정적 테스트를 통한 소프트웨어 테스트 효율성 향상에 대한 사례 연구

(A Case Study on the Improvement of Software Test Effectiveness through Static Testing)

김 효 영 † 한 혁 수 ††
(Hyo Young Kim) (Hyuk Soo Han)

요 약 아직까지 소프트웨어 개발조직에서는 사전검증 활동 및 충분한 테스트 설계가 수행되지 않고 있으며, 따라서 개발 초기에 식별, 수정될 수 있는 결함들까지 테스트시 검출되기 때문에 투입 노력 대비 테스트의 효율성은 떨어진다. 이러한 문제의식을 바탕으로 본 논문은 구현이전단계에서의 사전검증이 수행되지 않고, 테스트 케이스 설계가 충분히 진행되지 않은 경우에 테스트의 효율성을 향상시킬 수 있는 방법을 실사례를 통해 제안하고 있다. 테스트 단계에서의 코드 리뷰와 코드 품질 분석을 통해 테스트 우선순위 선정, 테스트별 테스트 케이스 설계 등의 주요 활동과 테스트팀과 개발팀과의 역할을 구체적으로 제시한다.

키워드 : 정적 테스트, 테스트 효율성, 소프트웨어 테스팅

Abstract Not enough verification or enough design of test is not performing in many software developments organization as yet. Therefore, defects that can be detected and corrected during the beginning phases of development are usually found during dynamic testing, it is often observed that testing is inefficient compared to effort for testing. This study aims to suggest a method for effective testing through case study. It is useful in case of not verification and not enough design of test in the previous phase than coding. We show in a concrete way major activities that determine prioritization of testing and level of test case design through static testing, i.e. code review and analysis of code quality. And also we show role between test team and development team.

Key words : Static test, Test effectiveness, Software testing

1. Introduction

There exists a broad range of applications for software and the importance of quality is rising. As services provided by software increase, the size as well as complexity of software are increasing, which require more effort for quality assurance. Software quality can be improved through systematic analysis of requirement, design, active testing, and through careful static reviews at each development stage. The most common method of several activities is the active test, which validate whether the software works right through test or not. However, potential defects and malfunction rate of the final system will increase if the test is not thoroughly completed for non-functional requirements such as performance and function tests for normal and abnormal cases[1].

As explained commonly in V&V models, planning and designing any tests start by gathering and analyzing requirements. Thus, such activities highly depend on development activities of previous processes such as requirement analysis, design, coding, etc[2]. Therefore, success of failure of test activities is determined according to systematic processes of development activities in each phase. Unfortunately, since a systematic development model often lacks in many software development organizations, it is

† 정 회 원 : LG전자 S/W & S센터 S/W Engineering Gr. 선임연구원
goma1@smu.ac.kr
†† 종신회원 : 상명대학교 소프트웨어학부
hshan@smu.ac.kr
논문접수 : 2006년 8월 4일
심사완료 : 2006년 12월 5일

difficult to apply a cohesive stage-to-stage testing model that flows smoothly from function and subroutine testing, to the testing of integrated modules, and ending with the testing of the complete system. Furthermore, this situation becomes increasingly serious as the size and complexity of the software project increase. This becomes further difficult when development lifecycles and Time to Market are short. Generally in such cases, there is a great dependency on test engineers, and testing is focused at the system level during the final stages of development. Most defects are found during testing of the software. This means that testing is inefficient compare to effort for testing because defects that can be identified and rectified at early stage are found lately[3,4].

It is difficult to apply a systematic test for various reasons, and it contains problems of delaying project by reworking at the last phase-it is a reality that this cannot be easily rectified. With such problems in mind, we study about improvement cases of test effectiveness through static test. We are considering about prioritizing tests through code analysis and review, and adjusting level of test case design by analyzing characteristics of software and current project situation. This paper suggests a more efficient testing activity model based on the results observed in the case study.

The following is organization of this paper. Chapter 2 introduces static test and the case study of IBM. Chapter 3 explores the inherent problems in the case study, reinforces the concept of static test methods, and reviews the results of improvement.

2. Related work

2.1 Static test

Software testing can be classified in many ways; it can be differentiated into dynamic test and static test depending on whether software runs to identify software defects. While dynamic testing requires running of the software, static testing can be conducted without operating the software. Static test is a method to identify unusual behaviors or errors without running the codes. Static testing includes inspection and review activities such as code walkthroughs and analysis with SRS and design documentation (HLD, LLD) [1,5,6].

(1) Review

Reviews can be done with all major outputs from each development phase, such as SRS, design documentation, and source code. Reviews such as inspection and walkthrough are a method of identifying defects statically by experts or developers.

Introduced by Fagen in 1976, inspection is a review-based method of identifying errors through application of a pre-made checklist. Different from walkthrough or technical review, inspection is a formal activity that relies on a series of steps including overview, preparation, individual inspection, inspection meetings, rework, and follow up. For individual inspection and inspection meetings, inspectors generally utilize checklists that contain major check items [1,7,8].

On the other hand, walkthroughs are informal reviews that can be carried out during software development, when in need. Walkthroughs generally do not utilize checklists to identify defect [1].

(2) Code Analysis

Source code analysis can be effective in identifying potential defects, and is performed with the aid of code analysis tools[9].

Since such tools can be used on software that is not yet executable, they offer a very useful means of code analysis for early start of test. More effectively than dynamic testing, such tools can often identify a large range of hidden defects such as memory leaks, as well as provide general metrics of code quality and complexity. We can decide test levels and scope through several metric information.

Besides several researcher mentioned about improvement test effectiveness by static test. Winkler studied the Usage Based Reading (UBR) technique, Usage Based Testing (UBT-i) that integrates testing scenarios and inspection techniques. He explained mention that is useful for design specification as well as defect detection in early stages of software development. Laitenberger proposed to apply static test by inspection, code analysis and dynamic test. Static and dynamic test are comple-

mentary each other [10].

Several researcher include Laitenberger suggested that static test like inspection and dynamic test should be applied in combination rather than in isolation [11-13]. Laitenberger show the effects of combining software inspection and structural testing on software quality by experiment. One of result of his experiment present 39 percent (on average) of the defects were not detected at all, it might be more valuable to apply inspection together with other testing techniques, such as boundary value analysis, to achieve a better defect coverage [10].

Lavenhar described that one of the most effective ways to identify and manage risk for an application is to iteratively review its code throughout the development cycle. According to his opinion substantial net improvements in software security can be realized through the formal use of design and code inspection. He presents best practices for performing code analysis to uncover errors in and improve the quality of source code. Methods include manual code auditing, walkthroughs, static analysis, dynamic analysis, metric analysis, testability analysis, crypto analysis, random number analysis, and fault injection [14,15].

2.2 Improvement of test effectiveness by testing techniques and test activities

Most of previous studies to improve test effectiveness are related application of test techniques. That kind of studies focused on the test cases design. Pizza and Strigini considered test effectiveness in improve of defect detection by to compare several test techniques [16]. And Frankl with his colleague are comparing several white box test techniques by coverage of that in effectiveness view of point [17].

We can consider that project team analyze the cause and identify areas for test improvement process by other approach for test effectiveness. Chernak proposed any practice for that in his article. Understand and document the test process, identify the factors for enhance test case design and improve defect detection rate, and then make up for the weak points [18]. That study focused on the test cases design also.

2.3 The practices of IBM

Many organizations put various efforts to carry out an efficient test and achieve high quality from early phases, and various cases have been introduced consequently. However, according to a technical report in 1999 by IBM, close relationship of testing to other development activities was stressed.

IBM's report introduces 28 best software development practices on testing. Among these, the report proposes functional specifications, review and inspection, formal check-in and check-out procedures, functional varieties of test, multi-platform testing, internal betas, automated test execution, beta programs, and nightly builds, as the 9 most fundamental and necessary practices [7,19]. Review and inspection are emphasized as static tests for effective method of testing. Such activities at the early stages of development prevent unnecessary efforts for testing and reduce dynamic test cycle times.

Other major practices are comprised in teaming testers with developers, code coverage, automated environment generators, testing to facilitate ship on demand, state task diagrams, memory resource failure simulation, statistical testing, semiformal methods, check-in tests for code, minimizing regression tests, instrumented methods for calculating MTTF (mean time to failure), benchmark tests, and bug bounties.

While diverse practices outlined in the IBM report are quite useful, the application of these practices will differ according to systematization of the organization's production environment, maturity level of the organization, and the skill level of developers and testers.

2.4 A challenge of previous works

A diversity of method proposed through many research, i.e. review and analysis of code quality. And many development organizations make an effort for review, static analysis. Generally many researches focus on method itself review, static analysis, testing and so on. Mostly they don't consider application of techniques and don't explain case study based on practical development situation.

Review is typical static test. Basically that is conducted in each phase while development by

developer. If most software development is systematic development, we don't need consider that kind of method for test effectiveness. Though most organizations recognize the importance of static tests, and some level of static testing is usually employed, these tests are often not applied systematically by that kind of reasons. Therefore testing in last phase of development lifecycle is non effectiveness. There is nothing for it but to do. This study review about how can be used various test method and what is the effective method for software that finish coding under against time for delivery. And then we propose applicable effective test method considering the situation of software development through case study.

3. A case study

This study has searched and applied the test effectiveness and quality improvement methods on the API test of a middleware application targeted for an embedded system. The approach used and its result are described in the following sections.

3.1 Status Analysis for improvement

Observing the project status for testing, the coding for this project had already been completed, but reference materials for test case planning were deficient. Furthermore, the number of software's APIs were 977 making it difficult to complete a full test in a given period of time. Therefore, we decided to focus on the APIs that have high risks, and planned the testing accordingly.

Basic standards for selecting APIs for testing were-APIs that are actually used, APIs with high risks (complicate codes, unstructured files, and code with many lines), and APIs where problems were found while applying to products. We used static analysis tool to measure code risk rate and determined test case design considering the analysis result and the functional characteristics. The selection standard was based on the advice of McCabe in his studies regarding complexity measurement [20]. We selected API codes with Complexity v(g) above 10 and Essential Complexity ev(g) above 40 as primary test materials. Code analysis identified 61 APIs for primary testing. We considered prioritization based on the analysis of results from previous tests, decided the level of test design, and carried out the test. White Box test as well as Black Box test is conducted about identified 61 API by code risk evaluation. We designed many test cases for Black Box test and White Box test. However we design normal and abnormal test cases for Black Box test, just conducted Black Box test in other APIs.

Table 1 shows the standard of code risk evaluation for selecting APIs subject to testing, and Table 2 shows the examples about the applied results.

Table 1 Standard of Code Risk Evaluation

Metric	Very High	High	Average	Low	Very Low	Guideline
Complexity v(g)	> 50	21~50	11~20	6~10	1~5	10 or less
Essential Complexity ev(g)	> 28	21~28	13~20	5~12	1~4	4 or less

Table 2 Examples about Evaluation Results

API	LOC	v(g)	ev(g)	iv(g)	Weight of Risk Factors			Risk Score
					v(g) (50%)	ev(g) (30%)	LOC (20%)	
REditCtrl'REditCtrl_Mxx	120	15	23	0.19	3	4	5	3.70
EditCtrl_Getxx	128	24	6	0.2	4	2	5	3.60
RListCtrl_Ensurexx	120	15	15	0.16	3	3	5	3.40
Dialog'Dialog_Createxx	119	3	27	0.27	1	4	5	2.70
EditCtrl_Ensurexx	68	11	7	0.23	3	2	3	2.70
EditCtrl_Insertxx	61	13	10	0.26	3	2	3	2.70
ClndCtrl_Convertxx	94	19	2	0.26	3	1	4	2.60
RListCtrl'RListCtrl_Createxx	80	1	22	0.17	1	4	4	2.50
ClndCtrl_ConvertSolxx	59	14	1	0.24	3	1	3	2.40
REditCtrl'REditCtrl_Reorderxx	90	8	8	0.2	2	2	4	2.40

3.2 Improvement point

After analyzing the identified defects by API Test, it was observed that most defects could have been identified without the use of any dynamic testing. Many defects could have been identified and repaired through code reviews before test. Table 3 shows defects organized by type. Among the values, errors of return values and defects regarding halts during operation were found to be pre-identifiable through review.

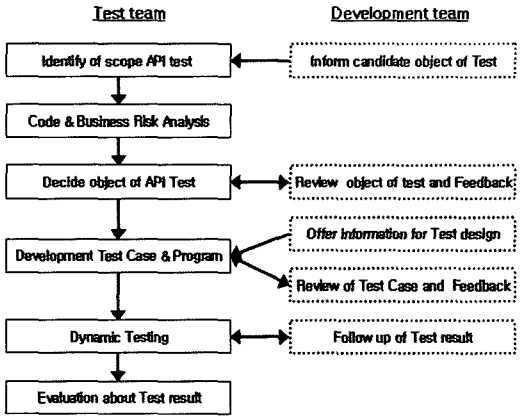

Figure 1 API Test Activities Before Implementing Modified Static Tests

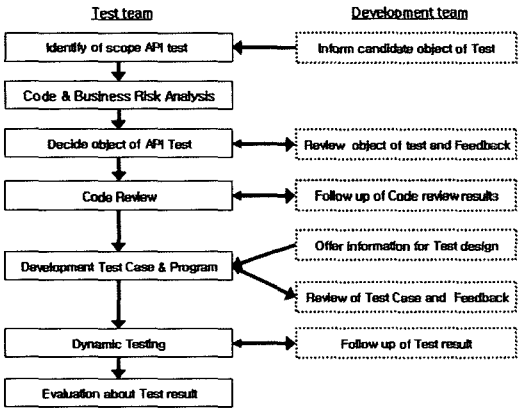

Figure 2 API Test Activities After Implementing Modified Static Tests

We carried out code reviews prior to test case design, identified potential defects, supplied this information to the development team, and then carried out the tests after correcting the identified defects.

Figure 1 depicts the major activities of API test before implementing procedures of static testing, and Figure 2 shows about activities after implementing modified procedures of static testing.

3.3 The result of Test effectiveness improvement by code review

Through the analysis of the first test results (Table 3), we identified pre-detectable errors like return value error according to the modified test processes described in Figure 2, carried out code reviews prior to test activities, and eliminated unnecessary test to raise the effectiveness of these tests. At this point, based on the analysis of prior faults, we also produced a checklist for repetitive problems so code review can be carried out more effectively. Table 4 is a checklist of items inspected by the test team.

After carrying out the static tests by code review based on the major review items suggested in table 4, a total of 59 errors were identified as follows: 3 memory release errors, 43 data validity verification errors, 5 repetition errors, 1 unnecessary condition error, and 7 return value errors for exception handling function. Those errors related return value error and halt under operation. We provided feedbacks for identified defects to the development team, designed test cases, and carried out the dynamic test on the corrected code. Especially in case of halt error have an effect on testing time. If we didn't code review, many time and effort are needs for dynamic test. After comparing current results to previous test results, return value error reduce from 24% to 14% in total defect (42% improvement). And Halt errors reduce from 51% to 22% (about 57% improvement) as shown in Table 5.

Table 3 Analysis of Defect (Before)

Defect Type	Function Error	Return Value Error	Display Error	Halt During Operation	Total
Number of Defects	62	97	35	204	398
%	16%	24%	9%	51%	100%

Table 4 Check point in Review

Type	Check item
Comment	Does the API module have a header at the start of the file?
	Are complicated algorithms and limited conditions sufficiently explained?
	Is there an explanation on codes with comments?
Exception Handling	Is Assertion used for valid values or ranges?
	Are errors being handled appropriately every time a function returns?
	Do resources and memory get released every time an error occurs?
	Is appropriate error handling codes returned from call functions when an internal error occurs?
	Does it check for out of bound for all used arrays?
Resource Leak	Does all memory used internally get released?
Control Structure	Is termination condition of loops accurate?
	Was there a repetitive or unnecessary condition applied?
	Is there reiteration of excessive condition or a need for re-construction?

Table 5 Comparative Analysis of defects

	Defect Type	Function Error	Return Value Error	Display Error	Halt During Operation	Total
Before	Number of Defect	62	97	35	204	398
	Percent (%)	16%	24%	9%	51%	100%
After	Number of Defect	63	14	N/A	22	99
	Percent (%)	64%	14%	N/A.	22%	100%

Checklist items are errors typically made by developers. Therefore, if checklist items are applied to coding standard or as basic review items, checklist items can be used as an effective tool in enhancing development activities. In the future, the items will be applied to coding standards so that developers can use them as a reference when coding. We also plan to reduce the re-working load of the development team through code inspection.

4. Conclusions

As can be learned from the previously introduced IBM case, it is necessary to systematically develop and start test activities from early stages of development simultaneously for effective testing. We can efficiently achieve the quality through dynamic testing by reviewing major stages and identifying errors in advance. This study focused on reviews and inspections, and teaming testers with developers, as noted in the best practices suggested by IBM. Also, by using code quality metrics to create risk-based test scopes and deciding level of test case design, we attempted to increase the overall effectiveness of testing activities.

Review activities are very effective methods for preventing product defects during development. It is also a very important activity in terms of test, as proved in suggested sample study case. Test team provides defect types from test results analysis, which can be pre-identified and improved during developing procedures, and preventing methods to development team. Then it will greatly improve effectiveness and quality of overall development.

There are various ways to improve quality and productivity. We must concern that productivity can be debilitated if practices are applied without considering the organization and the specifics of the project. Therefore, it is very important to decide what is most needed, and this can be best done by analyzing the current status. In this study, we showed practice to improve test effectiveness through reinforcement of static tests such as code reviews and code analysis. We plan to continuously attempt practical approach so that test can be carried out more effectively through the analysis of various factors related to test activity and test productivity.

References

[1] 한동수, 정인상, software test 입문, VI Land, 2004.
[2] W.S.Humphrey, Managing the Software Process, Addison-Wesley, 1990.

[3] M.E Fagen, "Advances in Software Inspection," IEEE transactions on Software Engineering, Vol 12, issue 7, pp.744-751, 1986.
[4] D.Graham, Software Inspection, Addison-Wesley, 1993.
[5] B.Hailpern, P.Santhanam, "Software debugging, testing, and verification," IBM System Journal, Vol.41, No.1, pp.4-12, 2002.
[6] R. S. Pressman, Software Engineering: A Practitioner's Approach, McGraw-Hill, New York, 1992.
[7] Ram Chillarege, "Software Testing Best Practices," IBM Technical Report RC 21457, Center for Software Engineering IBM Research, 1999.
[8] M.E.Fagen, "Design and code inspections to reduce errors in program development," IBM Systems Journal, Vol.15, pp.182-211, 1976.
[9] D. Winkler, S. Biffl, B. Riedl, "Improvement of Design Specifications with Inspection. and Testing," Proc. Of Euromicro 05, pp.222-230, 2005.
[10] O. Laitenberger, "Studying the Effects of Code Inspection and Structural Testing on Software Quality," Proc. 9th Int'l Symp. Software Reliability Eng., IEEE CS Press, pp. 237-246, 1998.
[11] V.R. Basili, R.W. Selby, Comparing the effectiveness of software testing techniques. IEEE Transactions on Software Engineering, pp.1278-1296, December 1987.
[12] B. Beizer, Software Testing Techniques. International Thomson Publishing Inc., 2nd edition, 1990.
[13] M.Wood, M.Roper, A.Brooks, and J. Miller, Comparing and Combining Software Defect Detection Techniques: A Replicated Empirical Study, in Proceedings of the 6th European Software Engineering Conference, pp.262-277, 1997.
[14] C. Michael, S.R. Lavenhar, Source code analysis tools overview. https://buildsecurityin.uscert.gov/portal/article/tools/code_analysis/overview.xml, September 2005. Published via the U.S. Department of Homeland Security Build Security In website.
[15] B. Chess, G. McGraw. Static analysis for security. Security & Privacy Magazine, IEEE, pp.76-79, 2004.
[16] M.Pizza, L.Strigini , "Comparing the effectiveness of testing methods in improving programs: the effect of variations in program quality," Proc. Ninth International Symposium on Software Reliability Engineering, ISSRE '98, Paderborn, Germany, IEEE Computer Society Press, pp. 144-153, 1998.
[17] P. Frankl, O. Iakounenko, "Further Empirical. Studies of Test Effectiveness," SIGSOFT '98, Nov,. pp.153-162, 1998.
[18] Y.Chernak, "Validating and Improving Test-Case Effectiveness," IEEE Software, January-February, pp.81-86, 2001.
[19] D.Brand, "A Software Falsifier," Proceedings, Eleventh IEEE International Symposium on Software Reliability Engineering, San Jose, CA, pp.174-185, 2000.
[20] McCabe, A Complexity Measure, IEEE Transactions On Software Engineering, Vol.Se-2, No.4, December, pp.308-320, 1976.

김 효 영

1999 Dept. of Multimedia, SangMyung Univ. Graduate School of Information and Telecommunications(Master). 2000~2002 Dept. of Computer Science, SangMyung Univ. Graduate School (Completion of Ph.D Courses). 2002~2004 Software Strategy Gr. Digital Media Laboratory of LG Electronics (Manager). 2005~Present Software Engineering Gr. Software & Solution Center of LG Electronics (Senior Research Engineer). Research Interests: Software Quality, Software Process, Software Usability Evaluation, Software Testing, Software Metric

한 혁 수

1985 Seoul National Univ. Dept of computer science (Bachelor). 1987 Seoul National Univ. Dept of computer science (Master). 1992 University of South Florida. Dept of computer engineering (Ph.D). 2001~Present Chairman of SITRI (System Integration Technology Research Institute). 2003 Executive Director of KIPA (Korea IT Inductry promotion Agency) in KSI(Korea Software Institute). 2004~2005 Dean of SangMyung University School of Software. 1993~Present Sang Myung University School of Software(Professor). Research Interests: Software Process, Software Quality, Software Usability Evaluation etc.