau_p^{L_{max}}$. Then, the *inconsistency count* (denoted as I_p) for the current p is defined as $\tau_p - \tau_p^{L_{max}}$. For a feature subset S_F^i, CSE calculates its *inconsistency rate* (R_i), which is the sum of I_p for all patterns of S_F^i divided by the size m of the give training sample set. The formula is $R_i = \frac{\sum_{j=1}^{P_{S_i}} (\tau_{p_j} - \tau_{p_j}^{L_{max}})}{m}$. In the application of CSE, a candidate feature subset S_F^i is said to be consistent if its $R_i \geq \delta$, where δ is a user defined inconsistency rate threshold [9].

As a consequence, in this study, we will have four combinations as the investigated FS algorithms, namely, GCS_PCE, GCS_CSE, BFS_PCE and BFS_CSE.

III. Approach

A. A bottom-up MT method

Generally speaking, conducting MT with only the MRs from the necessary properties of the entire system may lead to the following two problems.

1. Violations observed in such MRs usually give very little clue about the locations of the faults, because these MRs are generated from the necessary properties of the whole entire system, the violations could be due to any component in the system.
2. Construction of such MRs may not be easy.

Therefore, a well-planned integration MT schedule is required. However, simply applying the traditional integration testing involves many tasks, including physically decomposing and integrating the source code, implementing the test drivers and test stubs, generating test cases for each sub-component, etc. Thus, in this study, we propose an innovative bottom-up integration MT method, which combines the integration approach with MT, to address all the above problems. In the following discussion, we will use filter FS algorithm to illustrate our method, since FS algorithm suffers from the oracle problem and has a clear hierarchy and dependency in its sub-components, which can serve as suitable candidate to depict our method more vividly.

By inspecting the filter FS system, it is not difficult to find that FS can be decomposed into a main component (namely, search component) and a sub-component (namely, evaluation component) that is called by the search component. Thus, there is a dependency between these two components: the search component passes particular feature subset as well as the values of these features in each training sample into the evaluation component, to obtain the corresponding merit, decide the next search step, and output the final feature subset as the result of the whole FS system. Figure 1 illustrates such a dependency between the search and the evaluation components.

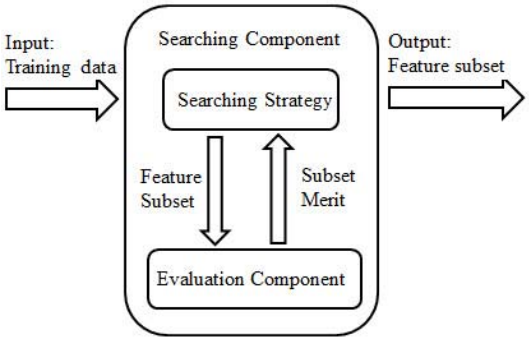

Fig. 1: Dependency between search and evaluation components

According to Figure 1, search component depends on the merit returned by the evaluation component. Thus, the evaluation component is at a lower level while the search component is at a higher level in this dependence relation. Therefore, it is not difficult to conclude that given a search strategy and an evaluation strategy, there are two important factors that affect the final result of FS: (i) the number and the locations of all feature subsets in the search space, which decide the next feature subset to be selected for evaluation according to the search strategy; (ii) the merit of each selected feature subset, returned by the evaluation component. According to all these observations, we can have the following important necessary property in FS algorithms.

Property 1 (Equivalent property)**.** *Given a search strategy and an evaluation strategy, if neither of the above two factors is changed, the result returned by the FS system should remain unchanged as well.*

The reason is obvious: from the perspective of the search component, nothing has been changed, hence the search path always remains unchanged as well. As a consequence, the final selected feature subset is the same. Based on this analysis, we now define two categories of MRs for the two components, as follows.

Category A, denoted as C_A, contains all MRs that satisfy (i) changes on the values of any feature subset in training samples **do not** affect the returned merit by the evaluation component; (ii) the search space (neither its size nor the position of each feature subset) of the search component **remains unchanged**. More specifically, the changes on C_A should be restricted to changing values on the existing samples, adding or deleting samples, but do not include any changes on the dimension or the order of all features in vector $\mathbb{F}$. Meanwhile these changes must always lead to an equivalent output relation for the evaluation component. Then, after Property 1, changes in these MRs will also result in an equivalent output relation for the whole FS algorithm.

Category B, denoted as C_B, contains MRs with any changes other than the ones in C_A. These MRs are based on the necessary properties of the entire FS system, that is, the combination of the particular evaluation and search components under investigation.

Apparently, C_A is restricted to a special type of MRs, with very strict conditions to be satisfied; while construction of MRs in C_B is less constrained such that the MRs in this category is more general. However, such a restriction on C_A has its special advantages. First, identifying MRs in C_A is generally much easier than identifying MRs in C_B, since the former one allows testers focus on one component and some equivalent relations only. And once such relations can be found, an MR is successfully identified. On the other hand, MRs in C_B is relatively difficult, because the proof of its necessity requires consideration of both the search and the evaluation algorithms. Secondly, MRs from C_A give definite information about the locations of the relevant faults, as shown in Proposition 1.

Proposition 1. *Any violation observed by MRs from C_A indicates that the relevant faults reside in the evaluation component.*

Proof: Violations revealed by MRs in C_A mean that after applying the changes required by C_A, there should be some difference between the source and follow-up outputs. Assume that these violations are due to the faults outside the evaluation component (i.e. in the search component). According to the definition of C_A, its changes on the source input do not affect the search process. Then, after Property 1, the source and follow-up test inputs always give the same output, even if these two outputs are both incorrect individually due to the fault in the search component. Therefore, this is contradictory to the observed violations. As a consequence, the violations revealed by C_A are only due to the faults in evaluation component. ■

On the other hand, MRs in C_B generally indicate that the faults are not restricted in any particular place of the whole system.

Proposition 2. *Any violation observed by MRs from C_B indicates faults in either the evaluation or the search component.*

With the above two categories of MRs, we now formally give our bottom-up metamorphic testing and debugging method, described as Algorithm 1.

Algorithm 1 Bottom-up MT and debugging

```
1: Construct MRs for C_A
2: Set V_A := violations observed in MT with C_A
3: while V_A ≠ ∅ do
4:    Fault fixing within evaluation component
5:    V_A := violations observed in Regression MT with C_A
6: end while
7: Construct MRs for C_B
8: Set V_B := violations observed in MT with C_B
9: while V_B ≠ ∅ do
10:   Fault fixing started with search component
11:   V_B := violations observed in Regression MT with C_B
12: end while
```

Algorithm 1 incorporates the idea of bottom-up integration testing, where the lowest level components are tested and debugged first, then, higher level components are integrated gradually for further testing. Such a procedure can facilitate tracking and isolating the fault in a clear hierarchy for a complex system. In our method, we have the similar procedure of a two-phase MT and debugging.

As discussed above, the evaluation component is at a lower level while the search component is at a higher level in this dependence relation. Thus, in our algorithm, we first test the evaluation component. According to Proposition 1, MRs from C_A are specialized in revealing faults in the evaluation components. Therefore, in the first phase, we construct MRs in C_A and then conduct MT with C_A. After observing the violations, we start to search and fix the faults which lead to these violations in the evaluation component, without considering the search component. After fixing all the faults revealed by C_A (that is, no more violation can be detected with regression testing), we continue with the second phase, that is to construct MRs in C_B and conduct MT with these new MRs. As stated in Proposition 2, violations observed in this phase may indicate the fault in any position of the whole system. However, since we have already carefully examined and eliminated some problems in evaluation component in the

first phase, thus, in the second phase, it is reasonable to start debugging from the search component.

It is obvious that the no matter how thorough the first phase of MT and debugging are, we can never guarantee a fault-free evaluation component. However, we still feel that it is reasonable to suggest debugging from the search component in the second phase, since at least we have more confidence about the evaluation component to some extent. Actually, this rationale has been further supported by our experimental study shown in Section V, where most faults in the evaluation component detected by MRs from C_B (second phase) can be covered by MRs from C_A (first phase). This promising result indicates that with the first phase of MT and debugging using MRs from C_A, we can clear some faults in the evaluation component, such that there will be very few faults left in the evaluation component for the second phase to reveal. Then in the second phase of MT and debugging using MRs from C_B, we have a strong reason to assume that the revealed faults by the observed violations are most likely in the search component.

To summarize, the adoption of the bottom-up MT and debugging method has successfully addressed the problems discussed above.

B. MRs used for four FS algorithms

In this section, we give the definitions of all MRs we have used for C_A and C_B, respectively. As a reminder, these MRs are not necessary properties of all the four algorithms. Thus, after giving the definitions, we will also indicate their necessity to each algorithm.

1. MRs for C_A.

MR1: Affine transformation on continuous features. We apply the same arbitrary affine transformation $f(x)=kx+b$, ($k\neq0$) to the values of all continuous features in each training sample. Then, the output feature subset should be the same.

MR2: Affine transformation on the continuous class. We apply the same arbitrary affine transformation $f(x)=kx+b$, ($k\neq0$) to the value of continuous class label in each training sample. Then, the output feature subset should be the same.

MR3: Permutation on training samples. We permute some of the samples in the training set. Then, the output feature subset should be the same.

MR4: Addition of uninformative samples. We add a sample into the training set, such that the value of each feature in this new sample is equal to the average value of the corresponding feature over all training samples. Then, the output feature subset should be the same.

MR5: Duplication of training samples with the original class label. We duplicate the all training samples for k times, with keeping their original class labels unchanged. Then, the output feature subset should be the same.

MR6: Duplication of a sample with assignment of a new class label to it. We duplicate any sample in the training set and assign a new class label that does not belong to $\mathbb{L}$. Then, the output feature subset should be the same.

2. MRs for C_B.

MR7: Addition of uninformative feature. We add an uninformative feature F_{new} at the end of source input feature vector $\mathbb{F}$, under which all samples have the same feature values. Then, the output feature subset should be the same.

MR8: Duplication of any unselected feature. Suppose feature F_i does not appear in the source output feature subset. We duplicate the whole column of F_i in the original training sample set. That is, F_i is added at the end of the original vector $\mathbb{F}$ as a new feature, under which the values of all samples are the same as the ones of the original F_i. Then, the output feature subset should be the same.

MR9: Duplication of any selected feature. Suppose feature F_i appears in the source output feature subset. We duplicate the whole column of F_i in the original training sample set. That is, F_i is added at the end of the original vector $\mathbb{F}$ as a new feature, under which the values of all samples are the same as the ones of the original F_i. Then, the output feature subset should be the same.

MR10: Deletion of any unselected feature. Suppose feature F_i does not appear in the source output feature subset. We delete the whole column of F_i in the original training sample set. That is, F_i and the values of all samples under it are deleted from the original vector $\mathbb{F}$ and the training sample set, respectively. Then, the output feature subset should be the same.

We have proved that MR1 to MR5 are necessary properties for both GCS_PCE and BFS_PCE; MR7, MR8 and MR10 are necessary properties for GCS_PCE in all cases but are only necessary properties for BFS_PCE when the search is set not to allow backtrack; MR3, MR5 and MR6 are necessary properties for both GCS_CSE and BFS_GCS; MR7 to MR10 are necessary properties for GCS_CSE in all cases but are only necessary properties for BFS_GCS when the search is set not to allow backtrack. Due to space limitations, we do not formally prove the necessity of all MRs for the four algorithms.

IV. Experiment set-up

In our study, we conduct mutation analysis to investigate the effectiveness of the proposed MRs and to justify the rationale of our bottom-up integration MT method.

A. Source test case generation

We utilize Weka as our testing object, since it is a very popular open-source machine learning framework that implements comprehensive algorithms [10]. The version used in our study is Weka 3.6.10.

As described in Section II-B, an input of FS system is a training sample set. In our experiment, we generate 100 source test cases for each FS system under testing as follows: (i) the number of features in each source training sample set is randomly assigned with value from 180 to 200; (ii) the size each source training sample set is randomly assigned from 20 to 50; (iii) the feature values and class label of each sample are randomly set, as either discrete or continuous to adapt to the corresponding algorithms. The reasons for adopting the randomly generated test cases rather than the real-life training samples, are two folds. First, random generation can provide large enough test suite. Secondly, it can avoid the bias from the domain knowledge.

B. Mutant generation

In this study, we use mutation analysis to evaluate the effectiveness of our proposed bottom-up MT method and MRs. The mutants are generated by MuClipse, an eclipse plugin of MuJava, with mutation operators at method level.

In our experiment, two categories of mutants are adopted.

- **Category A**, denoted as M_A, contains all mutants of evaluation components, that is the changes are in files for the merit evaluation procedure only.
- **Category B**, denoted as M_B, contains all mutants of search components, that is the changes are in files for the search procedure only.

For M_A, in each of the evaluation components, PCE and CSE, we randomly generate 30 mutants; and for M_B, in each of the search components GCS and BFS, we also randomly generate 30 mutants. Therefore, we have 30 mutants generated for both M_A and M_B in each algorithm. After excluding all mutants with runtime exception, for M_A, we obtained 26 mutants in GCS_PCE and BFS_PCE, and 25 mutants in GCS_CSE and BFS_CSE; for M_B, we have 26 mutants for GCS_PCE and GCS_CSE, and 24 mutants for BFS_PCE and BFS_CSE.

V. Empirical results and analysis

A. Killing rate in M_A

In this section, we will investigate the killing rate in M_A with MRs from C_A and C_B, respectively.

1. Effectiveness of C_A.

As stated in Proposition 1, MRs from C_A can only kill mutants related to evaluation components, that is mutants from M_A. In this section, we will first investigate how effective these MRs from C_A are, in terms of their killing rate among mutants from M_A. Figure 2 shows the results for programs GCS_CSE and GCS_PCE. Since the results of programs BFS_CSE and BFS_PCE are quite similar to those of GCS_CSE and GCS_PCE, respectively, we omit the figures of BFS_CSE and BFS_PCE, due to the space limitation.

In Figure 2, the bars except the last one indicate the killing rate of the corresponding MRs. We have found that in BFS_CSE and GCS_CSE, the three necessary MRs, MR3, MR5 and MR6 give killing rates as about 24%, 32% and 48%, respectively. In BFS_PCE and GCS_PCE, the killing rates of the five necessary MRs ranges from about 8% to 52%. MR3 performs the worst in all the four programs; while MR6 performs the best in BFS_CSE and GCS_CSE and MR1 performs the best in BFS_PCE and GCS_PCE.

Though an individual MR may not have a high killing rate, the collectively application of all the MRs can help to give a satisfactory effectiveness. The last bars in Figure 2 indicate the total killing rates of all MRs in C_A, which are roughly 52% for both GCS_CSE and BFS_CSE, and about 85% for both GCS_PCE and BFS_PCE.

2. Effectiveness of C_B.

In the investigation of MRs from C_B, let us first examine how many mutants these MRs can kill in M_A. The results

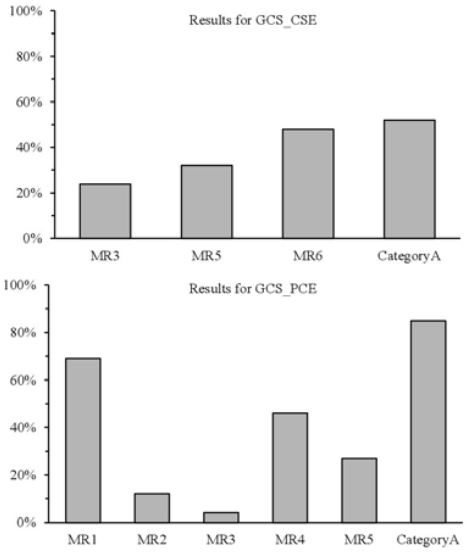

Fig. 2: Killing rate for C_A in M_A

for GCS_CSE and GCS_PCE are shown in Figure 3, and the figures for BFS_CSE and BFS_PCE are omitted due to their similarities to those for GCS_CSE and GCS_PCE, respectively.

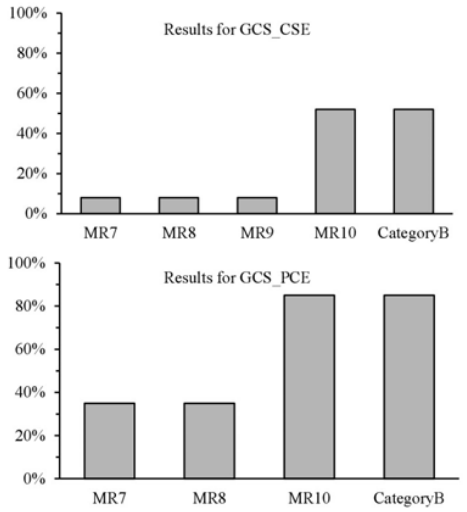

Fig. 3: Killing rate for C_B in M_A

We have found that in programs BFS_CSE and GCS_CSE, MR7, MR8 and MR9 give very low killing rates of about 8% while MR10 is the most effective one with killing rate of about 52%. And similarly in BFS_PCE and GCS_PCE, only MR10 has the highest killing rate around 85%.

The last bars in Figure 3 indicate the total killing rates of all MRs in C_B, which are 52% for both GCS_CSE and BFS_CSE, and 85% for both GCS_PCE and BFS_PCE. These results indicate that MRs in C_B can collectively deliver a satisfactory effectiveness in terms of M_A as well.

With the above results, we can find that C_A happens to kill the same number of mutants in the evaluation components as C_B. With Algorithm 1, if we assume the "perfect debugging", we can clear 52% to 85% faults in the evaluation component after the first phase of MT and debugging with MRs from C_A only. Besides, such debugging will be relatively easy since we know the locations of these faults must be in the evaluation component. However, if we do not follow the procedure in Algorithm 1 but directly use the general MRs in C_B as done in previous studies, though we may still reveal similar amount of faults, debugging with C_B will be much more difficult, since the fault could be at any place of the program.

More importantly, after looking into the mutants killed by these two categories of MRs, we have found stronger incentives for applying Algorithm 1. For each algorithm, let us denote the set of mutants from M_A collectively killed by all MRs from C_A and C_B as K_A and K_B, respectively. By inspecting K_A and K_B, we have found that K_A and K_B are not only have the same size, they have almost overlapped with each other. In GCS_PCE and BFS_PCE, K_A can cover about 95% of the mutants in K_B; while in BFS_CSE, GCS_CSE, K_A have covered K_B for about 85%. This information shows that the cleared faults in the first phase of MT and debugging with C_A have covered almost all the faults that C_B can reveal. Therefore, we have strong reason to assume that in the second phase of Algorithm 1, violations revealed by C_B are most likely due to the faults in the search component, since most faults in the evaluation component that can be revealed by C_B have already been fixed in the first phase of MT and debugging with C_A.

Additionally, we also find that in all the four algorithms, there exist mutants in K_A but not in K_B. This indicates that apart from the faults in the evaluation component that can be revealed by C_B, C_A can reveal some extra faults. By fixing these extra faults in the first phase of MT and debugging, we further improve the quality of the evaluation component and facilitate the following testing and debugging.

B. Killing rate in M_B

In this section, we will investigate the killing rate in M_B with MRs from C_B. Due to the page limit, we only give the result for GCS_CSE in Figure 4. We have found that all MRs in C_B are very effective in killing the mutants in M_B, with at least 60% killing rates. And the total killing rates of all MRs from C_B are 75%, 73%, 67% and 77% in BFS_CSE, GCS_CSE, BFS_PCE and GCS_PCE, respectively.

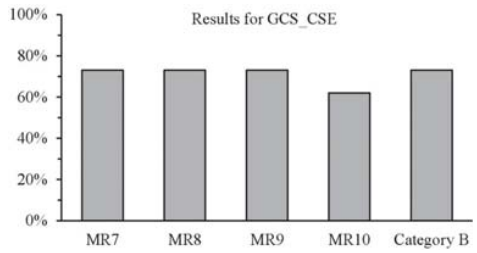

Fig. 4: Killing rate of C_B in M_B for GCS_CSE

VI. Conclusions

In this paper, we propose an integration MT method, which combines the bottom-up integration approach with MT. With this method, we can ease the MR identification and provide a clear hierarchy to track and isolate the faults for a complex system. Moreover, such an integration MT method is still physically conducted on the whole system, but using MRs based on the properties from different sub-component levels to achieve these goals logically. As a consequence, no extra task as required by traditional bottom-up integration testing, is needed. We use FS system to demonstrate the feasibility of our method. Experimental studies are conducted with a popular open source program, Weka. Experimental result has provided supportive evidence to justify the rationale of our bottom-up integration MT method.

Acknowledgement

This project is partially supported by Australian Research Council Discovery Project (DP120104773), Postgraduate Innovation Project of Jiangsu Province under Grant No. CXZZ13_0054, the National Key Basic Research and Development Program of China (2014CB340702) and the National Natural Science Foundation of China (91318301, 61321491, 61170071).

References

[1] 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, Tech. Rep. HKUST-CS98-01, 1998.

[2] T. Y. Chen, J. W. K. Ho, H. Liu, and X. Xie, "An innovative approach for testing bioinformatics programs using metamorphic testing," *BMC bioinformatics*, vol. 10, no. 1, pp. 24–35, 2009.

[3] X. Xie, J. W. K. Ho, C. Murphy, G. Kaiser, B. W. Xu, and T. Y. Chen, "Testing and validating machine learning classifiers by metamorphic testing," *Journal of Systems and Software*, vol. 84, no. 4, pp. 544–558, 2011.

[4] G. J. Myers and C. Sandler, *The art of software testing*. John Wiley & Sons, 2004.

[5] L. Molina, L. Belanche, and A. Nebot, "Feature selection algorithms: a survey and experimental evaluation," in *Proceedings of the IEEE International Conference on Data Mining*, Maebashi, Japan, Dec 2002, pp. 306–313.

[6] H. Liu, H. Motoda, and L. Yu, "A selective sampling approach to active feature selection," *Artificial Intelligence*, vol. 159, no. 1-2, pp. 49–74, 2004.

[7] M. A. Hall, "Correlation-based feature selection for machine learning," Ph.D. dissertation, Department of Computer Science, University of Waikato, Hamilton, New Zealand, 1999.

[8] ——, "Correlation-based feature selection for discrete and numeric class machine learning," in *Proceedings of the 7th International conference on Machine Learning*, Stanford, USA, June 2000, pp. 359–366.

[9] M. Dash and H. Liu, "Consistency-based search in feature selection," *Artificial Intelligence*, vol. 151, no. 1-2, pp. 155–176, 2003.

[10] I. H. Witten, E. Frank, and H. Mark, *Data mining: practical machine learning tools and techniques*, 3, Ed. Morgan Kaufmann, 2011.