# Eliminaton of Crucial Faults by a New Selective Testing Method

Masayuki Hirayama[†,‡], Tetsuya Yamamoto[†], Jiro Okayasu[†], Osamu Mizuno[‡], and Tohru Kikuno[‡]
[†] R&D Center, System Engineering Lab., TOSHIBA Corporation, Japan.
E-mail: masayuki.hirayama@toshiba.co.jp
[‡] Graduate School of Information Science and Technology, Osaka University, Japan.
Phone: +81-6-6850-6566 E-mail: {o-mizuno, kikuno}@ist.osaka-u.ac.jp

## Abstract

*Recent software systems contain a lot of functions to provide various services. According to this tendency, software testing becomes more difficult than before and cost of testing increases so much, since many test items are required. In this paper, we propose and discuss such a new selective software testing method that is constructed from previous testing method by simplifying testing specification.*

*We have presented, in the previous work, a selective testing method to perform highly efficient software testing. The selective testing method has introduced an idea of functional priority testing and generated test items according to their functional priorities. Important functions with high priorities are tested in detail, and functions with low priorities are tested less intensively. As a result, additional cost for generating testing instructions becomes relatively high. In this paper, in order to reduce its cost, we change the way of giving information with respect to priorities. The new method gives the priority only rather than generating testing instructions to each test item, which makes the testing method quite simple and results in cost reduction. Except for this change, the new method is essentially the same as the previous method. We applied this new method to actual development of software tool and evaluated its effectiveness. From the result of the application experiment, we confirmed that many crucial faults can be detected by using the proposed method.*

**Keywords:** *Selective testing, test case prioritization, functional testing.*

## 1 Introduction

Along with the expansion of the application field for software, the size and complexity of software have been expanding. Software with large size or high complexity provides various functions. These functions are basically black box functions for ordinary users [1], and the main concern of users is whether the behaviors and functions, which the target system provides, satisfy their purpose.

On the other hand, confirmation of functions' behavior from the user's viewpoint is performed in the system testing. Thus, in the software quality assurance, system testing has played an important role. However, the growing size of software exerts serious influence on software system testing [10]. That is, the amount of testing required also increases since large software contains many functions, operations and behaviors, all of which should be tested [1, 9, 14].

Generally, faults do not exist uniformly in software. Moreover, the influence on the user or the system varies among faults. From the viewpoint of efficiency in testing, focusing the portions which contain many crucial faults, and then testing these portions with high priorities, and finally eliminating these crucial faults are more effective. These portions with many crucial faults can be said the dangerous portions with respect to software quality [11].

Regarding the prioritizing test, various studies have been reported [2–6, 12, 13, 15].

Much research on test item reduction concerns regression test item reduction. Rothermel et al. proposed a test selection method using the information on differences in several versions' of the software and the information on testing coverage in the first version, and confirmed the effectiveness of their method [12, 13]. Related with this research, Harrold et al. proposed a cost effectiveness evaluation method in the regression testing using the Rothermel's method [6]. Elbaum et al. also proposed test item reduction in the regression testing and performed empirical evaluation of their method [3, 4]. They can control regression test items successfully by evaluating the statement coverage and functional coverage. On the other hand, Binkley et al. used the semantic difference of source codes and proposed a selection method of test items [2]. Wong et al. proposed a test item reduction technique based on the path coverage evaluation in the system execution [15].

Thus, test item reduction techniques in regression testing proposed so far are mainly based on the idea of using

difference information among the various software versions or the idea of using testing coverage information and faults information. However, these ideas are only available in the regression testing, and it is difficult to apply them to newly developing software. Moreover, even if the target software is the updated software, these techniques are not always effective for software which has a very complicated structure and high dependency between newly developing portions and existed portions. In addition, it is difficult to apply the technique based on the source code difference information, because functions to which a user assigns great importance, do not always coincidence with functions in source code modules.

In the previous work [7], we have presented a basic idea of the selective testing method to grasp dangerous portions in the target software in advance. The selective testing method has introduced the evaluation models (viewpoints) and metrics, which characterize each function in target software from probabilistic safety viewpoints. By using these models and metrics, dangerous portions in the target software are identified and are assigned high priorities in testing.

However, giving testing instructions for each test item was a little troublesome. In other words, additional cost needed for generating test instructions become relatively high. In order to reduce such additional effort, we reconsider the way of giving detail instruction for each test item, and revise our selective testing method. In this paper, we present a revised selective testing method, which gives a priority information only rather than generating testing instructions for each test item. Then, we also report an application example of the revised method, in which we discuss the effectiveness of the revised selective testing method.

In this paper, we will propose a simple procedure to prioritize the functions to be tested, while rather rigorous data is requested to use the conventional prioritizing method in [2–6, 12, 13, 15]. During the case study of application, we will confirm that the proposed method can be applied in the actual software testing.

# 2 Selective Testing Using Function Prioritizing

## 2.1 Outline of the proposed method

Generally speaking, the selective testing method tries to achieve the reliable detection of crucial faults that should be removed before shipment. The method enables the extraction of test items directly relating to these crucial faults for various functions that the software provides. Here, the "crucial faults" that should be removed are classified into the following two types:

**Type 1:** Fatal faults, which are related to important functions for users or systems, or

**Type 2:** Critical faults having severe influence on the system reliability or system safety.

In order to detect these crucial faults, we introduced the functional priority and fault severity to perform test item selection.

The proposed selective testing method consists of three phases: functional priority assignment, test specification designing, and test planning. Figure 1 shows the outline of the proposed method.

**Step 1** (Function priority assignment): The functions that the target system provides are first prioritized from various viewpoints. (Here we consider these kinds of viewpoints: system's viewpoint, user's viewpoint, and developer's viewpoint, which will be described in the next subsection.)

**Step 2** (Test specification construction for each function): The specifications for testing are constructed taking into account the functional priority. More detailed test items are constructed for the functions with higher testing priority. On the contrary, for functions with low priority, less severe test specifications are constructed. The priority of test items inherits the priority of the function. By changing the detailed level of test specifications, the quality and quantity of software testing can be controlled.

**Step 3** (Test planning): The sequential order of testing is also determined by taking into account the priority of each test item. Thus, we can effectively utilize the time for the testing and obtain the maximum reliability in a limited time.

In this paper, we try to simplify Step 2 only (that is, Step 1 and Step 3 are the same as in [7]). The difference will be described in subsection 3.2.

## 2.2 Viewpoints and metrics

In order to extract important functions, our method pays attention to three viewpoints: the system's viewpoint, user's viewpoint and developer's viewpoint.

**V1: System's viewpoint** Attention should be paid to functions which play important roles in the target system. These functions are likely to play various roles with their complex structure and large code size, or they are often developed by using existing codes having a similar function. For these reasons, this viewpoint can be evaluated by measuring the size or complexity and the reuse ratio of the software.

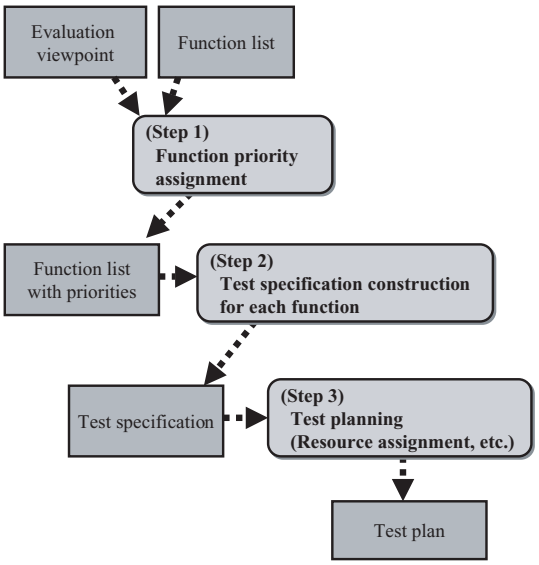


**Figure 1. Outline of selective testing method**

**Table 1. Metrics for evaluation**

| Viewpoints | Metrics |
|---|---|
| $V_1$: System's view | $M_{1,1}$: Size of function<br>$M_{1,2}$: Complexity of function<br>$M_{1,3}$: Ratio of newly developed<br>$M_{1,4}$: Ratio of reuse<br>$M_{1,5}$: Quality of reused code |
| $V_2$: User's view | $M_{2,1}$: Frequency of use<br>$M_{2,2}$: Complexity of use scenario<br>$M_{2,3}$: Impact of a function |
| $V_3$: Developer's view | $M_{3,1}$: Developer's skill<br>$M_{3,2}$: Experience of similar projects |

**V2: User's viewpoint** This is the viewpoint of those who use the target system. For this viewpoint, the complexity of the use scenario, the use frequency, and the fault impact of these functions are considered. In the case of software for cellular phones, for example, the functions of importance to young users are quite different from those important to elderly users. The user's viewpoint is a factor that is closely related to the faults which occur after the system is released, and to the impact of the faults.

**V3: Developer's viewpoint** Generally, engineers who developed the target system have good knowledge regarding its functional structure or functional relations. From their point of view, target functions playing core roles in the system are selected to be remarkable.

In the testing priority assignment, the test priority for each function is quantitatively evaluated, referring to the viewpoints mentioned above. Metrics ($M_{i,j}$) are prepared for each viewpoint ($V_i$) as shown in Table 1. Determining the metrics to be used is an important issue. In this study, we use the metrics that have been determined and used in the company, since it is the most familiar way to introduce the proposed testing method into the development field. For example, regarding a system viewpoint ($V_1$), five metrics are prepared: $M_{1,1}$ for functional size, $M_{1,2}$ for complexity of functions, $M_{1,3}$ for the ratio of newly developed portion, $M_{1,4}$ for reuse ratio, and $M_{1,5}$ for quality of reused code. Each metric has three levels of a value. This value is determined subjectively by the testing coordinator. For example, as for the functional size, the values 1, 2, and 3 are assigned according to the amount of functional specifications such as small, medium, and large, respectively.

When we consider the priority for function $K$, for example, the value of an evaluation metric $M_{i,j}$ is expressed by $X_{i,j}$, and the weight for this metrics is by $W_{i,j}$. The test priority for function $K$ ($P(K)$) is calculated by the following formula, where the weight for the metrics $W_{i,j}$ takes a value from 0 to 1.0.

$$P(K) = \sum W_{i,j} \times X_{i,j}$$

At this point, the function P(K) is a simple summation of weighted metrics, since all metrics used in this study have linear values. Thus, some other function would be appropriate for different metrics. Investigating the most appropriate function is remained as a future research.

In the case study to be described later, we assign 0 weights to the metrics in System's and Developer's viewpoints, since the testing in this case study is in the final acceptance test phase and thus the User's viewpoint is considered to be the most important.

## 3 Detailed Procedure for Steps 1, 2, and 3

As already pointed out, the current version of the selective testing method tries to reduce the testing cost by simplifying the procedure for Step 2.

In this section, we show objectives and operations of each step. To make each step more understandable, we will present concrete example in Section 4.

### 3.1 Priority assignment for functions (Step 1)

The priority of functions is determined by three steps as follows:

**Sub-step 1-1** Determination of the priority evaluating framework

In order to assign the priority of functions, viewpoints for the evaluation are determined in advance. Each

evaluating viewpoint has several metrics for the evaluation. In addition, the testing strategy that adjusts the weighting of the metrics is determined.

**Sub-step 1-2** Extraction of functions to be tested.

Based on the specification document of the software, the functions to be tested are extracted.

**Sub-step 1-3** Assignment of the priorities for functions.

The test priority of each function is determined, referring to the evaluated features by the weighted metrics. In this method, we consider three kinds of priorities: high, medium, and low. The evaluation is made from the viewpoints of use case information and features of the product.

Thus the sub-steps 1-2 and 1-3 of the proposed method extract functions to be tested and prioritized them.

### 3.2 Construction of test specifications (Step 2)

The step of test specification construction consists of two sub-steps as follows:

**Sub-step 2-1** Generation of test items.

In order to test each prioritized function, test items are specifically generated. For the functions with high priority, detailed test items are generated by using deviation analysis, taking into account operation patterns deviating from typical ones [8]. For the functions with low priority, only the test items for basic operation patterns are generated. Note that the priority of the test items inherits the priority of the function.

**Sub-step 2-2** Construction of testing specification.

For the test items generated at sub-step 2-1, the assigned priority is clearly described. In this method, we consider three kinds of priorities: high, medium, and low, again. Moreover, the test operators are proposed to test more carefully for the test items with "high" priority. On the other hand, for the test items with "low" priority, it is described that less efforts should be payed for the testing.

In Sub-step 2-2 of the current version, we assign the priority only for each test item. On the contrary, in the original version [7], we generated testing instructions for the test items. Additionally, depending on the priorities, we changed the levels of descriptions. That is, detailed instructions are assigned for high priority, and basic instructions are assigned for low priority.

### 3.3 Test planning (Step 3)

The sequential order of the testing and the testing operators are determined based on a consideration of the available time and resources. In order to assure high reliability in a limited time, crucial faults should be detected as early as possible.

In the test planning, we should determine the sequential order of testing by taking into account the priority of test items and the easiness of testing. For the test items with high priority, skilled operators are assigned. These results are summarized in the test plan document, and used for the testing operation.

## 4 Case Study: Construction of Test Specification

We applied our selective testing method to actual development of a software functional testing support tool(FTST) in a certain company. In this application, we generated test specifications by prioritizing the target functions. In this section, we show construction of test specification. As for evaluation of the resultant test specification, we will present case study in Section 5.

The FTST is a software tool that has a test priority assignment function, an automatic test priority score calculating function, a testing strategy setting function, and so on. The main specifications of this software are summarized as follows:

**Language:** C language.

**Size of software:** About 30000 LOC.

**Degree of reuse:** All newly developed.

### 4.1 Result of Step 1

As an application example, test items for the functions in the FTST are prioritized by using the above-mentioned selective testing method. For the priority assignment, we adopted the user's viewpoint ($V_2$) and evaluated three metrics: $M_{2,1}$, the frequency of use, $M_{2,2}$, the complexity of the use scenario, and $M_{2,3}$, the impact of a function. For example, as for the metric $M_{2,1}$, the frequency of use is measured by the questionnaire to the users by selecting the following three degrees, 1 day, 1 week, or 1 month. Moreover, we adopted an even-weight test strategy, i.e. the same weight is assigned for each metric. Then, the priority score is calculated by the following formula.

$$P(K) = 1.0 \times M_{2,1} + 1.0 \times M_{2,2} + 1.0 \times M_{2,3}$$

**Table 2. An example of function prioritizing (at Sub-step 1-3)**

| Function ID | Functions [Category, Operation] | Metrics [$M_{2,1}$, $M_{2,2}$, $M_{2,3}$] | Score | Priority |
|---|---|---|---|---|
| 4 | [ Main window , Select an item file, or a metrics DB ] | [ 3, 2, 2 ] | 7 | High |
| 28 | [ Evaluation view setting window , Explain the meanings of each evaluation ] | [ 1, 2, 2 ] | 5 | Medium |
| 30 | [ Evaluation view setting window , Cancel ] | [ 1, 1, 2 ] | 4 | Low |
| 44 | [ Score calculation , Show the calculated score ] | [ 1, 2, 1 ] | 4 | Low |
| 52 | [ Test strategy window , Save the test strategy in the DB ] | [ 3, 3, 1 ] | 7 | High |
| ... | ... | ... | ... | ... |

In the target software, we can consider that three properties, the frequency of use, the complexity of the use scenario, and the impact of a function have almost the equal affects for the testing. That is why we assigned the equal weights (= 1.0) in the above formula.

Table 2 shows main functions to be tested, their expected behaviors or operations and an example of the calculated priority for each function. For each function, the values of corresponding metrics are first presented. Then, the score is calculated by the formula $P(K)$. For this application experiment, functions having a calculated score of 6 or higher are regarded as high priority functions. Similarly, those having a score ranging from 4 to 6 are regarded as medium priority functions, and those having 4 or smaller as low priority functions. Since we performed the experiment in the actual software development, we have to choose the practical way for the selection of test items. That is, we have to increase the number of test items to be tested to prevent missing critical faults. That's why we expand the range of high priority functions twice as large as that of medium priority functions.

For example, the function ID-4 is a function that displays a main window of FTST. Concerning with this function, the frequency of use is 3, the complexity of use scenario is 2, and the impact of faults is 2. The score of this function is determined to be 7. From this score, this function is regarded as a high priority function.

### 4.2 Result of Step 2

In this case study, we focus on Sub-step 2-2 of the proposed testing method and investigate the effects of priority information to be assigned for each test item. We therefore did not control the way in which the test items were generated at Sub-step 2-1.

At Sub-step 2-1, the required operation for checking the behavior and expected results are described. Figure 2 shows an example of test specifications which is extracted from the analysis results for the behavior of the FTST. For this example, concerning the "Evaluation view setting window" of ID-28 function, a detailed test item "Select the target evaluation window" is generated. Concerning the function ID-52, 3 detailed test items "Press 'Save' button (ID-52-1),", "At reconfirm mode, press 'Save' button (ID-52-2)," and "At reconfirm mode, press 'Cancel' button (ID-52-3)," are generated.

At Sub-step 2-2, we describe the priority for each test item. The test item's priority is inherited from the function to be tested. For example, the test items ID-52-1, ID-52-2, and ID-52-3 have high priority since the function ID-52 has high priority. Similarly, the test item 28 has medium priority inherited from the function ID-28.

## 5 Case Study: Application to Software Testing

### 5.1 Purpose of experimental application

An example of test specification generation, taking into account functional priority based on the selective testing method, was shown in the previous section. In this section, a case study is conducted to apply this test specification document to actual development of the FTST. To be precise, the case study is to test the code for the FTST. For the case study, two independent testing teams, Team-A and Team-B, were organized in order to verify the effectiveness of the proposed method. Team-A used a test specification document prepared by the selective testing method described in Section 4 and Team-B used an ordinary test specification document prepared in a conventional manner.

**Functions with the priority**

| Function ID | Functions [Category, Operation] | Metrics [$M_{2,1}$, $M_{2,2}$, $M_{2,3}$] | Score | Priority |
|---|---|---|---|---|
| 28 | [ Evaluation view setting window , Explain the meanings of each evaluation ] | [ 1, 2, 2 ] | 5 | Medium |
| 52 | [ Test strategy window , Save the test strategy in the DB ] | [ 3, 3, 1 ] | 7 | High |

**Generated test items for functions**

| ID | Test items | Expected results | Priority |
|---|---|---|---|
| 28 | Select the target evaluation viewpoint. | Displays an explanation of the selected evaluation viewpoint. | Medium |
| 52-1 | Press "Save" button. | Test strategy is saved in the DB. | High |
| 52-2 | At the reconfirm mode, press "Save" button. | After showing the confirmation, test strategy is saved in the DB. | High |
| 52-3 | At the reconfirm mode, press "Cancel" button. | Back to "Test strategy setting window." | High |

**Figure 2. Generation of test items (at Sub-step 2-1)**

Our main objective of this case study is to gain the application example of the proposed method. In this experimental application, the evaluation was focused on the following two questions:

- By determining priority, are a larger number of fatal faults detected for the function with a higher priority?
- By determining priority, does any difference arise in the detection of critical faults related to reliability and safety?

## 5.2 Outline of the case study

As Figure 3 shows, two independent test teams were prepared. One team (Team-A) performed the proposed testing method and the other team (Team-B) performed the conventional test method. In the conventional test, all test items were tested in the order of the test specifications. Since the objective of this experiment is to evaluate prioritizing for functions, we control the experiment with the following conditions:

1. At first, we construct test specifications by the conventional method (we call it $P_B$.), and Team-B uses this specification $P_B$ in the experiment.
2. On the other hand, the test specification $P_A$ is constructed by assigning priorities for functions that are calculated according to the metrics. Team-A uses the test specification $P_A$ in the experiment.
3. In order to avoid the deviation due to the engineers skill, engineers who have almost the same technical experience and skill were assigned to both teams.

Here, we have to explain how to construct the test specification by the conventional method. In this method, firstly, the functional specification is carefully read and understood by the constructor of test specification. Next, the constractor generates the test specification so that the test items in the test specification can trace all possible operations described in the functional specification. As a result, the size of test specification becomes rather large. Furthermore, in the conventional method, test items are listed up in the order of functional specification document without any priority information.

The number of test items generated by the conventional method in the $P_B$ was 155. We should notice that no priority information is given to test items in the $P_B$. Thus, the test engineers in Team-B cannot use any priority information for test items.

By applying the proposed method, $P_A$ includes 155 test items with priority information: 53 items were classified as high priority, 72 items as medium priority, and 30 items as low priority. Concerning to the test execution, the test engineers on Team-A are only instructed the priority information (that is, the priority itself) for each test item.

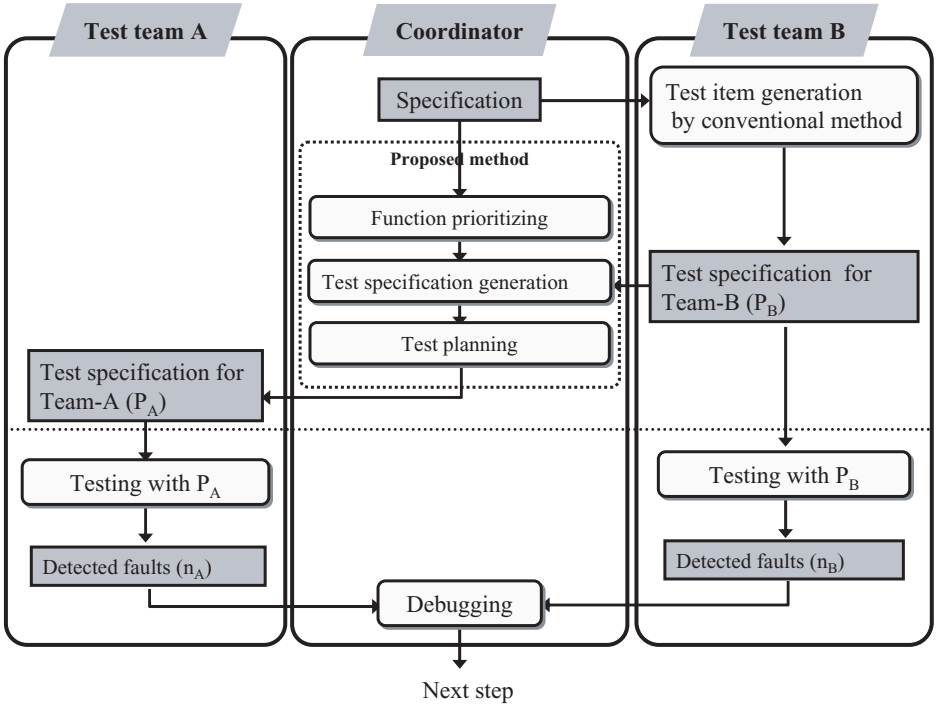


**Figure 3. Outline of experiment**

The period of time for each cycle was about one week, and the total duration of testing for these two teams were almost the same.

### 5.3 Results of experimental application

**Table 3. Result of experiment**

| **Proposed method (Team-A)** | | Seriousness of faults | | |
|---|---|---|---|---|
| | | Critical | Minor | Total |
| Priority of test items/ functions | High | 21 | 2 | 23 |
| | Medium | 12 | 2 | 14 |
| | Low | 1 | 1 | 2 |
| | Total | 34 | 5 | 39 |

| **Conventional method (Team-B)** | | Seriousness of faults | | |
|---|---|---|---|---|
| | | Critical | Minor | Total |
| Priority of test items/ functions | High | 7 | 6 | 13 |
| | Medium | 11 | 5 | 16 |
| | Low | 3 | 6 | 9 |
| | Total | 21 | 17 | 38 |

**(a) Fault detection**

The results obtained from the experimental application are shown in Table 3. The number of faults detected in the experimental application is shown in a matrix form expressed in terms of test priority and the seriousness of the faults. The "Priority of test items/functions" in the table means the priority for test items that is determined from the evaluated results of viewpoints and metrics. As explained in subsection 4.2, based on the calculated score, the priority is classified into three levels: high, medium, and low. The priority High is assigned for the score of 6.0 or higher, the priority Medium is for the score ranging from 4.0 to 6.0, and the priority Low is for the score of 4.0 or smaller. The "seriousness of faults" means the seriousness if the fault occurs. The seriousness is also classified into the following two levels:

**Critical:** Faults that seriously deteriorate the system reliability or product safety (For instance, a critical fault makes the system run out of control or halt).

**Minor:** Faults whose impacts are relatively small and many users can continue the operation without any troubleshooting (For instance, a simple indicator error is a minor fault).

In this case study, it is required for the testing to find as many critical faults as possible since it is in the final acceptance

testing phase. Under such a requirement, the seriousness of detected faults are classified into above two levels.

The numbers of faults detected by teams A and B, were 39 and 38, respectively, which are close to each other. However, by investigating in more detail, we can find advantages of the proposed method from the viewpoints of functions' priorities and seriousness of faults.

**(b) Detection of fatal faults**

Here, we consider the number of detected fatal faults, which are related to high priority functions. As shown in Table 3, 13 faults in total were detected by using the conventional method including all seriousness levels. On the other hand, 23 faults were detected in total by the selective method. For this experimental application, a function with high priority means that the frequency of its use is very high, faults related to the function cause critical damage if they occur, or the use scenario is complicated and numerous faults may easily be incorporated.

**(c) Detection of critical faults**

Regarding the detection of critical faults, the conventional method detected 21 critical faults, and the proposed method detected 34 critical faults. Focusing the functions with high priority, 21 critical faults were detected by using the proposed method, while the conventional method could detect only 7 critical faults in the functions with high priority. This fact shows that the proposed method has higher ability to detect "critical" faults in comparison with the conventional method. Furthermore, from the detailed investigations on the faults detected by both teams, it was shown that the most of the critical faults detected in the conventional method were also detected by the proposed method in this experiment. In other words, some critical faults may not be detected in the conventional method. We can say that the proposed method can perform more effective testing than the conventional method.

However, in this experiment, we cannot collect actual reliability data such as the number of faults detected after the shipment. The investigation using actual reliability data on the field is one of the most important future research.

With summarizing the results (a), (b), and (c) of the experimental application, we can confirm the followings:

1. The proposed testing method can effectively detect critical faults for the target software (as described in part (c) of subsection 5.3).

2. Fatal faults related to high priority functions are detected more intensively than in the case of conventional method (as described in part (b) of subsection 5.3).

Comparing between these two experiments' results, the new method in this experiment can detect the same or more faults than the original method [7].

## 6 Conclusion

In this paper, we have proposed a new testing method based on the priorities of functions in software. The priorities are calculated from the system's viewpoint, the user's viewpoint, and the developer's viewpoint. The proposed method consists of three successive phases: priority assignment, test specification construction, and test planning. Precisely speaking, the proposed method is designed from the previous testing method by adopting a simplified test specification.

In order to show the effectiveness of the proposed method, we performed an experimental application. In this application, functions were prioritized from the users' viewpoint, and for test items the priorities are calculated by test specification construction. It was shown that by adopting the proposed method, both critical and fatal faults were successfully detected.

In the future, we would like to further investigate the prioritizing method for functions or test items from other viewpoints, and try to confirm the effectiveness of the method by applying to the software testing of actual developments. Moreover, we must continue the study on the assigning the test resources and scheduling the test phase.


### Acknowledgment

Authors would like to thank anonymous reviewers who suggested many useful comments to brush up this paper.


## References


[1] B. Beizer. *Black-Box Testing*. John Wiley & Sons, New York, 1995.

[2] D. Binkley. Semantics guided regression test cost reduction. *IEEE Trans. on Software Engineering*, 23(8):498–516, Aug. 1997.

[3] S. Elbaum, A. G. Malishevsky, and G. Rothermel. Test case prioritization: A family of empirical studies. *IEEE Trans. on Software Engineering*, 28(2):159–182, Feb. 2002.

[4] S. Elbaum and G. Rothermel. Incorporating varying test costs and fault severities into test case prioritization. In *Proc. of 23rd International Conference on Software Engineering*, pages 329–338, 2001.

[5] R. Gupta, M. J. Harrold, and M. L. Soffa. An approach to regression testing using slicing. In *Proc. of International Conference on Software Mentenance*, pages 299–308, 1992.

[6] M. J. Harrold, D. Rosenblum, G. Rothermel, and E. Weyuker. Empirical studies of a prediction model for regression test selection. *IEEE Trans. on Software Engineering*, 27(3):248–263, Mar. 2001.

[7] M. Hirayama, T. Kishimoto, O. Mizuno, and T. Kikuno. A selective software testing method based on priorities assigned to function modules. In *Proc. of 2nd Asia-Pacific Conference on Quality Software*, pages 259–267, 2001.

[8] M. Hirayama, O. Mizuno, and T. Kikuno. Generating test items for checking illegal behavior in software testing. In *Proc. of 9th Asian Test Symposium*, pages 235–240, 2000.

[9] B. Marick. *The craft of software testing: subsystem testing including object-based and object-oriented testing*. Prentice-Hall, NJ, 1995.

[10] D. M. Marks. *Testing very big systems*. McGraw-Hill, 1992.

[11] K. Onoma, W. T. Tsai, M. Poonwala, and H. Suganuma. Regression testing in an industrial environment. *Communications of the ACM*, 41(5):81–86, 1988.

[12] G. Rothermel and M. J. Harrold. Empirical studies of a safe regression test selection technique. *IEEE Trans. on Software Engineering*, 24(6):401–419, June 1998.

[13] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold. Prioritizing test cases for regression testing. *IEEE Trans. on Software Engineering*, 27(10):929–948, Oct. 2001.

[14] I. Sommerville. *Software Engineering*. Addison-Wesley, MA, 4 edition, 1992.

[15] W. Wong, J. Horgan, S. London, and H. Agrawal. A study of effective regression testing in practice. In *Proc. of 8th International Symposium on Software Reliability*, pages 230–238, 1997.