Self-adaptive testing to determine sample size for flash memory solutions

Chul-Hoon Byun[1], Chang-Kyun Jeon[1], Taek Lee[1] and Hoh Peter In[1]
[1] Department of Computer Science & Engineering, Korea University
Anam-Dong, Seongbuk-Gu, Seoul 136-713, Korea
[e-mail: {damby0, curt_jeon, comtaek, hoh_in} @korea.ac.kr]
*Corresponding author: Hoh Peter In

Received April 6, 2014; revised May 7, 2014; accepted May 26, 2014; published June 29, 2014

Abstract

Embedded system testing, especially long-term reliability testing, of flash memory solutions such as embedded multi-media card, secure digital card and solid-state drive involves strategic decision making related to test sample size to achieve high test coverage. The test sample size is the number of flash memory devices used in a test. Earlier, there were physical limitations on the testing period and the number of test devices that could be used. Hence, decisions regarding the sample size depended on the experience of human testers owing to the absence of well-defined standards. Moreover, a lack of understanding of the importance of the sample size resulted in field defects due to unexpected user scenarios. In worst cases, users finally detected these defects after several years. In this paper, we propose that a large number of potential field defects can be detected if an adequately large test sample size is used to target weak features during long-term reliability testing of flash memory solutions. In general, a larger test sample size yields better results. However, owing to the limited availability of physical resources, there is a limit on the test sample size that can be used. In this paper, we address this problem by proposing a self-adaptive reliability testing scheme to decide the sample size for effective long-term reliability testing.

Keywords: Self-adaptive testing, embedded product testing, flash memory solutions

A preliminary version of this paper was presented at ICONI 2013 (The International Conference on Internet 2013) and was selected as an outstanding paper, December 12-16, Pattaya, Thailand. This version includes a concrete solution and additional experiment for self-adaptive testing at run-time. This research was supported by the Next-Generation Information Computing Development and Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planing(2012M3C4A7033345), and the Ministry of Education, Science and Technology(2012R1A1A2009021). And the R&BD Support Center of the Seoul Development Institute and the South Korean government (WR080951).

http://dx.doi.org/10.3837/tiis.2014.06.019

1. Introduction

Embedded system testing, especially long-term reliability testing, of flash memory solutions such as embedded multi-media card (eMMC), secure digital (SD) card, and solid-state drive (SSD) involves strategic decision making with regards to test sample size in order to achieve high test coverage. The test sample size is the number of flash memory devices used in the test. Reliability test cases based on various user scenarios of flash memory solutions are commonly executed. To generate various unexpected scenarios during long-term testing, an adequate number of test samples are required so that a significant number of deviations between the test samples can be obtained. The deviations between the test samples are often caused by differences of NAND memory or imperfect NAND memory verification. Therefore, the test result of a test case (TC) can vary considerably in different test samples. However, there is a natural bound on the test sample size owing to the limited availability of physical resources. Non-availability of test samples, physical test spaces, and short test periods are some of the limitations affecting long-term reliability testing in industrial manufacturing. Earlier, decisions regarding the sample size were dependent on the experience of human testers owing to the absence of well-defined standard.

Owing to a lack of understanding of the impact of the test sample size on the reliability test, manufacturers often choose to perform viable testing to meet delivery deadlines instead of focusing on the time-consuming optimal testing for zero-defects. However, a short testing window may fail to detect defects that may appear later in the user environment. Further, these defects cause manufacturers significant monetary loss, as they not only have to provide compensation to the clients for the defective products, but also re-test these products to identify the root cause of the problem. They may also need to explain why the defects were not detected during the original testing phase. To minimize the impact of the short testing window, a large number of test samples are required to significantly improve the likelihood of detecting such defects in the testing environment. These defects are often successfully reproduced by intensive testing with a large number of test samples. To overcome this problem, we need to develop a self-adaptive system for industrial embedded product testing.

In this paper, we present a self-adaptive testing scheme for deciding the test sample size that facilitates optimal testing in the restricted scenario as mentioned above. We state that a large number of potential defects can be detected if an adequate number of test samples are used to target weak features in reliability testing of flash memory solutions. For this purpose, we illustrate how to determine the weak features of the target product at run-time. However, if a large number of test samples are used as TCs that have no long-term defects, it results in a waste of resources. In this paper, we only treat the long-term reliability TCs of non-functional testing.

The ultimate objective of the proposed self-adaptive testing scheme is to determine time-to-market with zero-defects. Time-to-market is an important factor for successful testing in the industry. Time-consuming testing based on various user scenarios is directly correlated with the cost of testing. Moreover, it is considerably difficult and expensive to determine the root causes for field defects after the product has been launched than it is to find them during the developmental phase. In other words, time-to-market and zero-defects are highly correlated. In a certain sense, the zero-defects concept implies that no type of defect will occur in the field during the entire product life.

2. Related Work

2.1 Integration of Adaptive Concepts into the Static Prioritization Techniques

In recent years, many researchers have addressed TC prioritization (TCP) and TC selection techniques. Kavitha et al. proposed an algorithm that optimized TCP based on the three factors: customer priority, changes in requirement, and implementation complexity [1]. Zhang et al. also proposed a TCP algorithm based on various testing requirement priorities and TC costs [2]. In addition to requirement-based studies, there is a history-based TCP technique. The history-based TCP technique was proposed in [3] with an intention of improving test efficiency at the functional test level. Staats et al. proposed a test oracle-based technique [4] that considers the impact of test oracles on the effectiveness of testing.

In order to improve the accuracy of TCP, several researchers have integrated adaptive concepts into the static techniques. Jiang et al. introduced an adaptive random TC (ART) prioritization technique in [5]. Zhou et al. proposed a TC selection technique that uses coverage information for adaptive random testing [6]. Further, an adaptive regression testing strategy that focused on regression testing was proposed in [7]. They proposed a cost-effective technique for regression testing based on an organization's situation and testing environment.

2.2 Dynamic Test Case Prioritization for Embedded Product Solutions

TCP techniques face a unique challenge because they need to be executed in a manner that can dynamically modify at run-time based on the changes in the factors that drive TCP [8][9][10][11]. Kaushik et al. defined situations as events and proposed a paradigm called Dynamic Prioritization, which uses in-process events and the most up-to-date test suite to re-oreder test cases [12]. The factors that affect TCP are dynamically changed based on situations. However, previous studies have focused on events such as TC deletions and code changes and not on the issue addressed in our work.

In order to detect as many potential defects as possible in the reliability testing of embedded product solutions, it is necessary to make strategic decisions concerning the optimized sample size. However, to the best of our knowledge, existing studies have not investigated adaptive testing schemes for deciding the optimal test sample size in the domain of flash memory solutions. Improper test sample size can lead to potential field defects. With only a handful of available test samples, testers often inaccurately judge testing to be successful and complete. Our aim is to alleviate this problem by integrating dynamic TCP with the self-adaptive redistribution of test samples in order to achieve more intensive testing in the domain of flash memory storage solutions. This approach is not confined only to regression testing.

3. Self-Adaptive Testing Scheme

3.1 Challenges in Industry

In this section, we illustrate the challenge behind our self-adaptive testing scheme with an example of a typical scenario faced in an industrial product solution testing.

Suppose that there is a reliability TC for flash memory solutions that empirically requires three days with 30 test samples. Further, assume that there are insufficient test samples available for the test to be carried out in its entirety and it needs to be executed immediately to meet a deadline. In this scenario, the managers in charge of the testing procedure need to make a decision about initiating the testing effort. This is not a simple decision, as the limited physical

resources have to be judiciously allocated to the weak features of the target product to maximize the outcome of the testing procedure in the available short time period. Thus, the decision about the optimal test sample size for each TC is a complex problem in the flash storage manufacturing industry.

In addition, many product lines that are being tested are intended to be sold to two or more client enterprises. In this situation, it is considerably difficult to determine the optimal test sample size for each long-term TC because the available total test samples are limited and shared by all the planned test requests (TR) at the same time. To tackle this problem, testers usually make a decision to allocate more samples for initial testing than for regression testing. However, this could be wrong owing to the lack of an absolute standard for determining the weak features of the target product. In this case, if the sample size could be automatically redistributed based on a self-adaptive approach, it would be possible to obtain more accurate test results for each situation during run-time. Further, the number of test samples that are used during the system testing of flash memory solutions is closely related to the available human resources. Therefore, the self-adaptive approach, without human intervention, is worth considering.

Fig. 1 displays a case for cumulative non-functional defect rates of reliability testing of eMMC in industrial setting. From the graph, we can infer that the defect rates of the flash memory solutions follow a predictable pattern in the testing phase and the state of defects is focused on specific aspects in every release candidate.

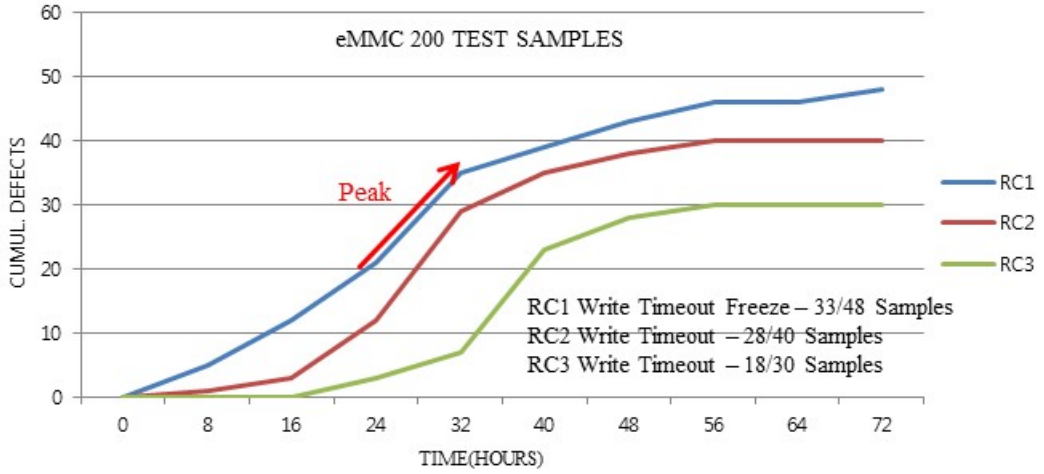

Fig. 1. eMMC defect rates in the testing phase

In **Fig. 1**, we note that the defect rates rise quickly and early in the cycle before it starts to fall at a slower rate over time after reaching a peak value. Further, we notice that the products still have a certain number of undetected potential defects even after the test deadline has been reached. It is impossible to detect all of the potential defects in a given testing period. Thus, we should target the weak features of the product such as write timeout freeze in RC1, with the maximum number of relevant TCs, as they are more probable to yield results during the testing phase. In other words, given that the test period is fixed, it is necessary to shift the peaks of the curves in **Fig. 1** towards left in order to find more long-term defects. We overcome this challenge by allocating proper test samples to the existing TCs. Thus, we claim that our self-adaptive approach concerning the test sample size solves the problems that are mentioned above.

3.2 Strategy for Detection of the Weak Features

We describe how to determine the weak features at run-time [9]. **Table 1(a)** lists the specific non-functional testing events from **Fig. 1**, obtained during run-time. The events composed of defect symptoms and defects. In particular, performance degradations, such as longer write and read times, are indicative of potential defects. Free block shortage of flash translation layer (FTL) causes a failure such as write timeout freeze. Similarly, read disturb of NAND memory causes a read timeout freeze. **Table 1(b)** lists the various root causes that could be linked to each testing event in a complex manner. Although the same event can occur in a TC, the root causes of the event can vary considerably in different test samples. Therefore, we need to focus more test samples on the TCs that detect such events.

Table 1. Reliability testing events and related root causes

(a) Reliability testing events.

EVENT	TESTING SITUATION
Defect Symptom	Write time increase. Read time increase.
Defect	Write timeout (Freeze). Read timeout (Freeze). Write data CRC fail. Read data CRC fail.

(b) Various root causes.

DEFECT SPOT	THE POSSIBLE ROOT CAUSE
NAND	Abnormal formation of NAND distribution.
Controller	Controller operation fail.
Firmware	Host interface layer fail. Flash translation layer fail. Flash interface layer fail. Flash recovery layer fail.

There are various user scenario-based TCs in a reliability test suite initially. The TCs in a test suite are expected to have similar test goals; however, it is not easy to accurately predict the test coverage of a long-term scenario-based TC. We notice that the defects history shows the exact goal and coverage of that TC. In that sense, testing is executed several times, and more accurate results can be obtained from the defects history. If a TC detects certain types of event, the TCs need to be categorized according to the type of event using the defects history database and the self-adaptive system intend to allocate more test samples to more vulnerable test suites. This intensive testing is similar to the Pareto rule. **Fig. 2** shows a change in a system when an event is detected at the run-time and the subsequent changes to the sample size due to adaptation. Test case#1 and test case#4 are associated with the type of first event. Here, an event is defined as the weak feature of the target product.

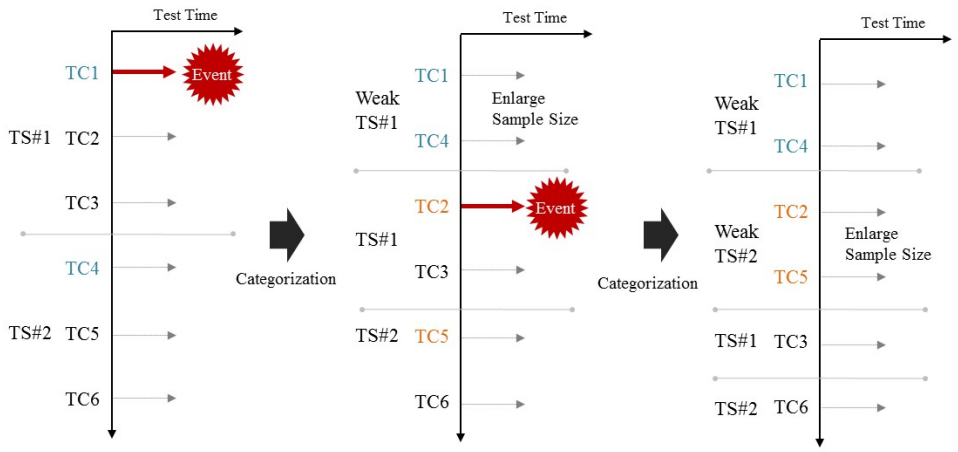

Fig. 2. System change associated with occurrence of events

3.3 Self-Adaptive Testing System Configuration

It becomes necessary to modify test sample size adaptively during run-time in embedded product testing. However, adaptive testing techniques, especially in the domain of flash memory solutions, have not been addressed in detail. In general, previous TCP studies are based on the static methodologies that rely on the requirements, histories, and code changes [1][2][3][4]. We assume such static-based TCPs as a precondition for our self-adaptive testing scheme. However, in this paper, test suite prioritization (TSP) is dynamically changed whenever there is an event. This ensures that we always focus on the weak features of the target product during a testing phase.

Fig. 3 illustrates the system configuration for self-adaptive testing scheme of long-term reliability testing of flash memory solutions. The central test server acts as a global controller, and locally, the test boards contain slots for the sample devices to be tested. The central test server communicates with each test board. Long-term reliability TCs are ready in the central test server. Human testers set their weight value manually based on the requirements, history, and code changes. The TCs are transmitted from the central test server to the test boards, and the test boards execute the testing procedures concurrently.

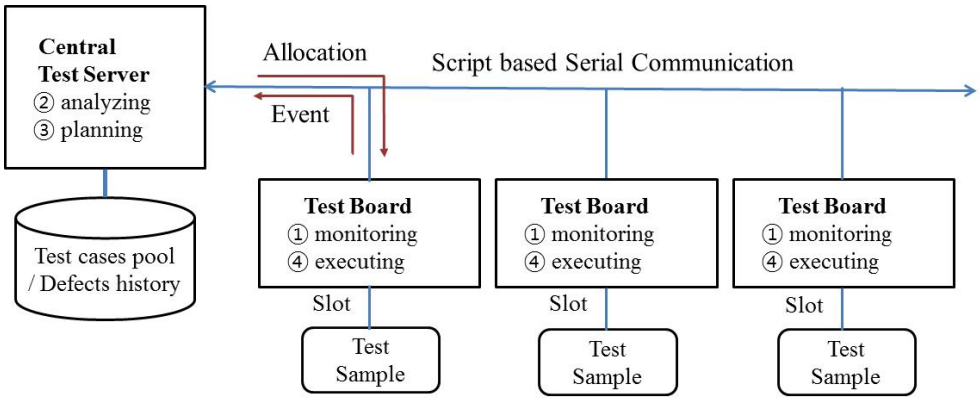

Fig. 3. Self-adaptive testing system configuration

Table 2(a). Role of the test board

PROCEDURE	ROLE OF THE TEST BOARD
Monitoring	● Observe and identify changes. - defects, defect symptoms. - response to the test server.
Executing	● Perform the entire testing as planned.

Table 2(b). Role of the central test server

PROCEDURE	ROLE OF THE TEST SERVER
Analyzing	● Analyze events. - identify individual test situations. ● Defects or defect symptoms. - identify the type of event. - find TCs involved with the event through defects history. ● No defect symptoms. - change input parameters of the TC.

Planning	● TC categorization and TSP. ● TC management. ● Sample size management - decide test sample size for each TC. - assign TCs to each test board.

The self-adaptive testing scheme follows the basic procedures of the self-adaptive software approach, MAPE-K model, which comprises monitoring, analyzing, planning, and executing as shown in **Table 2(a)** and **Table 2(b)** [8][13]. It works in a numerical order as shown in **Fig. 3**. In the monitoring phase, the test boards observe and identify new situations such as signs of defects. The signs of defects include defect symptoms and defects as shown in **Table 1(a)**. Subsequently, the central test server identifies the test results. If no defects are identified during the execution, i.e., if the test samples do not indicate any defects, then, the TC is restarted under a different set of input parameters from the decision table. Otherwise, the central test server schedules the testing for adaptation at the next phase. Modification to the new TSP and test sample size are decided at the planning phase. Finally, the central test server assigns the TCs to each test board and the testing is restarted.

3.4 Self-Adaptive Testing Procedure

The following phases illustrate the entire process. We assume that each of the TCs has been assigned a weight by human testers and the weight of the test suite is defined as the total weight of the TCs in that test suite. The weight is in agreement with the order of the priority. The scale of weights value may vary from target projects.

Phase 1: Monitoring test situations at run-time

In the first phase, test boards detect an event by checking the results of sub-functions in TCs at run-time. When an event occurs, the test board sends the event information to the central test server through script-based serial communication.

Phase 2: Analyzing the events

The type of event is identified in the central test server. Then, the server searches all the TCs related to that event from the defects history database. These TCs are called the weak TCs. **Fig. 4** shows the search process. When the read timeout event occurs in the 64 Gb Multilevel Cell (MLC) based product, three TCs are found. The search process is restricted to defects history database belonging to the same product line to improve accuracy and prevent restarting considerably many test slots.

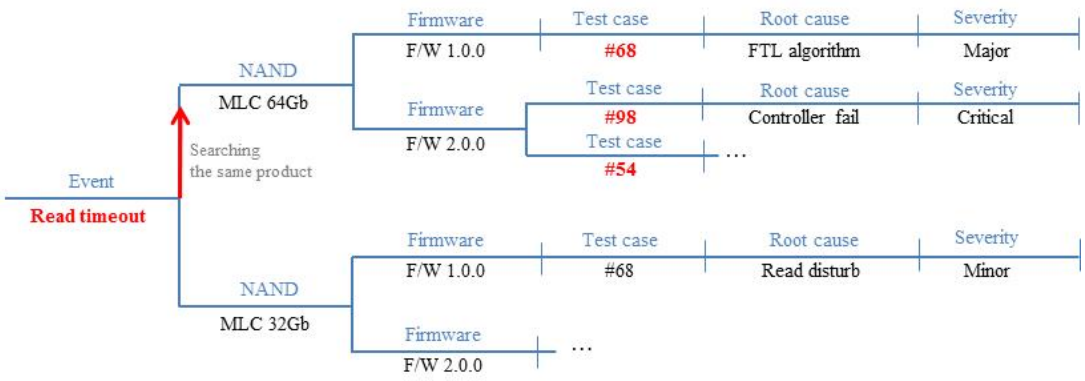

Fig. 4. TCs related to the event from defects history

Phase 3: Planning the test strategy

As mentioned in **Fig. 2**, the weak TCs are categorized and collected in the weak test suite. Then, predefined weights value of the test suites are rearranged according to the assigned categorization. The weak test suite, including all the TCs related to the event, has the highest weight and the sample size of the test suite is modified as a ratio of the weight in each test suite. The sample size of TCs in the test suite is determined by the initial weights of TCs. Finally, the test sample size is calculated according to the modified weight value and the test samples are redistributed to each test board.

Phase 4: Executing the test strategy

The central test server assigns TCs to the test boards as planned. Only few test slots are restarted. Many frequent changes are not desirable for test durability.

3.5 Self-Adaptive Testing Example

In this section, we show a simple example of the self-adaptive scheme. **Table 3(a)** shows the initial test settings. The number of test suites, TCs in each test suite, and the available test samples are as follows.

- Two test suites and available 210 test samples.
- Three TCs are in each test suite.
- Event occurs in test case#2.

The weight of the test suite is defined as the total weight of the test cases in that test suite. The test sample size of the individual test suite is proportional to the weight of this test suite. Moreover, we assume that the new event occurs at test case#2 in the monitoring phase. The event is a defect symptom such as write performance drop. The TCs related to the event are searched in the defects history database in the central test server. As a result, the weak test suite is composed of two TCs and it should have the highest priority. The weak test suite has double the weight value of the existing highest weight value of test suite #2 in **Table 3(b)** in order to prevent sample size to decrease than the existing sample size. Then, the test sample size of the individual test suites is modified as a ratio of the weight. The specific calculations for allocating sample size are as follows.

- Weight value of weak test suite = $12 \times 2 = 24$.
- Sample size of weak test suite = $210 \times (24/42) = 120$.
- Sample size of event occurring TC = $120 \times (8/14) \approx 68$.

Finally, all the TCs in weak test suite have increased the test sample size proportional to the weight of the TCs, which are concentrated on the weak features as shown in **Fig. 2**. **Table 3(b)** shows the adaptation result of this simple scenario. It can be observed that the weak test suite has increased the test sample size.

Table 3. Simple example of self-adaptive scheme

(a) Initial setting

Test Suite(TS)	Test Suite 1			Test Suite 2		
Test Case(TC)	TC1	TC2	TC3	TC4	TC5	TC6
TC Weight	2	8	4	6	10	2
TS Weight	14			18		
TS S/S	92			118		
TC S/S	13	53	26	39	66	13

(b) After adaptation

Test Suite(TS)	Weak Test Suite		Test Suite 1		Test Suite 2	
Test Case(TC)	TC2	TC4	TC1	TC3	TC5	TC6
TC Weight	8	6	2	4	10	2
TS Weight	24		6		12	
TS S/S	120		30		60	
TC S/S	68↑	51↑	10↓	20↓	50↓	10↓

3.6 Self-Adaptive Scheme Generalization

We denote the weight of the i[th] test suite as TSW_i and the weight of the k[th] TC in the test suite as TCW_k. We assume that the total number of TCs in the i[th] test suite is n and the total number of test suites is m. The weight of the test suite is computed as

$$TSW_i = \sum_{k=0}^{n} TCW_k \tag{1}$$

The test sample size rate of the i[th] test suite, $TSSR_i$, is computed as

$$TSSR_i = \frac{TSW_i}{\sum_{k=0}^{m} TSW_k} \tag{2}$$

From the sample size rate of the test suite, we define the test sample size for each test suite. The total sample size is denoted by SS_{total}. The sample size of the i[th] test suite, $TSSS_i$, is computed as

$$TSSS_i = TSSR_i \times SS_{total} \tag{3}$$

Thus, we can obtain the sample size of a TC in the test suite. The sample size of the j[th] TC, $TCSS_j$, is computed as

$$TCSS_j = TSSS_i \times \frac{TCW_j}{\sum_{k=0}^{n} TCW_k} \tag{4}$$

The self-adaptive testing scheme repeats the above steps automatically whenever a new event is encountered.

4. Comparative Experiment

4.1 Background

The simulation-based experiment for feasibility involves comparing three types of processes for distributing test samples. In this study, we assume that there are 50 TCs, 10 test suites, and 500 test samples. Because we assume that the test period is 72 h, each TC has to be executed during the same period. Next, three types of processes are used, each with different execution methods. These are concurrent, sequential, and self-adaptive methods, respectively. We prove that it is effective to concentrate TCs on more important features at run-time without human intervention.

4.2 Existing Methods

The concurrent execution method selected all of the TCs and executed them at the same time. Each TC had a fixed set of test samples based on the weight of the TC. The predetermined sample size was not changed even when the events were detected. The sample size and TSP were not altered during this process flow. If a defect is detected at a test slot, the testing for that slot was halted for the duration of the test period. Although all of the TCs were completed after 72 h, 29 potential defects were detected within the given test period as shown in **Fig. 5**. Moreover, the last detection of an event occurred at the 71[th] hour of the operation.

The sequential execution method selected one TC per test suite and executed the selected TC. Each TC had certain test samples assigned to it based on the weight of the test suite. The

predetermined sample size was not altered even if the events were detected, as in the case of the concurrent execution method. However, unlike the concurrent execution method, the test slots that have events are replaced with the next TC from the same test suite. Although all TCs were completed after 72 h, 19 potential defects were detected within the given test period as shown in **Fig. 6**. Notice that the last detection of an event occurred at the 7th hour of the operation.

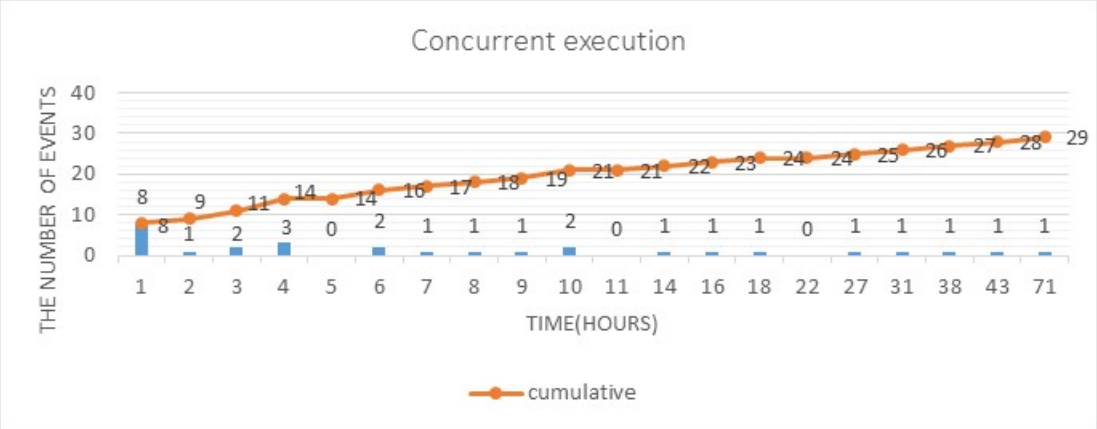

Fig. 5. Concurrent execution

Fig. 6. Sequential execution

4.3 The Proposed Self-Adaptive Method

The self-adaptive execution method selected all TCs and executed them at the same time, as in the concurrent execution method. However, when the events were detected, the prioritizations of the test suites were optimized by the proposed self-adaptive testing scheme. This strategy yielded much better results as compared to the previous two execution methods in our experiment as seen in **Fig. 7**. We notice that not only 30 potential defects were detected within the given test period, the last detection of an event occurred at the 22nd hour of the operation. The TSP and the test sample size were altered periodically throughout the entire testing period. The TCs that had high priority were executed intensively. Thus, the test sample size was optimized for each TC and for each event.

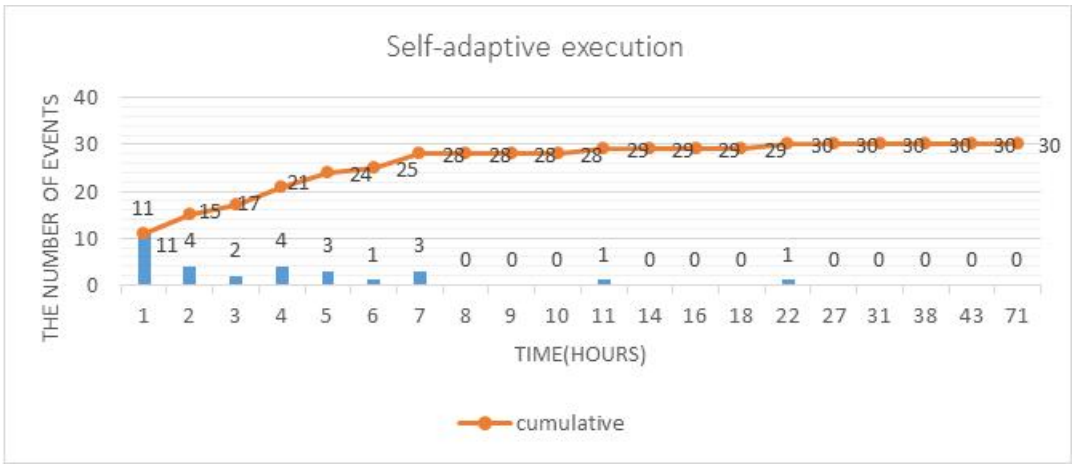

Fig. 7. Self-adaptive execution

Fig. 8 shows comparison of three graphs during the initial 22 h. The self-adaptive execution detects the largest number of events and has the fastest detection time as we can observe from the slope of the graphs. Especially, if the test is to be completed within a shorter test period to meet delivery deadlines because of a tight schedule, our proposed scheme would have a large effect because more defects can be detected early in the test. Further, faster beginning of analysis of defects can move the next release candidate version ahead in case of necessity as mentioned in **Fig. 1**. We also notice that every event of the same type does not have a different root cause every time. The root cause of each event can be determined through analysis of the event.

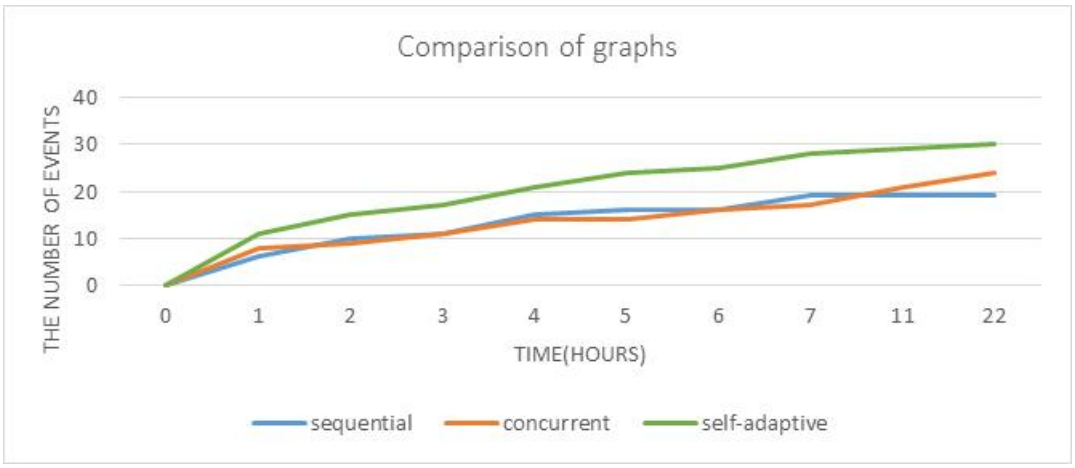

Fig. 8. Comparison of graphs

In this experiment, we contend that a large number of appropriate TCs should be concentrated on the weak features dynamically within a given test period. This is particularly important for long-term reliability tests that require a large number of test samples. However, there is a natural bound on the number of available test samples owing to physical limitations. We believe that our proposed method will address the challenges of optimal test sample size for long-term reliability testing in industrial manufacturing of flash memory solutions, which are facing a critical limitation of available test samples. Using this method, it is possible to detect the potential defects in the available test samples and test period.

5. Conclusion

It is impossible to detect all defects in a given time period with the available resources during product testing. Thus, it is necessary to concentrate all available resources on the weak features of the target product within the given test period. This is a critical issue for finding as many potential defects as possible during the long-term reliability testing of embedded software products. For this purpose, strategic decision making with regard to test sample size must be considered in the domain of flash memory solutions. In this paper, we addressed the challenge of optimal test sample size in industrial applications. We also introduced a scheme for self-adaptive testing during run-time. The self-adaptive testing scheme enables long-term reliability testing on optimal test samples during the entire test period without human intervention. Based on our feasibility study, we believe that the proposed scheme has the potential to lower the number of field defects considerably.

Strategic access of the test sample size is important for all types of embedded products. However, we only focused on flash memory solutions in this paper because we developed our proposed self-adaptive scheme on the predictable defects pattern and specific aspects of these defects. Moreover, unlike other solutions, due to the type of events that would occur at run-time is limited in flash memory solutions, better effects can be obtained by using our self-adaptive scheme. We have a plan to apply to other solutions without any limitations to ensure that the optimized sample size depending on real-time events decreases the potential defects. For this purpose, studies on the extended self-adaptive scheme for handling the run-time situations without any particular pattern have to be preceded.

References

[1] R. Kavitha, V. R. Kavitha and N. S. Kumar, "Requirement based test case prioritization," in *Proc. of Communication Control and Computing Technologies (ICCCCT)*, pp. 826-829, Oct. 2010. Article (CrossRef Link)

[2] X. Zhang, C. Nie, B. Xu and B. Qu, "Test Case Prioritization Based on Varying Testing Requirement Priorities and Test Case Costs," in *Proc. of Quality Software (QSIC)*, pp. 15-24, Oct. 2007. Article (CrossRef Link)

[3] D- Dmfrsqnïm, P. Runeson and A. Ljung, "Improving Regression Testing Transparency and Efficiency with History-Based Prioritization – An Industrial Case Study," in *Proc. of Software Testing, Verification and Validation (ICST)*, pp. 367-376, Mar. 2011. Article (CrossRef Link)

[4] M. Staats, P. Loyola and G. Rothermel, "Oracle-Centric Test Case Prioritization," in *Proc. of Software Reliability Engineering (ISSRE)*, pp. 311-320, Nov. 2012. Article (CrossRef Link)

[5] B. Jiang, Z. Zhang, W. K. Chan and T. H. Tse, "Adaptive Random Test Case Prioritization," in *Proc. of Automated Software Engineering (ASE)*, pp. 233-244, Nov. 2009. Article (CrossRef Link)

[6] Z. Q. Zhou, "Using Coverage Information to Guide Test Case Selection in Adaptive Random Testing," in *Proc. of Computer Software and Applications Conference Workshops (COMPSACW)*, pp. 208-213, July 2010. Article (CrossRef Link)

[7] M. J. Arafeen and H. Do, "Adaptive Regression Testing Strategy: An Empirical Study," in *Proc. of Software Reliability Engineering (ISSRE)*, pp. 130-139, 2011. Article (CrossRef Link)

[8] M. Fredericks, J. Ramirez and H. C. Cheng, "Towards run-time testing of dynamic adaptive systems," in *Proc. of Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pp. 169-174, May 2013. Article (CrossRef Link)

[9] J. Ramirez, C. Jensen and H. C. Cheng, "A taxonomy of uncertainty for dynamically adaptive systems," in *Proc. of Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pp. 99-108, June 2012. Article (CrossRef Link)

[10] P. Vromant, D. Weyns, S. Malek and J. Andersson, "On interacting control loops in self-adaptive systems," in *Proc. of Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pp.

202-207, 2011. Article (CrossRef Link)
[11] B. Sander and D. Eleco, "A self-adaptive deployment framework for service-oriented systems," in *Proc. of Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pp. 208-217, 2011. Article (CrossRef Link)
[12] N. Kaushik, M. Salehie, L. Tahvildari, L. Sen and M. Moore, "Dynamic Prioritization in Regression Testing," in *Proc. of Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 135-138, Mar. 2011. Article (CrossRef Link)
[13] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and Research challenges," in *Proc. of ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, Vol. 4, No. 2, May 2009. Article (CrossRef Link)

Chul-Hoon Byun is a M.S. candidate in the College of Information and Communications at the Korea University and a research engineer in Samsung electronics. His research interests include adaptive software testing, risk-based software engineering and embedded software engineering. He received his B.S. degree in Media (major) and Computer Science (dual major) from Ajou University in Suwon, South Korea.

Chang-Kyun Jeon is a Ph.D. candidate in the College of Information and Communications at the Korea University and a principal engineer in Samsung electronics. His research interests include requirement engineering, value-based software engineering, and embedded software engineering. He received his M.S degree in Control and Instrumentation Engineering from Kwangwoon University in Seoul, South Korea.

Taek Lee is a Ph.D. candidate in the College of Information and Communications at the Korea University. His research interests include software metrics, software defect prediction, information risk analysis, man-machine interaction, user behavior modeling in software system, and information security. He received his M.S. degree in Computer Science from Korea University.

Hoh Peter In is a professor at the College of Information and Communication at Korea University in Seoul, South Korea. His primary research interests are embedded software engineering, social media platform and service, and software security management. He earned the most influential paper award for 10 years in ICRE 2006. He has published over 100 research papers. He was an assistant professor at Texas A&M University. He received his Ph.D. in Computer Science from the University of Southern California (USC).