Software Quality Test Method of Satellite Control System using Software Test Automation System

Cheol Oh Jeong, Byoung-Sun Lee, In Jun Kim, Yoola Hwang, Soojeon Lee
Unmanned Vehicle Systems Research Group
Electronics and Telecommunications Research Institute (ETRI)
Daejeon, Rep. of Korea
E-mail: {cojeong, lbs, ijkim, ylhwang, soojeonlee}@etri.re.kr

***Abstract*—Software (SW) quality management is a quality protection activity applied throughout SW development. SW quality management enables to implement software that is defect-free and meets development requirements. Quality management is performed through quality requirements, quality assurance methodology and quality testing. This paper shows SW quality test methods and results applied at the development stage of satellite control system. SW quality test was carried out by static test and dynamic test according to Verification-Validation (V-V) model proposed in SW engineering. The static test verifies that the source code is well implemented according to the requirements and design at the verification stage, and the dynamic test confirms in the validation phase whether the implemented product is running well. In order to efficiently perform the static test and the dynamic test, an SW test automation system is implemented. In this paper, we introduce a SW test automation system implemented for SW static test and dynamic test execution. Also, we show the results of the quality test that resulted in satisfying the requirements through potential errors or defects in the source code derived from static tests and requirements-based coverage results derived from dynamic tests.**

Keywords-Quality; Test; Satellite Control System (SCS); Static Test; Dynamic Test; Test Automation System; Coverage.

I. INTRODUCTION

The function and role of the software, which has been recognized as an auxiliary role of the hardware in the past, is gradually becoming a key factor that determines the completeness of the product. Accordingly, efforts are being made to ensure software quality from the development stage in order to secure software quality and efficient quality control. The satellite control system continuously monitors the state of the satellite during the operation of the satellite and periodically sends commands to the satellite to carry out a key function that enables the satellite to operate normally during the life of the satellite. Therefore, the SW of the satellite control system requires quality control from the development stage, so that the stability of the system should be secured. It is necessary to construct and operate a test automation system to efficiently perform the quality test of the SW requiring high reliability. The test automation system consists of a static test function that verifies the coding rule violation, code duplicate and complexity and a dynamic test function that checks the requirement based coverage required by the international standard [4]. Static testing is a technique for analyzing design documents and source code without direct execution of SW and examining whether coding rules, memory errors, duplicate code and code complexity are implemented according to international standards through coding standards, checklists, and so on [5]. On the other hand, the dynamic test is a technique to find SW defects by comparing the actual results with the expected results after executing the SW using test cases [6]. In this paper, we introduce a static test system and dynamic test system that verifies and validates the SW quality of the satellite control system. We present SW quality results through SW error and latent defect analysis derived from static test results, as well as coverage ratio analysis derived from dynamic test results.

The rest of the paper is structured as follows. In Section 2, we present an overview of SW quality test and SW test automation system including SW static test flow and dynamic test flow. In Section 3, we show the results of SW static test and dynamic test. In Section 4, we show the derived result of SW quality analysis from SW test results. Finally, in Section 5, we present conclusion and future work.

II. SW QUALITY TEST

A. Overview of SW Quality Test

SW quality control by V-V technique is performed to verify "Is the product being built correctly?" during verification phase and to validate "Is the right product being built?" during validation phase according to development and testing stages [1][3][4]. In the verification stage, the validity of the coding according to the specification is verified using the static test. In the validation stage, dynamic tests confirm whether the implemented SW satisfies the specification and requirements, as shown in Figure 1 [2]. Software test automation system was implemented and applied to verify and validate software quality efficiently.

B. Overview of SW Test Automation System

Software test automation system was implemented in Windows 7 environment, and it consists of Redmine (open source software) [8], Git (open source software) [9], Continuous Integration Automation Test (CIAT) and Web-based Quality Management System (WQMS), as shown in Figure 2. REDMINE is an issue management tool that manages issues that arise during both static and dynamic testing. GIT is a source configuration control tool that checks

 ISBN: 978-1-61208-624-8

the version state of committed SW. CIAT is Continuous Integration (CI) server that performs static and dynamic tests and WQMS is a web-based quality monitoring system that allows administrators or development departments to constantly monitor static and dynamic test results and to control test quality and manage schedules [3].

Figure 3 shows the functional flow diagram of the SW test automation system. Test verification of the SW begins with the developer or the coding authority connecting directly to the test automation system and committing the SW. Once the SW is committed, the tests are run automatically after checking for the latest version. After the test is over, the test results will be distributed via e-mail to developers, administrators, managers and others involved in SW development. If there are errors present in the test result, the error will be corrected and the quality of the development SW is verified through retest.

SW test automation systems can perform static and dynamic tests. Table I shows test items which can be performed using the SW test automation system. For static tests, one can test for coding rules, memory errors, duplicate code, and code complexity. For dynamic tests, one can test for coverage. Tests can be performed using a test automation system for SWs implemented in C # languages [5][6].

TABLE I. TEST ITEMS OF SW TEST AUTOMATION SYSTEM

Test	Test Item	Tool
Static Test	Coding Rule Verification	FxCop
	Memory Error Verification	Sparrow QCE
	Duplicate Code Verification	CPD
	Code Complexity Verification	CCM
Dynamic Test	Coverage Measurement	SquishCoCo

The static test is performed according to the static test flow, as shown in Figure 4.

We derive dedicated SW coding rules for the static test by selecting the coding rules of error with high frequency of occurrence and the rules of high importance. The selection criteria for coding rules are as follows.

Rule selection criteria
- Rules that can be linked to faults when faults are dynamic
- Rules that may cause maintenance problems
- Rules that developers can easily understand

Rule exclusion criteria
- Rules that apply only to Visual Basic
- Rules that apply only to C / C ++
- Rules that difficult and abstract to explain
- Rules that proof is tough

Selected rules applying to static test are shown in Table II [7].

TABLE II. RULES SELECTION BY SELECTION CRITERIA

Coding Rule Verification

Tool	Rule Description	Applied Rules
FxCop	Design Rules	26
	Globalization Rules	1
	Interoperability Rules	11
	Maintainability Rules	0
	Mobility Rules	0
	Naming Rules	0
	Performance Rules	1
	Portability Rules	1
	Reliability Rules	0
	Security Rules	20
	Usage Rules	16
Total Applied Rules		76

Memory Error Verification

Tool	Rule Description	Applied Rules
Sparrow	API Usage	1
	Design	1
	Forbidden	1
	Misuse	3
	Program Crash	4
	Quality	24
Total Applied Rules		34

The requirement-based SW dynamic test was performed by the procedure shown in Figure 5.

First, the test items were derived from the requirements specification analysis for the system development, and then the test cases were designed to enable dynamic test input and output verification. A test case was implemented in a script format using a macro program so that the test automation system and the test case can be operated in an interlocking manner. The dynamic test system was performed by linking the implemented test case with the test automation system. The test cases implemented in the script format were implemented considering the test procedures and the efficient recursive test. Dynamic tests were performed on all the user interfaces of the implemented system. The test case implementation uses AutoHotKey, a macro program tool, and the scripted task case performs the dynamic test in the order of Setup, Start, Output, and End as shown in Figure 6.

The Setup phase copies the necessary Data Base (DB) and data files before starting the test, and the Start step compares the actual results with the expected results while performing the scenarios in the test procedure based on the user interface. The Output stage compares the actual results with the expected results for the entire procedure to determine the success of the test. Finally, the End step executes the test case script as a procedure to initialize the test.

III. RESULTS OF STATIC TEST AND DYNAMIC TEST

A. Static Test

Code rule verification and memory error verification were performed on a total of 80,602 lines of source codes according to the static test procedure. The static test execution result is provided through the expression screen of the test automation system to manager and SW developers, as shown in Figure 7.

Static test results are shown in Table III. A total of 871 rule violations were detected in the coding rule verification including all violations from critical warning, critical error,

and error violation. A total of 3,340 rule violations were detected in memory error verification, including all violations from level 1 to level 3.

TABLE III. STATIC TEST RESULTS

Coding Rule Verification Results

Coding Rule	# of Detection
Design Rules	768
Globalization Rules	0
Interoperability Rules	0
Performance Rules	3
Portability Rules	0
Security Rules	0
Usage Rules	100
Total Violation Number	871

Memory Error Verification Results

Memory Error Rule	# of Detection
API Usage	0
Design	0
Forbidden	0
Misuse	0
Program Crash	170
Quality	3170
Total Violation Number	3,340

B. Dynamic Test

Dynamic testing was performed on a requirement based basis and was performed on a flight dynamics subsystem. A total of 20 test cases were derived for the 10 test items through the requirements analysis for the dynamic test execution. The errors/faults that are found during the dynamic test are delivered to the developer and the recursive test is performed to confirm the normal operation of the system through the retest after the error correction. The result of the dynamic test is provided through the display screen of the test automation system, as shown in Figure 8.

Dynamic test results of performing a requirement based dynamic test are shown in Table IV. Of the total of 20 test cases, 18 test cases have been confirmed to pass successfully.

TABLE IV. DYNAMIC TEST RESULTS

Test Case ID	Test Description	Pass/Fail
TFDS-01	Six-hour Orbit Prediction without Maneuver	Pass
TFDS -02	One-month Orbit Prediction without Maneuver	Pass
TFDS -03	One-month Orbit Prediction with Maneuver	Pass
TFDS -04	Two Stations Bias Calibration	Pass
TFDS -05	Orbit Determination Using Commercial Data without Maneuver	Pass
TFDS -06	Orbit Determination including Maneuver	Pass
TFDS -07	Real-time Orbit Determination	Pass
TFDS -08	Event Prediction (Eclipse & Sun-interference test)	Pass
TFDS -09	Station-Keeping Maneuver Planning and Reconstruction	Pass
TFDS -10	Station-Keeping Maneuver Preparation	Pass
TFDS -11	Station-Relocation Maneuver Planning	Pass
TFDS -12	De-Orbit Maneuver Planning	Pass
TFDS -13	Collision Avoidance	Pass
TFDS -15	Fuel Accounting by TOT Method	Pass
TFDS -17	H Management	Pass
TFDS -18	On-board Orbit Propagator	Pass
TFDS -19	Earth Acquisition	Pass
TFDS -20	Plot Services	Pass
TFDS -23	Data Management	Fail
TFDS -24	Process Management	Fail

IV. SW QUALITY ANALYSIS RESULT

The SW quality level of static test can be confirmed by the following formula.

Violation ratio (%) = # of violations/total line x100

When the SW quality level of static test is derived by reflecting the number of violations in Table III for a total of 80,602 lines of source code, the violation ratio of about 1% has calculated for the coding rule verification and about 4% for the memory error verification has calculated, as shown in Table V.

TABLE V. SW QUALITY LEVEL OF STATIC TEST

Classification	# of Source Code Line	# of Violation	Violation Ratio
Coding Rule Verification	80,602	871	1.08%
Memory Error Verification		3,340	4.15%

The SW quality of static test results can be considered to be high. It is possible to implement a more reliable SW by debugging the violation code with the recommended code format suggested by the coding rule format.

The SW quality level of dynamic test can be confirmed by the following formula.

Coverage ratio (%)=# of success cases/# of total test cases

When the SW quality level of dynamic test is derived by reflecting the number of success cases of 18 for a total test cases of 20, 90% of coverage ratio is calculated. The dynamic test was terminated when reaching a preset coverage target of 90%. The quality of the dynamic test results is considered to be high as the preset coverage target is achieved. As the preset coverage target was achieved, two test cases were not performed dynamic test and were marked as fail. By re-configuring coverage targets and performing source code debugging and dynamic testing, we expect 100% coverage to be achieved.

V. CONCLUSION & FUTURE WORK

This paper introduced the SW quality test method of SCS using SW test automation system implemented to ensure the quality and stability of software during the software development period. We also presented the static test results and dynamic test results as well as SW quality analysis results. This allows for the assurance of quality and stability of the development software which was complemented by identifying and modifying the software errors and defects during the development period. Through this, it is possible to analyze SW quality of satellite control system, and SW quality and stability are secured. When participating in similar satellite control system development program in the future, it is expected to apply the implemented software test automation system to ensure that the quality and reliability of mature software development is applicable.

 ISBN: 978-1-61208-624-8

ACKNOWLEDGMENT

This work was supported by the Space Core Technology Development Program of NRF-Ministry of Science, ICT and Future Planning [NRF-2014M1A3A3A03034729, Development of core S/W standard platform for GEO satellite ground control system]

REFERENCES

[1] National IT Industry Promotion Agency. nipa: SW Development Quality Management Manual. [Online] Available from: http://codedragon.tistory.com/attachment/cfile22.uf@233B814954902830161BD7.pdf, Mar. 17, 2018

[2] Electromics & Telecommunications Research Instittute, ETRI: Technical Document - SW Quality Management System of SGC Core Platform Development, Jan. 2015

[3] C. O. Jeong, B. S. Lee, I. J. Kim, Y. L. Hwang, and S. J. Lee, "Automated S/W testing system for Core S/W of Ground Control System", KOSST Conference, Jun. 2015

[4] C. O. Jeong, B. S. Lee, I. J. Kim, Y. L. Hwang, and S. J. Lee, "SW Quality Test Method of Satellite Control System", The Korean Society for Aeronautical and Space Sciences, Fall conference, Nov. 2015,

[5] C. O. Jeong, B. S. Lee, I. J. Kim, Y .L. Hwang, and S. J. Lee, "SW Static Test Method of Satellite Ground Control System", The Korean Society for Aeronautical and Space Sciences, Spring conference, Apr. 2016

[6] C. O. Jeong, B. S. Lee, I. J. Kim, Y. L. Hwang, and S. J. Lee, "Software Dynamic Test Environment Development of Satellite Ground Control System", The Korean Society for Aeronautical and Space Sciences, Fall conference, Nov. 2016

[7] C. O. Jeong, B. S. Lee, I. J. Kim, Y. L. Hwang, and S. J. Lee, "SW test automation system implementation for securing SW quality and stability", Int'l Conference on Software Engineering Research and Practice (SERP'17), Jul. 2017

[8] REDMINE: [Online] Available from: http://www.redmine.org/projects/redmine/wiki/Download, Mar. 17, 2018

[9] Git: [Online] Available https://git-scm.com/downloads, Mar. 17, 2018

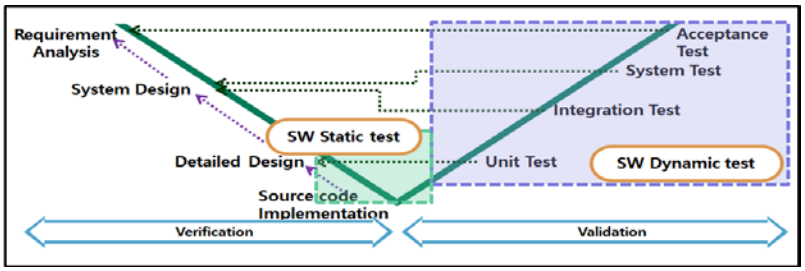

Figure 1. SW Quality Management System

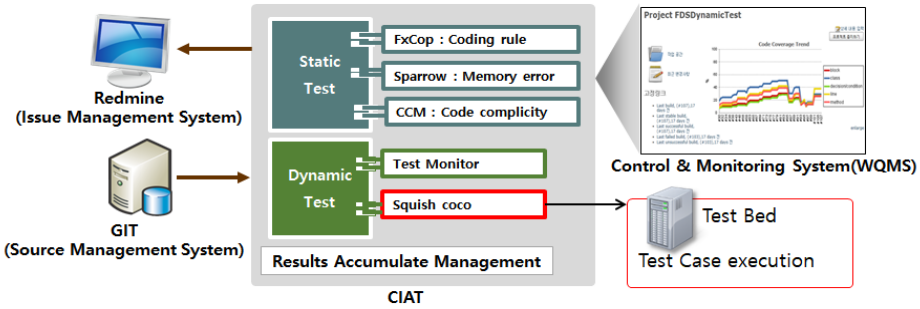

Figure 2. Test automation system configuration

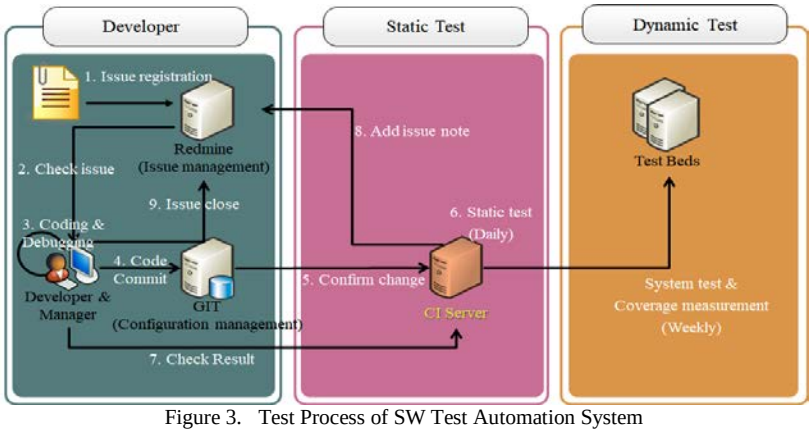

Figure 3. Test Process of SW Test Automation System

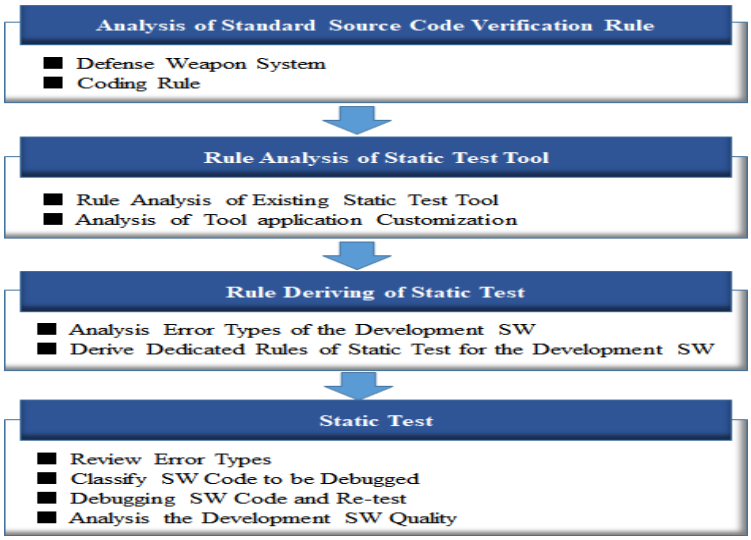

Figure 4. Static Test Flow

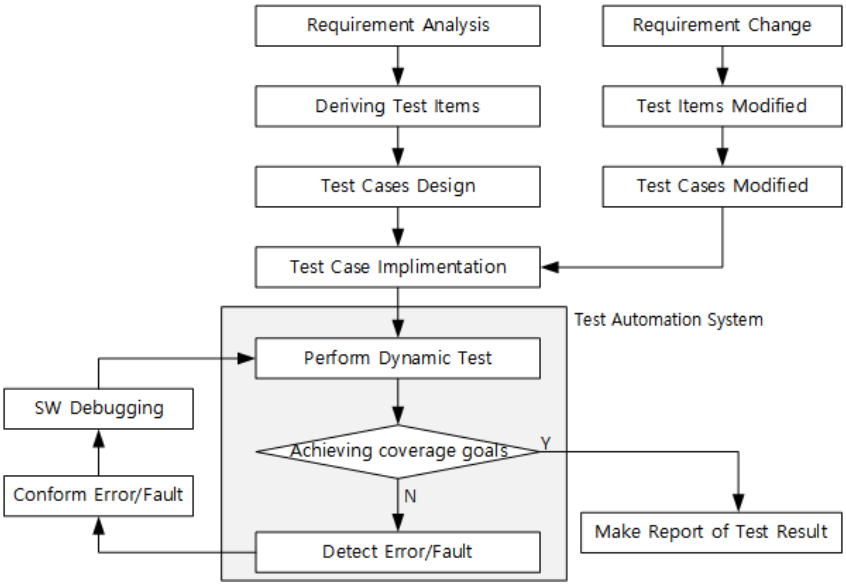

Figure 5. Dynamic Test Process Diagram

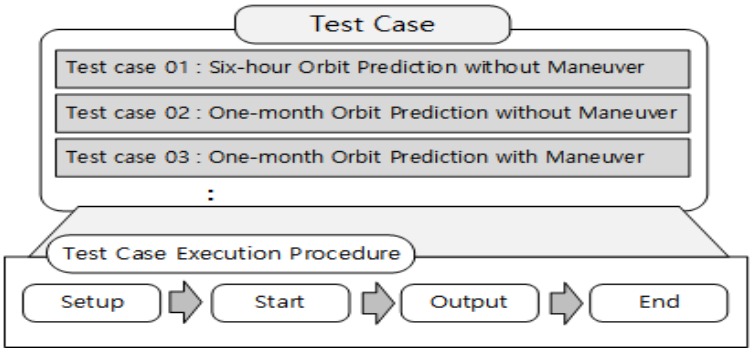

Figure 6. Test Case Script Structure

 ISBN: 978-1-61208-624-8

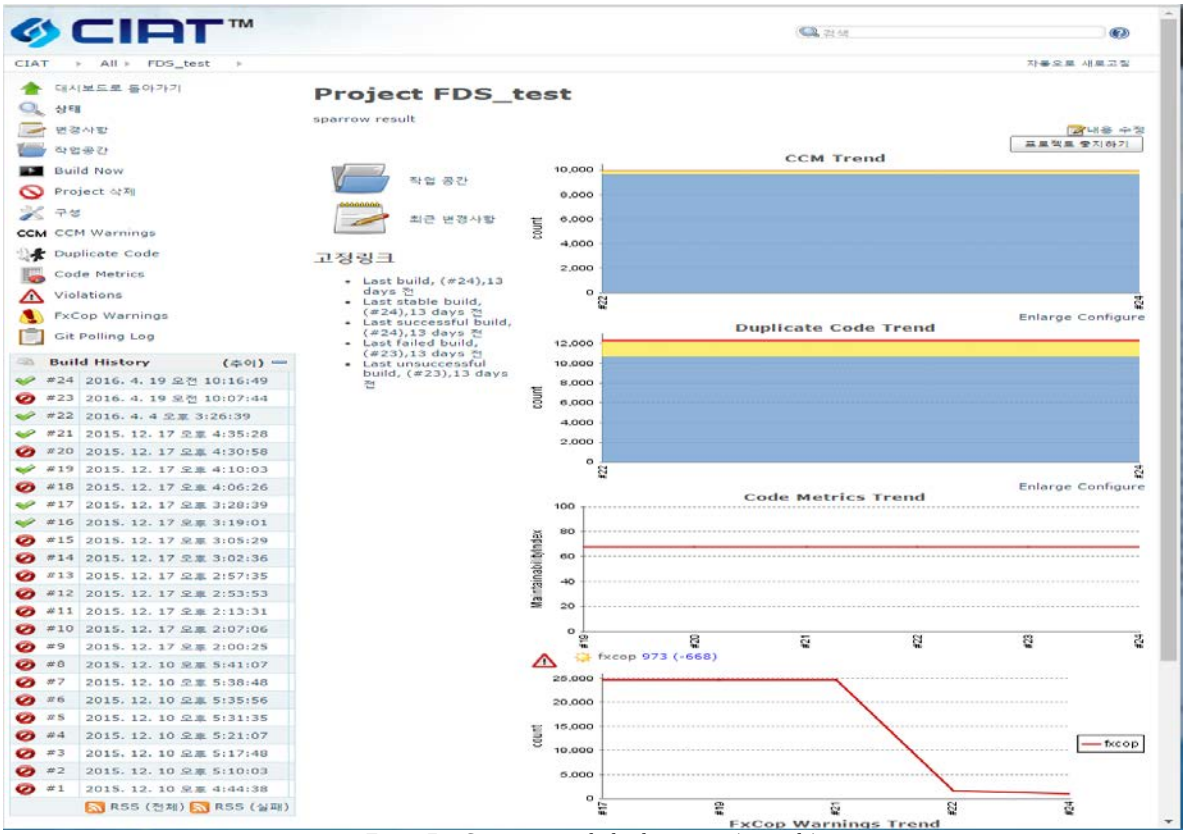

Figure 7. Static test result display screen (example)

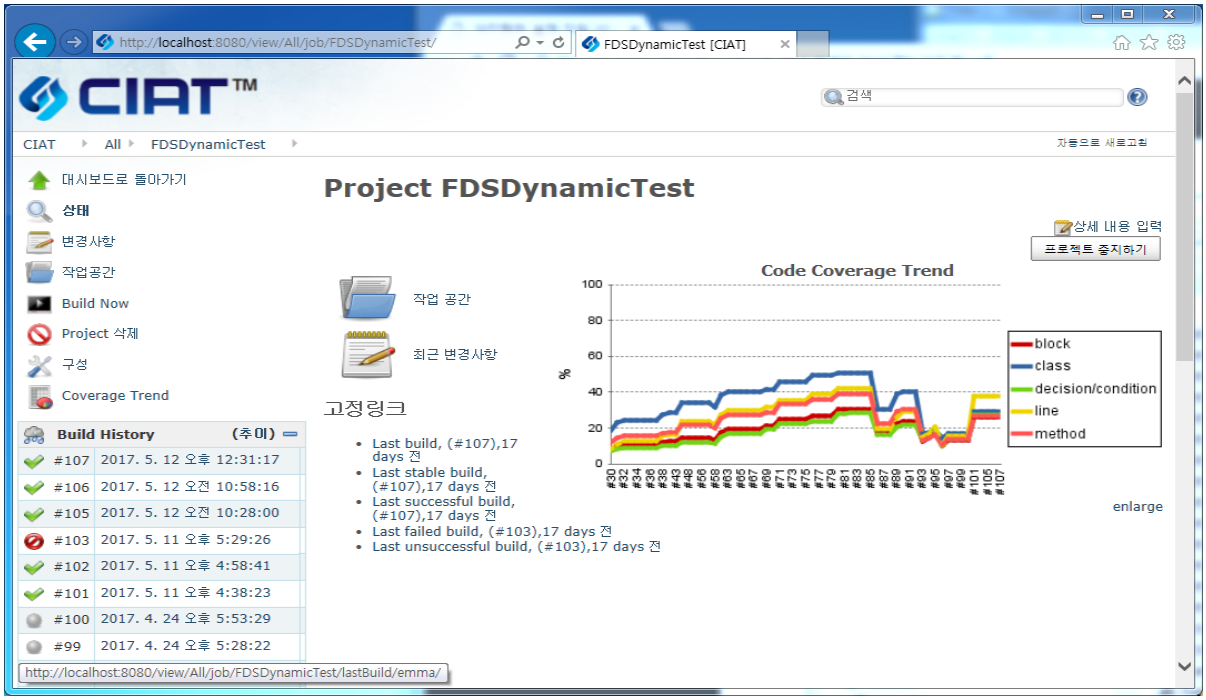

Figure 8. Dynamic test result display screen (Example)