Proposal of Automated White-Box Testing Tool for S/W Safety Guarantee in Railway System

Hyunjeong Jo, Jonggyu Hwang, Jonghyen Baek, Sehchan Oh, Kangmi Lee and Yongkyu Kim
Train Control & Communication Research Division, Korea Railroad Research Institute (KRRI)
Uiwang-city, Gyeonggi-do, Korea

Abstract
The railway system, especially railway signaling system is being converted to the computer system from the existing mechanical device, and the dependency on software is being increased rapidly. As the software becomes more complicated for the intellectualization and automation in accordance with the development of computer technology, the relative importance took by software in the railway system is being increased further. Though the size and degree of complexity of software for railway system are slower than the development speed of hardware, it is expected that the size will be grown bigger gradually and the degree of complexity will be increased also. Accordingly, the validation of reliability and safety of embedded software for railway system was started to become influential as the important issue. Accordingly, various software test and validation activities are highly recommended in the international standards related railway software. In this paper, we presented an automated white-box testing tool composed of data flow and control flow analysis module for railway system software, and presented its result of implementation. We developed the data flow and control flow analysis tool estimating test coverage as an important quantitative item for software safety guarantee in railway software.

1. Introduction
Recently, safety requirements of railway system software became the international standards in accordance with IEC 61508 and IEC 62279 [1][2], and with the enactment of the Railway Safety Act, the atmosphere requiring various software tests and validation activities required by relevant international standards in relation to these railway systems are being created in Korea also [3]. However, software validation has been mainly dependent on the documents for development process so far, and the qualitative analysis according to the test is made to only a fraction of it [4][5]. And, the criteria on test and validation of railway system software in Korea in accordance with international standards or any study on technologies coinciding with them are in the actual situation of early stage only which is started just now [6]. Therefore, it is our situation where not only validation of the document on safety activities being required by international standards in relation to the railway system software, but also concrete development of technology and judgment criteria for verification to cope with analysis and validation through software tests are highly required. Especially, in case of the validation on vital railway system software most of whose SIL (Safety Integrity Level) grades are classified as 3 or 4, white-box testing composed of data flow analysis and control flow analysis module which can draw quantitative results among railway software validation items being required by international standards is defined as ‘HR : Highly Recommend’ condition in the relevant international standards [1][2]. Results through data and control flow analysis of railway software can be utilized as the code-based test coverage, and we may see that those are very important items enabling to measure the code coverage being applied for the software validation in accordance with software safety grades in other vital fields such as the aviation and nuclear power plant, etc. also [7-11]. Vital railway system which is connected with human lives directly must secure the safety as the embedded system with large-scale software, and for this purpose, application of the code coverage like this is inevitable for the validation of railway software. Thus, this paper designed and developed the automated tool for data flow and control flow testing to validate this software for railway system. This paper is composed in the manner that the chapter 2 shows the testing of software for railway system, and chapter 3 shows the result of implementation on the design and development of the automated tool for data flow and control flow testing on the railway software, and finally the chapter 4 forms its conclusion.
2. Software Testing for Railway System

The embedded software for railway system can lower the quality cost by verifying the bug through testing process from its early stage of development, and in addition, the software testing process is required essentially also at the aspects of validation process and maintenance after completion of development. Although it is necessary to conduct testing through automated tool since the software for railway system is the vital software depending on the hardware and requiring high safety, only testing tools for partial items with the target of software for general embedded system for industrial use are being introduced so far.

Although safety-related matters in relation to the railway system software are being required by international standards such as IEC 61508 and IEC 62279, but testing tools for the general control system for industrial use are not satisfying the requirements of railway-related international standards partially. Thus, to develop the automated tool for testing software for railway system in this paper, we prepared a guideline for the assessment on safety of railway software through analysis on international standards in relation to the safety of railway software like Figure. 1, and analyzed and drew items which are essentially required to be automated among testing items in this guideline. One of the things required by international standards as highly recommended validation item is the very white-box testing on the vital railway system software. The white-box testing tool, that is, data flow and control flow analysis module measure and report the coverage in accordance with the area of source code and whether the branch statement and conditional statement, etc. were executed or not. Coverage is being used as the representative scale with which software can be validated quantitatively in other industrial fields too, and further to this, it makes us guess the software quality also. In general, various kinds of coverage are presented in the software test field, and among them, the control flow analysis module for railway software developed in this paper provides the statement coverage, branch coverage and modified condition/decision coverage. In accordance with IEEE Std. 1008-1997 standard, it is mentioned that entire software must be validated by the test case in the unit test stage, and the statement and branch coverage of whole software codes must be satisfied by the test case [7]. To measure this test coverage of unit test performance, this standard recommends using the automated means. In addition, IEC Std. 60880-2006 standard also recommends using the methods such as the statement and branch coverage, etc. to validate software implementation stage [10]. And, the statement and branch coverage are already utilized widely in the actual industrial fields also.

And, RTCA/DO-178B which is the standard norm of aviation industry classified software grades according to the importance in accordance with the kind of accident identified through assessment on the system safety, and defined the code coverage requisite in accordance with its grade [11]. According to [11], it is defined that the higher the grade of importance to which the safety is required,
the modified condition/decision coverage must be satisfied, and the cases proving its applicability are presented also [12][13]. Thus, modified condition/decision coverage must be satisfied in the vital railway system software such as the electronic interlocking device also which is classified as SIL grade 3 or 4.

Data flow testing tool extracts data flows by analyzing the definition/information on use from the source code. If the test is performed by using the information extracted by analysis module, it evaluates how much test data generated automatically and those input by the user satisfy the data flow coverage. Data flow testing tool evaluates the quality of test performance on the basis of definition and use of specific variables within the data program code. As explained in the above, data flow testing tool largely provides coverage information for the All-Def suitability criteria, All-Uses suitability criteria and All-DU Paths suitability criteria. All-Def suitability criteria means that, the statement such as \( i = 0; \) or \( i = a + b; \) which is the definition of variables used within the specific function can be existed in several places within the function. The criteria evaluating whether definition statement in several places was executed become the All-Def. For example, if only two among three statements were executed at the actual program execution although a certain variable was defined three times within the program code, it is judged that 100% of the All-Def was not satisfied, but only 66% was satisfied.

The case for All-Uses is the same. There may be several places where the use of specific variable occurs within the program code. It judges whether all of these uses were executed during the actual program execution. For example, if only three places were performed when performed actually although a certain variable was used five times within the code, only 60% of the All-Uses were satisfied. All-DefUse Paths judges whether all of these were executed by tracing all of the paths where a certain variable was defined and used within the program. For example, if only six among them were executed when executed actually although whole paths where variables were defined and used were found out to be 12 places when analyzed, it means that only 50% of the All-DefUse Paths were satisfied. On the other hand, the control flow analysis module consisting of the automated tool for test coverage of software for railway developed in this paper can provide all of the statement coverage, branch coverage and modified condition/decision coverage. Detailed contents for each of the coverage are as follows.

- **Statement coverage**
  Block is the set of consecutive statements and it means the area being executed sequentially without any branch or repeated statement. If any branch is occurred, a new block is created. The statement coverage consisting of these blocks is the one obtained in the ratio of the statement performed actually in comparison with the statement within the block area being existed in the test target source code.

- **Branch coverage**
  Branch coverage evaluates whether the control flow according to the branch statement such as the “if” statement was performed variously like the true case and false case. If it is assumed that there are 6 determining conditions within the program according to the branch statement, the branch coverage will become 4/6, 66% if 4 conditions among them are performed. Normally, branch coverage should be performed at least more than once for whole results possible to be occurred in all of the decision of program.

- **MC/DC (Modified condition/decision coverage)**
  Modified condition/decision coverage evaluates not only determining conditions of the branch statement, but also whether they were performed variously for each of the conditional statement consisting of the branch statement. If you see the program code in Figure 2, branch statement is existed in the third line. When showing the determining conditions of the branch statement in the truth table, it will be as the following Table 1.

```c
int f (int a, int b) {
    int c = a + b;
    if (a > 10 && c > 50) {
        printf("c = %dn", c);
    } else {
        printf("b = %dn", b);
    }
}
```

<table>
<thead>
<tr>
<th>State</th>
<th>Condition1 (a&gt;10)</th>
<th>Condition2 (c&gt;50)</th>
<th>Condition1&amp;&amp; Condition2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>2</td>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>3</td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>4</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

Table 1: Truth table of the operator

Figure 2: Example program of the MC/DC
Case for the modified condition/decision coverage is the combination where each of the condition satisfies the determining condition. Because the states satisfying the determination of condition 1 are Nos. 1 & 3, and the state for condition 2 is fixed as true, which means that whole conditional statements are determined to be true/false in accordance with the true/false of the condition 1. The states satisfying the determination of condition 2 are Nos. 1 & 2. Thus, the states satisfying determining conditions of the condition 1 and condition 2 are Nos. 1, 2 & 3. In case of the modified condition/decision coverage, if the state of each condition such as 1, 2 and 3 is performed, the coverage will become 100%.

Commercialized overseas tool which measures the code-based test coverage automatically is existed, and the vital aviation field utilizes this tool [14]. However, despite that it is the aviation field which has more advanced technologies in developing and testing software than other industrial fields, it has not developed any automated test coverage tool suitable to be applied to the aviation field with its own domestic technology yet. Accordingly, in the actual application, it is difficult to utilize it in the industrial site because the exclusive guideline to software validation tool must be enacted as well as the support of customizing technology is impossible. In addition, automated foreign test coverage tool has demerits such that it cannot measure the above-mentioned coverage effectively at a time, and the tool suitable for the measurement of each of the coverage must be utilized separately.

Thus, this paper enabled customizing technologies to be supported by developing its own data flow and control flow analysis module which is the main item measuring the test coverage among validation items (coding rule inspection, metric support, control flow analysis, data flow analysis, boundary value analysis, etc.) required in the software field quantitatively so that it can be utilized in the railway industry field, and it was manufactured in such a manner that it can have the compatibility with any tool measuring other validation items also. And, above all, since it has the strength possible to measure diversified coverage effectively at a time by using this tool only, we may anticipate high practicality.

3. Design & Result of Implementation of Data Flow and Control Flow Analysis Module

Like this, train-related international standards require the validation of software highly recommended, and since one of the assessment items easy to automate and analyze the testing according to it is the very data flow & control flow testing on software, we designed and developed the automated tool for data flow testing on the railway software as a supporting tool.

3.1 Implementation on the design & development of data flow testing module

Table 2: The list of functions for data flow testing module

<table>
<thead>
<tr>
<th>Function name</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program analysis and CFG generation</td>
<td>Extracts lists of function being implemented in the program and the highest API function. Control flow graph by function becomes the base model of test, and the purpose of test process is to perform the maximum majority parts of the maximum majority control flow graphs. Utilizes to generate test scripts and test data by generating the call graph between each function. Stores analyzed contents and CFG at the repository.</td>
</tr>
<tr>
<td>Generation of test cases</td>
<td>Generates test cases by analyzing definitions and uses for each data used at the API. Stores generated test cases at the repository.</td>
</tr>
<tr>
<td>Test performance</td>
<td>Stores results of its performance at the repository by performing the generated program by test case after consolidating and compiling actual test target sources and source codes generated for the test, etc.</td>
</tr>
<tr>
<td>Report on results of data flow testing performance</td>
<td>Reports after analyzing definitions – uses by data.</td>
</tr>
</tbody>
</table>

*CFG: Control Flow Graph, API: Application Programming Interface*
The Table 2 is the list of functions for data flow testing module. To implement each function of the data flow testing tool, it is consisted of detailed modules as follows.

- Data flow analyzer: Analyzer which analyzes the definition of data and information on use
- Coverage calculator: Module which evaluates whether it was reached or not reached while the test data is being executed and calculates its coverage based on the definition of data and information on use
- UI(User Interface): Module which performs the data flow analyzer and test, and provides the result of analysis and test result to the user visually

The operation scenario of data flow testing is based on the process such as test case preparation, performance and the report on results. The main scenario for use is that the data flow testing tool performs whole processes automatically, but reinforces the test gradually by making the tester intervene as to the parts not tested automatically. The following Figure 3 is the operation scenario of the data flow testing tool.

![Figure 3: Data flow testing scenario](image)

Test preparation stage is largely consisted of the test target selection and test case preparation. The tester shall prepare the C/C++ source code possible to be compiled. Data flow testing tool analyzes which functions and types are defined after receiving the source file as input, and then provides the tester with the list of function which will become the test target. On this basis, the tester selects its function to apply the data flow testing. Selected function analyzes the defined position of variable and the used position information by the analyzer, and extracts requirements to satisfy the All Defs/Uses and All DU Paths. Test data automatic generation module generates test data for accomplishing data flow coverage automatically, and in this case, the user can add test data directly so that he/she can test logic meaning kept by the data.

Test performance stage is consisted of the work to make test programs possible to be performed and the work actually performing it by using prepared test cases. Test program possible to be performed is the program in the form of combining source program which is the target of the test with the program which calls it actually and extracts the result of performance. This is generated in the data flow testing tool automatically. Data flow testing tool performs test cases prepared at the test preparing stage automatically, and accumulates results of performance at the internal repository based on the analyzed requirement data. In this process, most of the works are automated by the data flow testing tool, and there is almost no part to be intervened by the tester directly. The last stage is the work analyzing the test result performed so far, and evaluating whether the test is enough or not. After completing the testing, the tester can verify test results by function, by requirement and by test case, and can proceed the test by adding meaningful data possible to satisfy the requirement again after identifying the requirement not satisfied through test results.
Entire screen of program is consisted of the menu, project screen and screen by function. Basic menu is consisted of the menu item to perform functions such as project generation, opening, work performance and product information. Project screen is consisted of the screen showing the list of currently working project and source code function included in the project. It shows the source included in the project and the list of function defined in the source. Since the information on data flow shows definitions/information on use for whole variables after extracting them, if a series of data flow information is provided all at once, it becomes the level impossible to be grasped by the user. Therefore the screen possible to follow the definition and information on use by each function is provided on the basis of the graph in the unit of function. In this case, since the information is not seen at first sight simply with the figure only, the graph emphasis function is provided in accordance with the selection of user.

Table 3: Detailed module by function

<table>
<thead>
<tr>
<th>Function</th>
<th>Implementation module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program analysis and CFG generation</td>
<td>Program Analyzer, CFG Generator, Code Rewriter, Data Flow Analyzer</td>
</tr>
<tr>
<td>Generation of test cases</td>
<td>Test Data Generator</td>
</tr>
<tr>
<td>Test performance</td>
<td>Test Driver Generator, Builder, Execution Engine</td>
</tr>
<tr>
<td>Report on the result of data flow testing performance</td>
<td>Data Flow Coverage Evaluator, Graphic User Interface</td>
</tr>
</tbody>
</table>

Like All Defs/All Uses, All DU Paths provides the information using graphs also. Provided that, since All DU Paths are indicating the paths, the variable selected by user provides the information on path passed through together with the defined node. Of course, the information shown in accordance with the selection of user also provides the screen emphasized up to the path not in the unit of node. The UI of data flow testing tool is prepared in the same Java language, and the graphic library (UI toolkit) is implemented by using the swing which is the standard UI tool of Java. Data flow testing tool supports the operation system of Microsoft Windows and is operated in the form of independent application. The function of data flow testing is consisted of the following modules, and each module is consisted of the relation like the Table 3.

Program analysis and CFG generation function is consisted of the program analyzer, CFG generator, code rewriter and data flow analyzer modules. Program analyzer extracts the list of function and list of variable by reading C/C++ source codes. Extracted code information is delivered to the code rewriter and CFG generator. Code rewriter inserts probe codes to measure the All-Defs/Uses and All-DU Paths coverage when it performs the test target program. CFG generator generates CFG by analyzing the statement and expression information within the function, and extracts a list of input variables which affect the performance of function, and that of output variables which are affected by the function performed. Data flow analyzer extracts the position information and path for the part where the variable used in function is defined and the used part. Since extracted information is used at another module, it is stored at the common repository which is the common storing place.

Test case generation function is consisted of the test data generator module. Test data generator module generates test data for the input variables recorded at the CFG generator module. Test data generator module extracts input values for each variable by using the domain division technique based on the constant value which affects the control flow, and generates test cases between each variable by using the simple/pairwise combination method. Generated test case is stored at the common repository which is the common storing place. Test performance function is consisted of the test driver generator, builder and execution engine modules. Test driver generator module generates test driver codes automatically on the basis of information on function stored at the common repository. Builder generates a binary possible to perform the test by compiling and linking source codes where test driver codes and probe codes are inserted. If the preparation stage for test performance is completed, the execution engine module performs its test by using the test case generated at the test data generator module, and extracts covered results and output data after performance.
Data coverage evaluator module calculates figures of the performance flow extracted by the execution engine and stores them at the common repository based on the All-Defs/Uses coverage and All-DU Path information. GUI module outputs performance flows and input/output data by each test case stored at the common repository to the screen. The development of data flow testing tool utilizes a program analyzer module which analyzes C/C++ source code, a test driver generator which is the test driver code generator, a builder which compiles and links the original code and driver code, an execution engine which extracts results by performing the prepared test case and the common repository which is being used by each module commonly. In case of the module by function, the data flow analyzer and data coverage evaluator were designed for the data flow testing tool.

Figure 4: All Defs/All Uses data flow analysis

Figure 4 shows the requirement which must be reached to satisfy the All Defs or All Uses coverage in a specific function in the unit of function from the viewpoint of graph shown in the left. For example, as for the len variable defined at the #41 block, the test data shall be generated so that more than one of the #41, #40 and #58 blocks can be performed to satisfy the All Defs coverage, and the test data passing through all of the #41, #40 and #58 shall be made to satisfy the All Uses coverage. And Figure 5 shows whether test data had satisfied the analyzed All Defs/All Uses requirements after performing the test. Reached node is indicated as 'T'. In accordance with the result of test performance, additional test data for the area not addressed by automatically generated test cases can be generated. The result can be seen by accumulating whether whole test data were reached or not, or it is possible to verify whether they were reached by test case.

Figure 5: All Defs/All Uses coverage
Although whether it was covered or not could be verified in the unit of each block and path in the performance graph, the data flow coverage for the whole performance and data flow coverage by function are provided for the quantitative coverage figure in the coverage summary as the following Figure 6.

3.2 Implementation on the design & development of control flow testing module

Control flow analysis module is a module creating the graph by analyzing source codes, and finding the existence of error by generating and performing test cases possible to satisfy the statement coverage, branch coverage and modified condition/decision coverage. Control flow analysis module has the program analysis, test case generation, test case performance, report on result of test performance and test scenario, and has the data structure and repository showing the program analysis and information on test result. Control flow analysis module has main functions such as the creation of control flow graph, automatic creation of test data, measurement of the statement coverage, branch coverage and modified condition/decision coverage. Following Table 4 is the one organizing an explanation on main functions. Operation scenario of control flow analysis module having these functions is largely progressed as preparation, performance and report on result of test case. Main scenario for the use of control flow analysis module is that it shall perform the entire process automatically, but the test will be reinforced gradually through intervention of the tester as for the part not tested automatically. Following Figure 7 is the operation scenario of control flow analysis module.
Table 4: List of function for control flow analysis module

<table>
<thead>
<tr>
<th>Name of function</th>
<th>Description on function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program analysis and CFG generation</td>
<td>Extracts lists of function being implemented in the program and the highest API function. Control flow graph by function becomes the base model of test, and the purpose of test process is to perform the maximum majority parts of the maximum majority control flow graphs. Utilizes to generate test scripts and test data by generating the call graph between each function. Stores analyzed contents and CFG at the repository.</td>
</tr>
<tr>
<td>Generation of test cases</td>
<td>Generates test cases for the statement coverage, branch coverage, MC/DC(modified condition/decision coverage). Stores generated test cases at the repository.</td>
</tr>
<tr>
<td>Test performance</td>
<td>Stores results of its performance at the repository by performing the generated program by test case after consolidating and compiling actual test target sources and source codes generated for the test, etc.</td>
</tr>
<tr>
<td>Report on the result of control flow testing performance</td>
<td>Reports the statement coverage, branch coverage, MC/DC(modified condition/decision) coverage.</td>
</tr>
</tbody>
</table>

Test preparation stage is largely consisted of the selection of test target and test case preparation. First of all, the tester should prepare the C/C++ source code possible to be compiled. Control flow analysis module provides a list of function which will be the test target to the tester after receiving source files as input and analyzing which function and types are defined. Tester extracts the test case suitable for the corresponding function automatically by using the automatic test data generation function of control flow analysis module. Test performance stage is consisted of the work to be carried out actually by using the work to make test programs possible to be performed and the prepared test case. This will be created by the control flow analysis module automatically. Control flow analysis module performs test cases prepared in the test preparation stage, and stores results of performance in the internal repository. In this process, most of the works are automated by the control flow analysis module, and there is little part to be intervened by the tester directly.

The last stage is the work evaluating whether the test is sufficient or not by analyzing test results performed so far. Control flow analysis module evaluates whether the test is sufficient or not on the basis of the figures for statement coverage, branch coverage and modified condition/decision coverage. Tester should test more if the target figure for results of test performance is not reached, and must reinforce the test case so that the target figure can be reached. Control flow analysis module provides the code information performed actually in accordance with the figures for statement coverage, branch coverage and modified condition/decision coverage and the input value as its result of performance. Tester can reinforce the test case to test those parts after checking the statements and branches which will not be carried out actually on the basis of the information to be provided by the control flow analysis module.

Figure 8: Entire screen of program of control flow analysis module
Challenge H: For an even safer and more secure railway

Entire screen of the developed program for automated control flow testing tool is consisted of the menu ①, project screen ② and the screen by function ③ like Figure 8. Basic menu is consisted of the menu items to perform the function such as the creation of project, opening, work performance and product information. Project screen is consisted of the current project under working and the screen showing the list of source code function included in the project. This tool analyzes source codes by using the basic menu ①, and it can extract the control flow information internally in this process. And, the call graph between functions of source code shows associated functions in the unit of group by selecting their groups.

- Analysis on the source program
  It analyzes source codes by using the basic menu of program. This process extracts control flow information internally.

- Analysis on the source program: Call graph
  Call graph shows the functions having a relative relation in the unit of group after selecting the group. If any graph list is selected first, it shows the call relation of all the functions belonging to the corresponding group.

- Analysis on the source program: Control flow graph
  Control flow graph is existed by one for each function. CFG is expressed in the form of graph like by combining the node and edge expressing the flow of performance. The node of CFG shows the set of statement without branch, and the edge shows the relation of performance order between the statements. CFG expresses the external function being called in a special form, and provides it so that it can go and call CFG of another function by using it.

- Generation of test cases
  Test case is generated by using the basic menu. The proceeding situation is shown since it requires a lot of time to generate test cases.

- Test performance
  Test case is performed by using the basic menu. The proceeding situation is shown on the basis of name by each case since it requires a lot of time to perform test cases.

- Verification on the result of test performance: Coverage
  Result of test coverage largely provides whether figure information and codes were performed or not. Whether codes were performed or not is provided by recording the number of performance times at the node and edge on the basis of CFG.

- Verification on the result of test performance: Input/output
  In this function, it is possible to verify the result of performance and input/output values of test case. The result of performance is considered as success if the function was performed normally and as failure if the program was discontinued during the performance.

Figure 9: Screen for test execution of the automated control flow analysis module
Challenge H: For an even safer and more secure railway

On the basis of the design development process like this, the screen for execution result of the automated control flow testing tool for railway system software whose manufacturing was completed actually is like Figure 9. When you click “View call graph,” the relation between each function is shown in the graph. The list of whole area variables are shown in the right side panel and it is indicated as the relation of definition or use in a certain function. And also it expresses performance flows of function in a graph. Right side panel provides information which analyzes MC/DC conditions for branch statements defined in the function with the truth table. In case of the second branch statement, it is consisted of two conditions and there are three cases to satisfy the MC/DC. Then test is performed with the prepared test case. Test script is the test performance unit and is consisted of test cases. The number of success, failure and warning are shown in the screen during the course of performance, and the proceeding situation is displayed.

Conclusion

Recently, the dependency of railway systems to computer software is being increased rapidly in accordance with the development of computer technology, and the high reliability and safety are required in the vital railway software according to this technological development. Accordingly, international standards in relation to the railway system software require the testing and validation of software as mandatory matters, and for the accurate analysis on the white-box testing tool which can draw quantitative results of the test coverage among software validation items being required by these international standards, we presented an automated tool developed in this paper. This data flow and control flow analysis makes the result of All Defs/All Uses, All DU Paths, statement, branch coverage and modified condition/decision coverage, etc. which are the code-based test coverage drawn, and this test coverage is the very important validation item which is being applied in the vital aviation and nuclear power plant, etc. already to validate software in accordance with the software safety grade. First of all, functional design of the software control flow analysis tool developed exclusively for the railway system was explained, and its result of implementation was shown concretely.

The tool for data flow testing like this expresses results of analysis on All Defs/All Uses, All DU Paths data flow as the unit of function from the viewpoint of graph so that they can be grasped easily in position of user. Although whether it was covered or not could be verified in the unit of each block and path in the performance graph, the data flow coverage for the whole performance and data flow coverage by function are provided for the quantitative coverage figure in the coverage summary. Also, control flow analysis module, which is the tool performing test coverage measurement for validation of software for railway developed in this paper like this automatically, expressed results of analysis on the statement of source code, branch and modified condition/decision coverage in the control flow graph and function call graph so that they can be grasped from the position of user. And, by showing the result of each measurement in the final figure too, it was manufactured so that the comparative analysis can be grasped easily also.

Data flow and control flow analysis module which is the automated white-box testing tool for this railway software test coverage is the tool to be utilized largely for the validation of software for railway system in the prospective customers such as the railway operating agency, etc. basically. At the same time, it is considered that its degree of utilization will be sufficiently high in the development process of software for railway-related industries, and for the unit or integrated test stage of corresponding product developed. Of course, ultimately, the efficiency at the actual industrial sites of railway field will not be maximized until the result of measurement on code-based test coverage presents judgment criteria which can be matched to the software safety integrity level (SWSIL) grade desired by the user. Thus, we have a plan to study further each of the assessment criteria on the result of coverage measurement so that it can be utilized at the actual railway industrial sites. If this tool is used widely in the software validation and development stages, it will be able to contribute greatly to the securement of safety and reliability by preventing any error in vital railway software through it before anything happens.

References