Journal of Computer Science 3 (3): 144-148, 2007
ISSN 1549-3636
© 2007 Science Publications

A Case Study of Black-Box Testing for Embedded Software using Test Automation Tool

[1]Changhyun Baek, [2]Joongsoon Jang, [3]Gihyun Jung, [4]Kyunghee Choi, and [5]Seungkyu Park
[1, 4, 5]Graduate School of Information and Communication, Ajou University, Republic of Korea
[2]Division of Industrial and Information System Engineering, Ajou University, Republic of Korea
[3]Division of Electronics Engineering, Ajou University, Republic of Korea

Abstract: This research shows a case study of the Black-Box testing for Temperature Controller (TC) which is one of the typical embedded systems. A test automation tool, TEST, was developed and some kinds of TCs were tested using the tool. We presented statistical analysis for the test results of the automated testing and defined the properties of the software bugs for the embedded system. The main result of the study were the following: (a) test case prioritization technique was needed because the review phase, in the test process, takes long time; (b) there are three types of software defects for the embedded software; (c) the complexity of the system configuration have an effect on the software defect; (d) the software defect was distributed in vulnerable points of the software; and (e) testing activities reduce the number of the software defects. The result can be useful in the asymptotic study of test case prioritization.

Key words: Black-box testing, embedded software, system test, software defect, test automation tool

INTRODUCTION

As a scale of market for the embedded software has grown up, product's quality and reliability hold a key post in the software development process. In 2004, one of the world famous mobile companies did additional after service because of some kinds of software defects. Like this, there have been many cases of defects of the embedded system as software with short product life cycle recently. For this, more effort for the quality assurance of embedded software has been required[1]. Each enterprise or companies have tried to reduce defects on software development process, moreover, to describe requirements and to verify that it is satisfied, between in each process. To use the test automation tools is one of good approaches. They are skillfully devised for convenient use and give testers the testing environment which is more correct than the manual testing. The conventional test automation tool for the embedded software usually was used for Unit Test. Also, many times of modification and adaptable periods were required to apply to specific system. These problems caused impossibility for the testing. Generally, embedded system's configuration of hardware is very various and there would be a difficulty for generalization of testing environment, so the conventional test automation tool for the embedded software testing couldn't be applied to various systems[2-4]. Because of this situation or restriction the conventional development of embedded system company has not used the existing automation tool for the testing, or tested in limited situation. It had to develop new testing automation tool for the testing fitted on a developing system as well[5-7].

In this study, we developed the test automation tool, *Testing stand for Embedded Software Testing (TEST)*, which is dedicated to the embedded system, introduced the annual testing results and presented properties of the software defects for the embedded software.

Temperature controllers: Temperature Controller (TC) is one of the typical embedded systems that gets input variables by environmental factors of outside and is controlled by switch input by user. TC operates with ambient sensor, in-car sensor, humidity sensor and photo sensor on environmental factor of outside. As a user inputs each switch, it decides indicator's on/off. Embedded software inside of TC carries out decision of intention like indicator, display and actuator by environmental condition and user's input. Fig. 1 shows a hardware configuration of the TC. TC is classified with FATC (*Full Automatic Temp. Controller*) and MTC (*Manual Temp.Controller*). In FATC, internal software's function is complicated so every possible cases' testing is required.

Corresponding Author: Seungkyu Park, Graduate school of Information and Communication, Ajou University, San 5, Suwon, Kyunggi, Republic of Korea

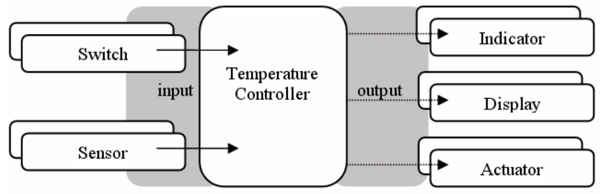
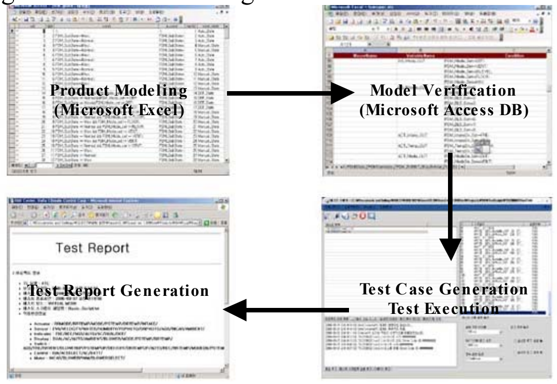


Fig. 1: Hardware configuration of TC

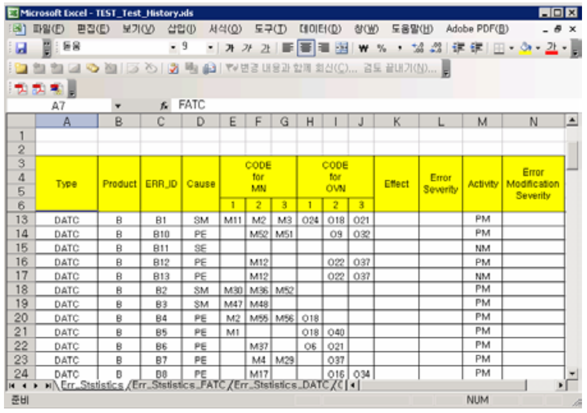

Product Modeling (Microsoft Excel)
Model Verification (Microsoft Access DB)
Test Report
Test Report Generation
Test Case Generation
Test Execution
Fig. 2: Snapshots for test phase using TEST

Test automation tool: The main reason to use the automation tool for testing is to overcome the demerits of the existing manual test. The time cost can be reduced for framing Defect Record, documentation and writing report. The exactness of test result can be also guaranteed in previous studies which compare the manual test and the automation test, the manual test requires more than two times of expense and time as well[8].

Testing software's maximizing of the automation on every procedure on testing can be a standard to ability of tool. How much the developed testware supports to test procedure like test script generation, test execution and test report generation, there would be ability as an automation tool. A study for the realization of the automation tool like test case's operation of automation or automated comparison between actual result and expected result has been in progress. Also, a study of organization of test report template for embedded software has begun[9-11].

TEST (Testing-stand for Embedded Software Test) includes 5 functions for verification of embedded software[12]. 'product modeling' which reflects FSM and product specification on Microsoft Excel, 'model verification' which verifies exactness of input by user[13-15], 'Test script generation' which creates test cases automatically based on product specification, 'test execution and comparison' which executes test cases and 'test report generation' which reports test result that is used as data of defect verification[16,17]. TEST embodied in Microsoft Visual Basic.Net, VBA Macro and many DLL based on C.

Test accomplishments: Now as it introduced earlier, it shows the result of test accomplishments about total 7 tests for TC by using TEST. As it explained earlier, TC is divided by two types. In MTC, however, functions of SW are simple and the outbreak of defect was not reported, so we ruled out all test results which is from testing MTC. Accordingly, only 7 results of ATC are introduced next. Fig. 3 shows the primitive data of test result. One software defect is classified by some criterions to find its properties.


Microsoft Excel - TEST_Test_History.xls

A7 FATC

Type	Product	ERR_ID	Cause	CODE for MN			CODE for OVN			Effect	Error Severity	Activity	Error Modification Severity
				1	2	3	1	2	3				
DATC	B	B1	SM	M11	M2	M3	O24	O18	O21			PM	
DATC	B	B10	PE		M52	M51		O9	O32			PM	
DATC	B	B11	SE									NM	
DATC	B	B12	PE		M12			O22	O37			PM	
DATC	B	B13	PE		M12			O22	O37			NM	
DATC	B	B2	SM	M30	M36	M52						PM	
DATC	B	B3	SM	M47	M48							PM	
DATC	B	B4	PE	M2	M55	M56	O18					PM	
DATC	B	B5	PE	M1			O18	O40				PM	
DATC	B	B6	PE		M37		O6	O21				PM	
DATC	B	B7	PE		M4	M29		O37				PM	
DATC	B	B8	PE		M17			O16	O34			PM	

Err_Ststistics / Err_Ststistics_FATC / Err_Ststistics_DATC

Fig. 3: Primitive data for analyzing test results

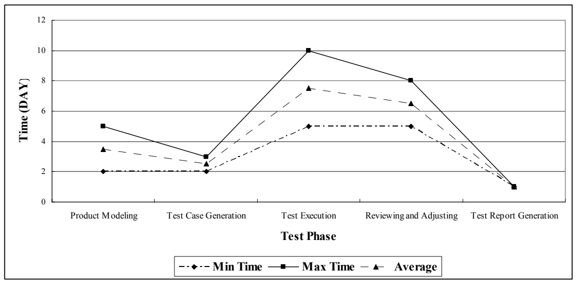


Fig. 4: Time cost for test phases

Time cost for testing: As the result in Fig. 4, it takes from 2 weeks to 1 month to test a unit system. For testing a unit system, the number of accomplished test case, there can be small differences of number of the test cases, is about 50,000.

In each phase, it was realized that the time for executing test cases, reviewing and adjusting the found out defect, takes too long. During test process, the time for carrying out of the test case takes more than 35% of the whole process. In ‘reviewing and adjusting defect’ phase which is directly related to this, during executing test cases, it starts from where the defect happens, so if a software defect is found out, the test period should be longer. In consequence, a study that test case can be operated in preferred order to find out such software defect quickly during the test process, has been going on.

```
if ( iVar >= 255) ----------------- □ exact code
if ( iVar >  255) ----------------- □ defect code
```

Fig. 5: Example of programming defect

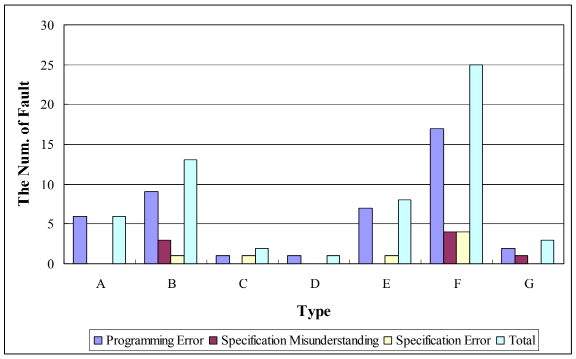


Fig. 6: Classifications and test results of the software defects

Software defect classification: As the result using by test automation tool, 1-25 software defects were found out in each unit system. Fig. 5 described the found out defect in Programming Defect, Specification Misunderstanding and Specification Defect.

Programming Defect is traditional software defect and means that it happened by programmer's defect and it takes 74% of found out defect. For the example of this defect can be like using 'bigger' instead of 'bigger or equal'. Fig. 3 indicated the example of the Programming Defect.

Specification Misunderstanding means that a developer misunderstands the given specification and defect happens. A developer analyzes by his or her own, however, between multitudinous and complicated logics are mentioned on specification, so defect happens. These kinds of problems are almost 14% of the whole.

Specification Defect is when a specification of product development includes defect by a developer. In this case, test automation tool reported there was a defect on SUT, but, in fact, there was a problem in specification that used by test automation tool and it takes 12% of the whole.

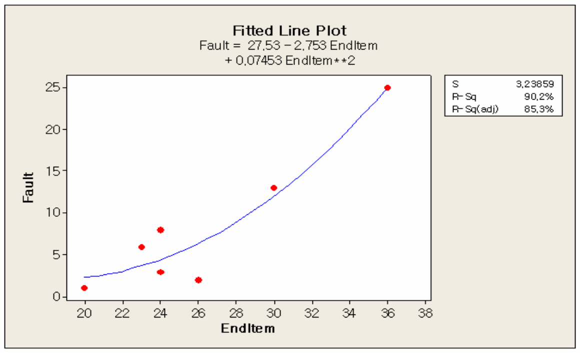


Fig. 7: Distribution of software defect

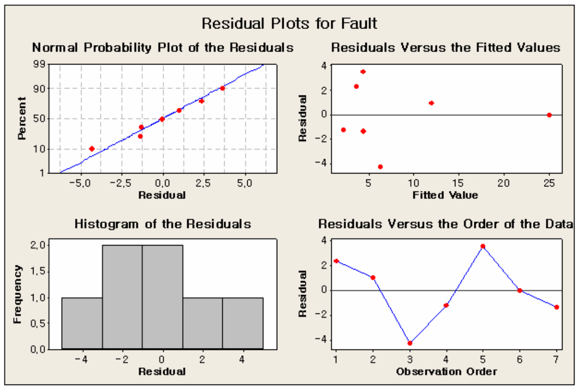


Fig. 8: Residual plots for software fault

Hardware configuration and software defect: Fig. 7 shows number of software defect about number of actuator or sensor connected in one unit TC. Through the 7 each of system it can get the result that as much as complicated organization of hardware has given, more and more software defect can be lied. When the number of organization factor of hardware which are connected in system, are getting more, it means there are many things to control inside of embedded software, also it can cause compilation by software's increasing. These result means when embedded software is measured, the complexity of hardware configuration can be used as a factor, like using lines of program was for existing traditional software's complexity.

We can predict the number of software defect which are in the given software using these results. Fig. 7 shows that $N_{SoftwareFault}$ (the number of software fault) would be increased linearly as $N_{EndItem}$ (the number of End Item) have been increased. This fact could be used to decide the time to stop software testing. Fig. 7 and 8 were drawn by MINITAB based on real test result. We did a polynomial regression analysis between $N_{SoftwareFault}$ and $N_{EndItem}$. We can find out the regression equation like Equation 1. (S=3.23859, R-Sq=90.2%, R-Sq(adj) = 85.3%, $F_{Regression}$ = 18.38 and $P_{Regression}$ = 0.010)

$$Y = 0.0745X^2 - 2.7535X + 27.531$$

Equation 1 Regression Equation between $N_{SoftwareFault}$ and $N_{EndITem}$

Output variable and software defect: Fig. 9 is the result of mapping with software defect and output variable. Those test objects, 7 TC have similar hardware organization and depending on system, additional input has existed.

Therefore from black-box test which verifies software through in/output, result of output defect is considered as software defect. As shown on the Fig. 9, specific output defect is centralized. In the Fig., output: O17,

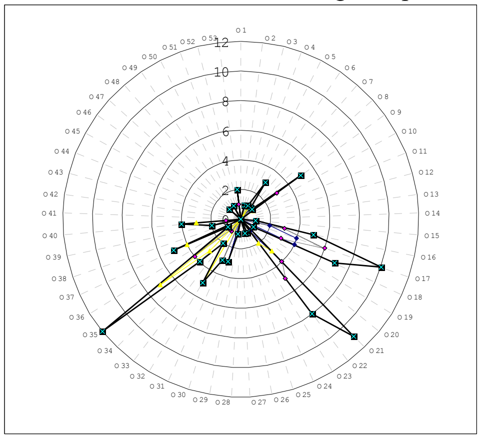


Fig. 9: Distribution of software defect based on output variables

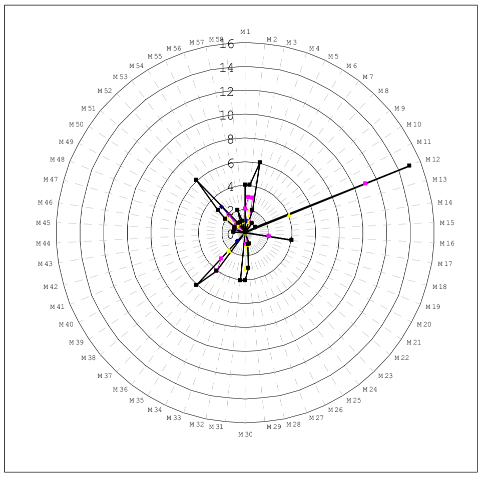


Fig. 10: Distribution of software defect based on internal modules

O21 and O35, were assigned as that value through the many different conditions and then more software defect was caused by comparing with other input. Fig. 10 is about distribution of internal logic. In Fig. 9, it takes a look at the relationship between defect and specific output. Similarly with this, in Fig. 10 it is about a relationship between defect and specific internal logic. These results can be used as prediction of confidence of embedded software. It tells framing specific logic or system related with a specific variable should be clear. Reorganizing complicated formation of specific logic in simple system is needed.

These results could be used to order the test cases as a test case prioritization technique. For example, we should change the execution order between the test cases to find out the software defect more quickly. As we mentioned earlier, the review phase require much time to decide whether test result is the software defect or not. For that reason, recently various test case prioritization techniques are researched.

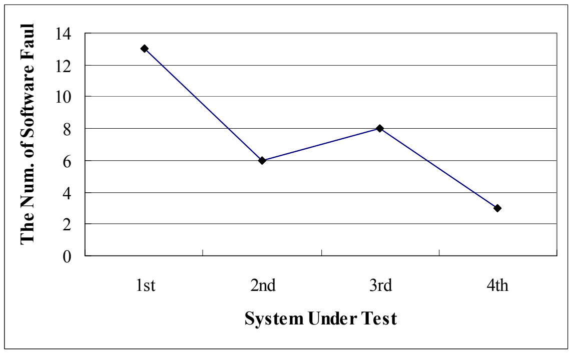


Fig. 11: Software defects is on a decreasing trend

Test case prioritization techniques have been shown to improve regression-testing activities by increasing the rate of defect detection, thus allowing testers to fix defects earlier. Hema Scrikanth et al. developed a prioritization scheme with two main goals: identifying the severe defects earlier and minimizing the cost of test case prioritization. They proposed the prioritization scheme based on customer-assigned priority and requirement complexity[18]. This scheme has a restriction that prioritization is done by expert although it is impossible to collect talented expert in a small or middle enterprise.

Sebastian Elbaum et al. orders the test cases based on the number of function calls and proposes the scheme that modification parts of the software have higher priority than others when testers want to test the software in the passion of the regression testing[19]. This method could be applied to the black-box testing because its ideas are on the basis of the white-box testing.

Software defect reduction: Fig. 11 shows number of software defect of TC delivered by two same companies. They are different types of system, but basic organization of HW/SW is similar. After development of test automation tool, 13 software defect were found out at initial applied model, however only 3 defect were discovered at fourth model. This shows as one company which orders an embedded system applies test automation tool, a positive manner for quality verification has an effect on product developer, so operation from development step to increasing of product's quality is progressing. Also it can be analyzed as applying test automation tool, it can be guaranteed of verification ability and discussion about developer's product.

CONCLUSION AND FUTURE WORKS

In this study, TC which is one of the embedded systems was introduced as the automation tool for black-box test. Through the results of the several TC test, additionally using test automation tool to embedded software which has taken manual test already, so it can take apart more fallacies. Accordingly, it confirms this would work on positive side of increasing product's quality. In the future, in the future, as in a view of rate between timing of test accomplishment and discovering defect, study which is for increasing of utility factor for test, formatting of superior quality of test case and organizing test case in ranking with new techniques, will be advance. In this part as it introduced in part 3, it shows the result of test accomplishment about total 7 tests of TC by using test automation tool. As it explained in part 2, TC is divided by two types. In MTC, however, function of SW is simple and the outbreak of defect was not reported, so it was excepted from the test result. Accordingly, only 7 results of ATC are introduced in this part.

REFERENCES

1. Ireson, W.G., C.F. Coombs and R.Y. Moss, 1995. Handbook of Reliability Engineering and Management. McGraw-Hill Professional.
2. Beizer, B., 1995. Black-Box Testing. John Willey & Sons.
3. Beizer, B., 1990. Software Testing Techniques. 2nd Edn., International Thomson Computer Press.
4. Jang, Y., K. Yeo and H. Lee, 2003. A case study of the manual testing and automated testing. KISS Conference.
5. Broekman, B. and E. Notenboom, 2003. Testing Embedded Software. Addison Wesley.
6. Rodd, K., 1998. Practical Guide to Software System Testing. K.J. Ross & Associates Pty. Ltd.
7. OVUM, 1999. Software Testing Tools, OVUM.
8. Sowers, M., 2002. Software Testing Tools Summary, Software Development Technologies Inc. White Paper.
9. Hayes, L., 1995. The Automated Testing Handbook. Software Testing Institute.
10. Fewster, M. and D. Graham, 1999. Software Testing Automation: Effective Use of Test Execution Tools. ACM Press, Addison Wesley.
11. Dustin, E., J. Rashka and J. Paul, 1999. Automated Software Testing. Addison Wesley.
12. Baek, C., S. Park and K. Choi, 2005. TEST: An Effective Automation Tool for Testing Embedded Software. WSEAS Trans. on Information Science & Applications, Vol. 2.
13. Kim, B., C. Baek, J. Jang, G. Jung, K. Choi and S. Park, 2004. A design and implementation of the check module for the test of embedded software. KISS Conference.
14. Kim, B., C. Baek, J. Jang, G. Jung, K. Choi and S. Park, 2005. A design and implementation of virtual environment operator for the embedded software test. KCC 2005.
15. Hoffman, D., 2001. Using test oracles in automation. Pacific Northwest Software Quality Conference.
16. Jeon, H., C. Baek and S. Park, 2005. A test report for the effective defect tracing of the embedded software. Vol. 32, KICS Conference.
17. IEEE-SA, 1998. IEEE829: IEEE Standard for Software Test Documentation. IEEE-SA Standards Board.
18. Srikanth, H and L. Williams, Requirements-Based Test Case Priotization.
19. Elbaum, S., G. Rothreml and S. Kanduri, 2004. Selecting a cost-effective test case prioritization technique. Software Quality J., 12: 185-210.