ELIMINATION OF CRUCIAL FAULTS FOR EMBEDDED SOFTWARE USING FUNCTIONAL PRIORITY TESTING

Masayuki Hirayama[1], Tetsuya Yamamoto[1], Jiro Okayasu[1],
Osamu Mizuno[2], and Tohru Kikuno[2]

[1] R&D Center System Engineering Lab., TOSHIBA Corporation
1, Komukai Toshiba-cho, Saiwai-ku, Kawasaki 212-8582, Japan
[2] Department of Informatics and Mathematical Science,
Graduate School of Engineering Science, Osaka University
1-3 Machikaneyama, Toyonaka-shi, Osaka 560-8531, Japan

ABSTRACT

This paper discusses a new method for eliminating crucial faults in embedded software. Recent embedded software systems contain various functions or provide various services. Reflecting functional explosion of embedded software, the size and complexity of software increases so much. It is difficult to ensure their quality and to eliminate crucial faults by conventional software testing method because, in such large and complex software, too many test cases are required in order to cover all functions in a specification.

In this paper, we newly introduce an idea of functional priority testing and develop a new selective testing method. In this method, with prioritizing the functions in the target software, test items are selected according to their functional priorities. Important functions with high priorities are tested in detail, and functions with low priorities are tested less intensively. With using functional priorities, effective testing will be performed. The effectiveness of selective testing will be evaluated during experiments in actual software testing.

KEYWORDS

Software testing, functional test, prioritising for functions, crucial faults

1. PROBLEMS IN ELIMINATION OF CRUCIAL FAULTS

The growing size of software exerts serious influence on software system testing[7]. 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, 10]. Conventionally, discussion of software system testing has focused on techniques to increase coverage of source code and functions [3, 4, 6, 11]. But recently, in the development of large system, if an attempt is made to extract as many test items as possible so as to achieve coverage of all functions of the target system, the number of test items will become

unmanageably huge. There is simply insufficient time available to test all these test items. Therefore, software system testing, which focuses on coverage, has become inappropriate.

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 testing these portions with high priorities and eliminating these crucial faults are more effective. These portions with many crucial faults can be said the dangerous portions with respect to software quality.

In order to grasp these dangerous portions in the target software in advance, we introduce the evaluation models (viewpoints) and metrics, which feature each function in target software from probabilistic safety viewpoints. With using these models and metrics, dangerous portions in the target software are identified and are assigned high priorities in testing.

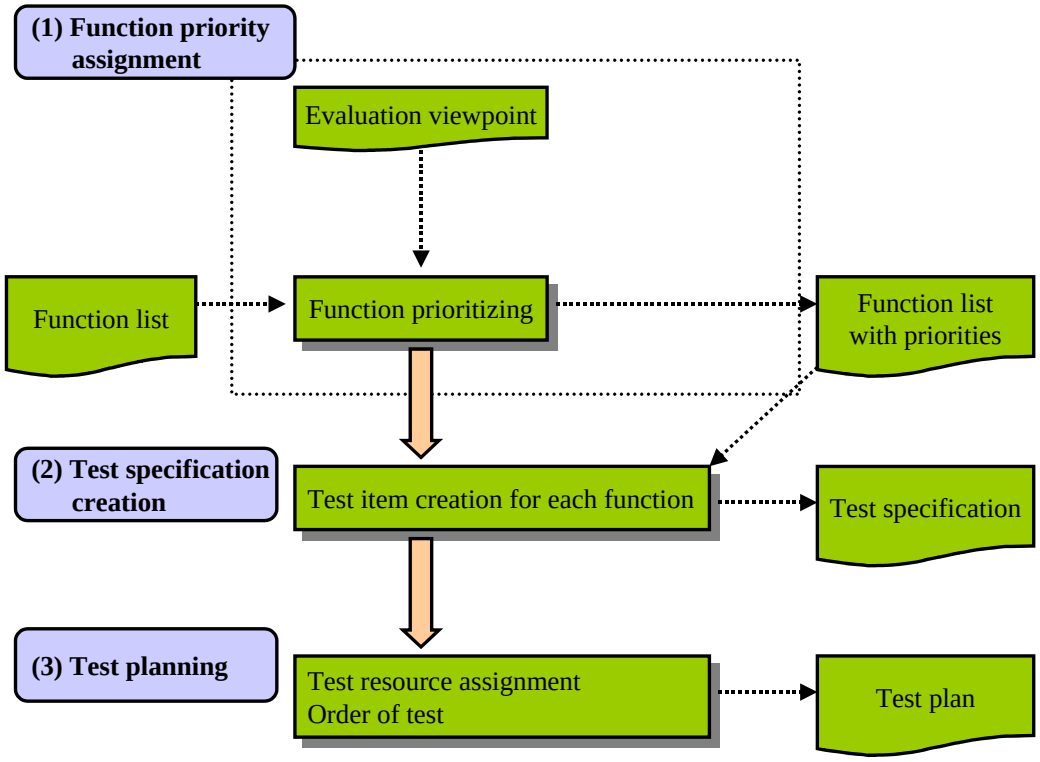

Figure 1: Outline of proposed method

2. SELECTIVE TESTING METHOD

In order to realize the above-mentioned concept, the selective software testing method is said to be useful, and various methods have been introduced so far[2, 8, 9]. However, the target of these methods are different from ours. So, we develop a new selective testing method. In the proposed method, we aim to detect and remove two classes of faults. The first class includes crucial faults that may cause serious influence to the system reliability or safety. The second class includes faults in the important functions in the software specifications.

As Figure 1 shows, the proposed testing method consists of three phases: function priority assignment, test specification creation, and test planning.

Function Priority Assignment

Test items for functions in the target system are prioritized by referring to use case analysis results. Concerning the assignment of priority, the viewpoints and metrics for test priority evaluation are

prepared. For example, system's view, user's view, developer's view, developer's skill, and development process are the viewpoints. With using these viewpoints, priorities are strictly assigned for functions.

Table 1 shows an example of assignment of priorities for functions in a specification. For the priority assignment, we adopted the user's viewpoint and evaluated three metrics: M_{21}, the frequency of use, M_{22}, the complexity of the use scenario, and M_{23}, the impact of faults. In Table 1, these metrics are evaluated intuitively by actual developers with values from 1 to 10. The priorities are calculated as means of these three metrics, and classified into three levels: high, medium, and low.

The test items for each function inherit the priority for the function. That is, test items for the same function have the same priority.

TABLE 1
PRIORITY ASSIGNMENT FOR FUNCTIONS

Function ID	Functions		Metrics			Priority
	Category	Operation	M_{21}	M_{22}	M_{23}	
9	Project Window	Refresh latest data	6	9	8	7.7 (High)
23	Pattern generation window	Show all items	5	7	6	6.0 (Medium)
58	Value setting dialog	Show the values	8	4	6	6.0 (Medium)
63	Test execution dialog	Execute test	6	3	7	5.3 (Low)
…	…	…	…	…	…	…

Test Specification Creation

Referring to the priority evaluation for functions, test items for each function are designed. For a function with high priority, the use case analysis is performed and deviation analysis from a normal case is done, and finally detailed test items are extracted [5]. On the contrary, for a function with low priority, only the normal operation indicated in the specifications is checked.

The determined priority for each function is indicated in the test specification clearly. The test operator may understand the importance of test items by referring the indicated priority. The quality of testing can be controlled.

Test Planning

Considering the test period and resources available, the execution order of generated test items is controlled and the assignment of system testers is determined. In the actual testing phase, we recommend two types of testing mode: The first is the quick testing mode that only focuses on test items with high priorities. The second is the full testing mode that evaluates all test items. With mixing these two testing modes effectively, safety of embedded software is ensured.

3. AN EXPERIMENTAL APPLICATION

Outline of the Experiment

In order to confirm the effect of new testing method, we applied the proposed method to some actual development projects. In this experiment, we mainly try to evaluate the effectiveness of the

priority assignment for functions and priority based testing method.

The target software of the experiment is a supporting for functional testing. The main features of this software are as follows:

- Language: C
- Software size: 30KLOC (all newly developed)

As Figure 2 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 evaluation of 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. The 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 for both teams.

During the test phase, each test execution consisted of 2 regression cycles. 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.

The number of test items in the P_B generated by the conventional method was 155. By applying the proposed method, spec-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 are only instructed the priority information for each test items, and the test engineers decide the detail patterns or test data for each test items.

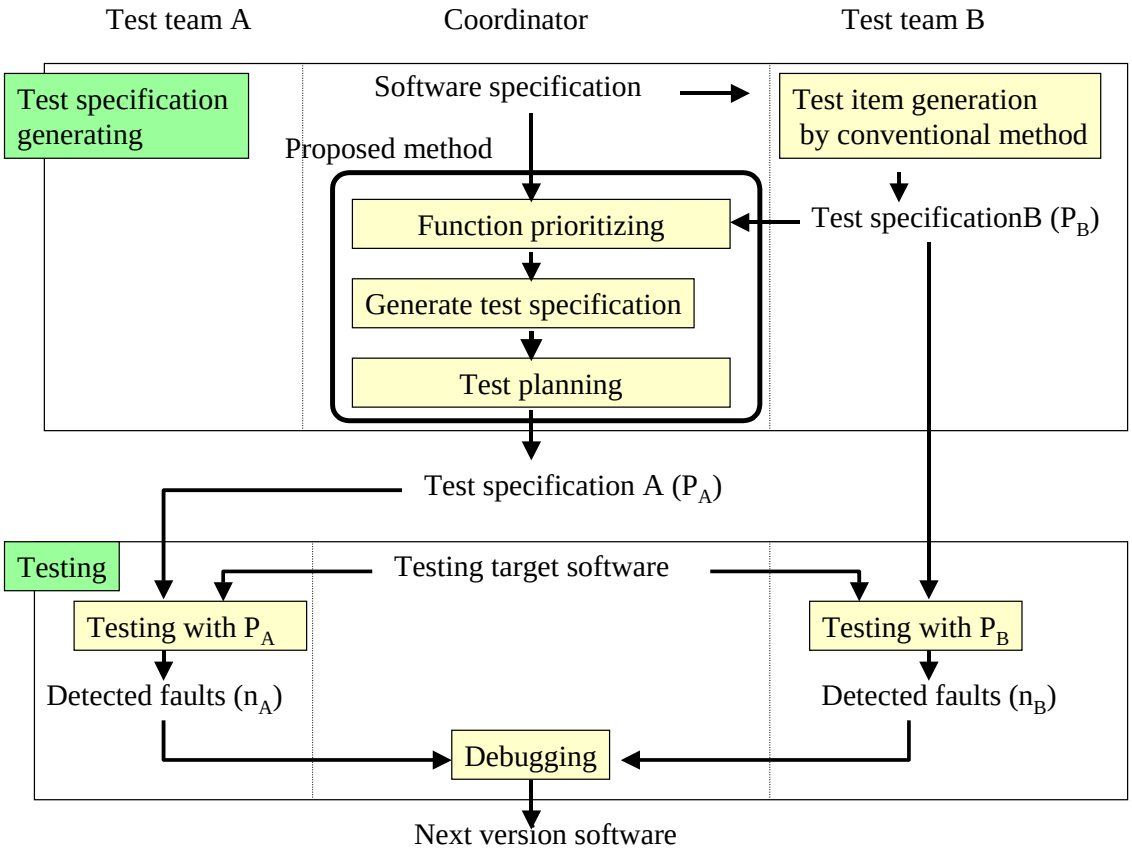

Figure 2: Outline of experiment

Results of Experiment

Table 2 shows the results of the experimental application. The number of faults detected in the experimental application is shown in a matrix expressed in terms of test priority and the seriousness of the faults. The "test priority" in Table 2 denotes the priorities for test items that is determined from the evaluated results of viewpoints and metrics. The "seriousness of faults" means the seriousness if the fault occurs. The seriousness is classified into the following two levels:

Serious: Faults that seriously deteriorate the system reliability or product safety, for instance, a fault makes the system run out of control or halt.

Trivial: Faults whose impacts are relatively small and many users can continue the operation without any troubleshooting, for instance, a fault like a simple indicator error.

TABLE 2
NUMBER OF TEST ITEMS THAT DETECT FAULTS

Proposed Method (Team-A)		Seriousness of faults		
		Serious	Trivial	Total
Priority of function	High	21	2	23
	Medium	12	2	14
	Low	1	1	2
	Total	34	5	39

Conventional Method (Team-B)		Seriousness of faults		
		Serious	Trivial	Total
Priority of function	High	7	6	13
	Medium	11	5	16
	Low	3	6	9
	Total	21	17	38

The numbers of detected faults for teams A and B, that is, n_A=39 and n_B=38, respectively, were almost the same. However, by investigating more detail, we can find advantage of the proposed method from the viewpoints of functions' priorities and seriousness of faults.

Detection of faults related to high priority functions

As far as the detected faults related to high priority, 13 faults in total were detected by using the conventional method. On the other hand, 23 faults were detected in total by the proposed method. In the experiment, a function with high priority is more frequently used and has high probability for causing serious damage if the faults fired. So faults for high priority functions for users must be detected and removed, without any exception, even if they are trivial. From this viewpoint, the fact that the proposed method detected more faults for high priority functions is meaningful.

Detection of serious faults

Regarding the detection of crucial faults, the conventional method detected 21 serious faults, and the proposed method detected 34 serious faults. Focusing the functions with high priority, 21 serious faults were detected by using the proposed method, while the conventional method could detect only 7 serious faults in the functions with high priority. This fact shows that the proposed method has higher ability to detect "serious" faults in comparison with the conventional method. Furthermore, most of

the serious faults detected in the conventional method were also detected by the proposed method in this experiment. We can say that the proposed method can perform more effective testing than the conventional method.

With summarizing the results of the experimental application, we can confirm the followings:
(1) The proposed testing method can effectively detect crucial faults for the target software.
(2) Faults related to high priority functions are detected more intensively than in the case of conventional method.

4. CONCLUSIONS

In this paper, we have proposed a new testing method based on the priorities of functions in software. The proposed method consists of three successive phases: priority assignment, test specification creation, and test planning. It is easy to apply in the actual software testing.

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 the priorities are indicated in the test specification for the testing team with the proposed method. By adopting the proposed method, both crucial and important faults were successfully detected. This indicates that the effectiveness of prioritizing strategy for test items from probabilistic safety viewpoint of software.

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 would like to continue the study on the weighting of evaluation metrics.

REFERENCES

[1] Beizer B. (1995) Black-box Testing: techniques for functional testing of software and systems. *John Wiley & Sons*.
[2] Cai K.Y. (1998) Software defect and operational profile modeling. *Kluwer Academic Publishers*.
[3] Cohen D.M., Dalal S.R., Parelius J., and Patton G.C. (1996) The combinatorial design approach to automatic test generation. *IEEE Software* **26:9**, 83–88.
[4] Coward P.D. (1988) A review of software testing. *Information and Software Technology*, **30:3,** 189–198.
[5] Hirayama M., Yamamoto T., Kishimoto T., Mizuno O., and Kikuno T. (2000) Generating test items for checking illegal behavior in software testing. *In Proc. of 9th Asian Test Symposium*, 235–240.
[6] Malaiya Y.K. (1995) Antirandom testing: Getting the most out of black-box testing. *In Proc. of 6th International Symposium on Software Reliability Engineering*, 86–95.
[7] Marks D.M. (1992) Testing Very Big Systems. *McGraw-Hill*.
[8] Musa J.D. (1996) Software-reliability-engineered testing. *IEEE Computer*, **29:11,** 61–68.
[9] Elbaum A.M.S. and Rothermel G. (2001) Incorporating varying test costs and fault severities into test case prioritization. *In Proc. of 23rd International Conference on Software Engineering*, 329–338.
[10] Sommerville I. (1992) Software Engineering, 4th edition. *Addison-Wesley*.
[11] Wong W.E., Horgan J.R., London S., and Mathur A.P. (1995) Effect of test set minimization on fault detection effectiveness. *In Proc. of 17th International Conference on Software Engineering*, 41–50.