High-Reliability Design of Software Applying Model-Based Development

Y. KOBAYASHI H. KATO K. TOYOZUMI

As automobiles have increasingly come to utilize electronic control in recent years, the streamlining of a software development system has become essential for providing high-reliability software in a short development period. As a means of securing reliability and a reduced development period, investigation was made on applying model-based development, which has received much attention in recent years, to the development of mass-produced software of JTEKT. This report describes the building of a software development process with the model-based development applied, through which the development period is expected to be shortened by about 30% while maintaining reliability equal to or higher than that of the conventional software development.

***Key Words**: model-based development, electronic control unit, software, high reliability, reduction of development period*

1. Introduction

In line with the trend of rapid computerization of vehicles in recent years, the number of source code lines of Electronic Control Units (ECU) software has increased and its specifications have become complicated. In addition, in response to requests for shortening the development period, time is required to address compliance to standards such as Automotive Software Process Improvement and Capability dEtermination (Automotive SPICE), Capability Maturity Model Integration (CMMI), and others, which makes it difficult to shorten the development period. Under such circumstances, streamlining of software development systems has become essential in order to achieve shortened development period while securing reliability. A model-based development has been attracting attention as one means of achieving this.

A model-based development is a technique to carry forward the development by creating models from the function development stage and verifying the behavior and performance. To mention two principal advantages of this technique, one is that functions are able to be designed and verified by simulatable models from the upstream process. This can reduce the outflow of bugs to the downstream process, reduce rework to the upstream process, and eventually shorten the development period. The other advantage is that the model is able to be reused to software development. Ambiguity of requirements is able to be eliminated, and at the same time software is able to be automatically created from the model, and this technique can thereby contribute to securing reliability and shortening the development period[1].

When directing attention to the condition of JTEKT, an increase of software functional requirements has increased the number of source code lines by five to ten times in the last ten years. In connection with this, resources necessary for software development keep increasing, too. In addition, requirement specifications from functional development to software development have sometimes caused rework due to misunderstanding caused by the ambiguity of the content, and there has been room for improvement in development efficiency.

In order to solve the problems at JTEKT as described above, investigation has been made on the application of the model-based development to mass-production development. In this report, a development process which has been built to meet the JTEKT development patterns is shown, and it has been confirmed that this process has enabled shortening of the development period by securing high reliability in the design stage and streamlining operations in each relevant process with reliability equal to or higher than that of the conventional software development maintained.

2. Construction of Software Development Process

In this chapter, the development process applying the recently built model-based development and the work contents implemented in each process are described. In addition, maintenance of reliability compared to the conventional development process is confirmed.

2. 1 Development Process Built

Figure 1 shows the conventional software development process[2], **Fig. 2** shows the software development process investigated applying the recent model-based development, and **Table 1** shows the work contents in each process.

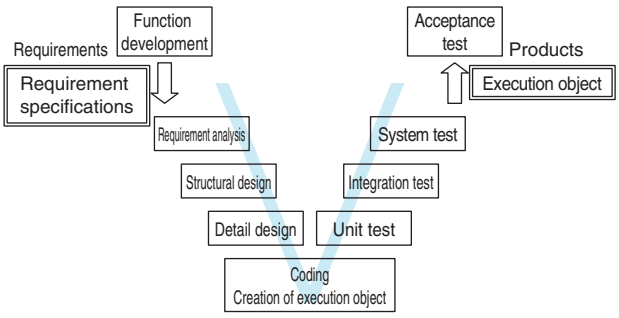

Fig. 1 Conventional development process

2. 2 Difference in Process from Conventional One

In this section, discussion is made on differences between the conventional development process and this subject development process. However, advantages of each process will be described later in **Chapters 3** and **4**. First of all, requirements for software development are test patterns whose model has been verified for functional validity and its judgment criteria, in addition to the simulatable model. Next, a Model In the Loop Simulation (MILS) test has been added to the design stage as a new process. This process verifies one functional unit of the model as a target. To the function check test pattern received as a request, each test pattern of white box/black box is added and the model is verified in further detail. By this, the reliability is secured before generating an execution object, and rework is reduced. Next, a Processor In the Loop Simulation (PILS) test is substituted for the conventional unit test. This process targets verifying

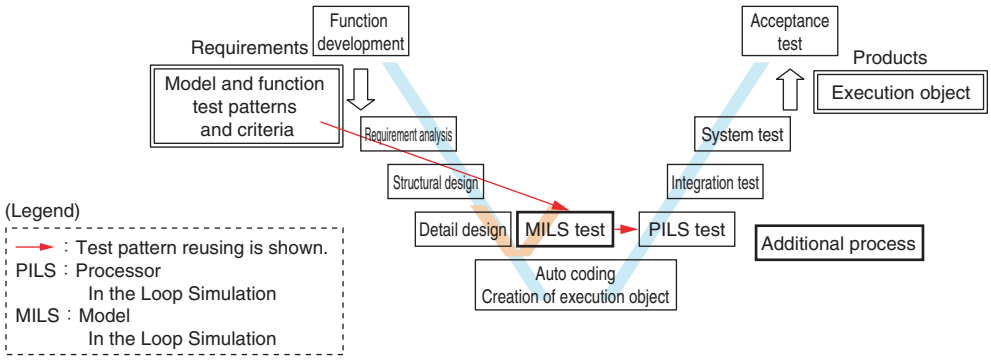

Fig. 2 Development process applying model-based development

Table 1 Contents of main work in each process

Process	Conventional	Recent study
Requirement analysis	Analysis of received requirements, requirements classification and securing of traceability	Same as before
Structural design	Software structure and layout design, interface design between modules	Software structure and layout design, interface design between received model and software
Detail design	Design of function achieving method for single module	Model recombination with mass productivity taken into account
MILS test	–	Creation of function/white box/black box test specification for received model, and test implementation
Coding	Coding based on detail design and static check of standard compliance	Auto coding setting of auto coding tool, and static check of standard compliance
Unit (PILS) test	Creation of function/white box/black box test specification for function module, and test implementation	PILS test implementation for function module using reused MILS test pattern
Integration test	Creation of integration test specifications for inter-module interface, and test implementation	Same as before
System test	Creation of test specifications under all modules integrated conditions, and test implementation	Same as before
Acceptance test	Performance check for requirements	Same as before

the execution object that compiles software obtained by auto coding. The execution object is written in the microcontroller, and behavior is verified to secure the reliability. This time, for test patterns, those used for the MILS test are reused. Now, verification is made to ensure that the execution object operates in the performance equivalent to that of the model received.

2. 3 Confirmation for Maintaining Reliability

This section describes how reliability equal to that of the conventional process is maintained for the execution object. Conventionally, execution module reliability was secured by four kinds of tests. In the process recently built also, the integration test, system test, and acceptance test perform behavior verification equivalent to that conducted by the conventional tests. However, the unit test has conventionally performed the behavior verification of the execution object using a simulator or target ECU. On the other hand, in the PILS test in the recently built process, also, behavior is verified under the execution module conditions using an actual microcontroller. As a result, the unit test also can secure the reliability equivalent to that of the conventional test. With the foregoing description, verification equal to conventional method can be made from the unit test to the acceptance test.

In addition, investigation was made on whether or not the Software In the Loop Simulation (SILS) test should be adopted, which was generally performed for securing reliability. This was to carry out behavior verification not with the execution module but with the software as the simulation target, and for test patterns also, test conditions of the MILS test were reused as well as PILS test. The difference was only the point of verification not using the execution object but using software, and finally, because behavior confirmation on the execution object level was required, the adoption of SILS was shelved.

3. Shortening of Work Hours in Each Process

As described in **Section 2. 2**, it is possible to reduce rework because the MILS test is added and behavior is verified on the design stage. On the other hand, the number of processes has been increasing and an increase of the development period has been a concern. In this chapter, measures for shortening the development period of each process for which more work hours have been conventionally taken are discussed, and the verification results of the effect are shown.

3. 1 Analysis of Conventional Work Ratios

In this section, the process to be subject to period shortening is analyzed from the work ratio in the conventional process. **Figure 3** shows the work ratio of each process in one conventional project. Because work of detail design, coding, and unit test accounted for a large percentage, efforts have been primarily made to shorten work hours of these processes.

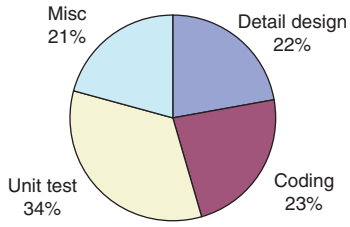

Fig. 3 Work ratio in conventional development process

3. 2 Measures for Shortening Work Hours in Detail Design Process

In this section, discussion will be made on the work hours which can be shortened through reduction of work and improvement in review efficiency by substituting the model for specifications generated in the detail design process, and adding information and traceability required for software to the model. Conventionally, in the detail design process, detail design was performed for achieving unit functions on the basis of the requirement specifications, and specifications were created. On the other hand, this time, because a model is received as requirements, the achievement method is clearly indicated in the model itself. Making use of this, it has been decided to substitute the model itself for the specifications. In this regard, however, because the model only was short of information as compared to the specifications conventionally created, it was decided to add information and traceability, required as software, to the model in this process. With the foregoing, the time for investigating the achievement method and/or time required for design review can be shortened, and the time for creating specifications can be reduced because the function achieving method is shown in the model itself. In addition, functions for automatically outputting the specifications of the information and the format equivalent to those of the conventional JTEKT specifications from the model have been built, and the improvement in review efficiency using the model and paper specifications has also become possible. As a result, work hours of this overall process have been able to be shortened.

3. 3 Measures for Shortening Work Hours of Coding Process

This section discusses how work hours can be shortened by auto coding using the model in the coding process. In the conventional coding process, based on the detail design, coding has been performed. In this study, auto coding using a model in place of this work is performed. The work is only to substitute tool setting in accordance with detail design, and shortening of the

process period is enabled. Tool setting is performed on the basis of the information on detail design. It has become possible to efficiently create codes with high accuracy by combining automated optimum design functions of the tool based on the setting with the JTEKT setting know-how accumulated to date. In addition, by linking the code with the model (specifications) and building the traceability, the comparison between the code and the model has become easy, so that an efficient review has become possible.

3. 4 Measures for Shortening Work Hours of MILS Test and PILS Test Processes

This section discusses how work hours can be shortened by using tools for test pattern analysis and reusing the test patterns in the MILS test process and the PILS test process. First of all, in the MILS test, test specifications are created as is conventionally done. For the function test portion, function test patterns received as requirements are used. By this, the requirements, specifications, and judgment criteria are able to be easily shared with the requesting source. The test patterns and judgment criteria are added to the model, and the validity is confirmed by simulation. In the white box/black box test, efforts have been made to shorten work hours to enable the portion conventionally covered by the review to be systematically and mechanically performed by introducing the coverage analysis with tools. Next, in the PILS test, as test specifications generated in the MILS test are reused as they are, man-hours for creating specifications do not occur. In addition, under the same simulation environment as that of the MILS test, the test case is also able to be reused as it is. Furthermore, the report on the results is automatically created also, so, as a result, the work in the PILS test is reduced.

3. 5 Verification of Shortened Work Hours

In this section, for the work hours of each process achieved to shorten work hours discussed in **Sections 3. 2 through 3. 4**, a part of functions of projects mass-produced in the past is extracted, a model is created, and the results of verifying reduction effects are shown. For the MILS and the PILS tests, man-hours including both added are compared to man-hours of the conventional unit test. **Figure 4** shows verification results, and indicates the ratio to the recent work hours when the work hours required for the conventional development process are designated as 100%. In each process, the work hours are shortened, and particularly the unit test has the work hours shortened by more than one conventional unit test even if two processes are combined in spite of an increase of the number of processes. In each process that accounts for a large work hour ratio, work hours can be shortened, and based on conventional work ratio and shortening ratio of each process, shortening of the development period by about 30% as a whole can be expected.

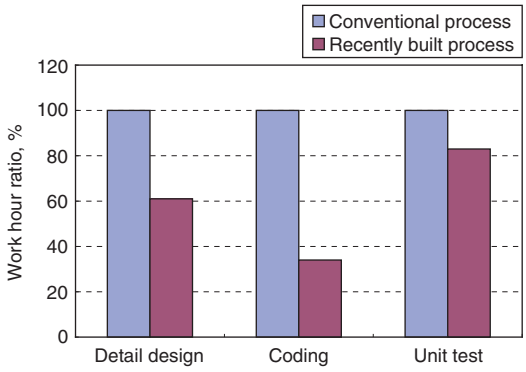

Fig. 4 Results of work-hour reduction in each process

4. Reduction in Rework

In this chapter, measures to decrease rework in the recently built development process are described, and the verification results of decreased rework hour as compared to the conventional development process are shown.

4. 1 Measures for Rework Reduction

In this section, measures for decreasing rework in the recently built development process are described.

Compared to the conventional development process, the following three means are primarily able to be mentioned as means for further decreasing rework. That is, ① as requirements for software development, the model itself is used for requirement specifications, ② in the stage of design process, the MILS test is introduced, and ③ function test patterns used in functional development and the judgment criteria is reused.

First of all, by using the requirement specifications as the model, misunderstanding by ambiguity found in conventional requirement specifications is eliminated. This can eliminate difference in specifications interpretation between requesting side and software development side, and reduced rework is able to be expected.

Next, with respect to the addition of the MILS test, as described in the preceding chapter, implementation of the unit test is enabled in the design stage by the model able to be simulated, and high reliability in the design stage can be secured. Conventionally, the unit test was conducted after the execution object was generated after the coding process, and therefore the rework process when a bug was found was markedly long.

Lastly, discussion is made on reduction in rework achieved by the reused function test pattern used for functional development as requirements and its judgment criteria. Conventionally, in the unit test stage, a function test of one-function module was conducted, and the test specifications were created on the detail design

specifications created by software developers. In this regard, however, when any erroneous reading and/or omission of requirement specifications occurred in the detail design specifications stage, the test specifications were created in accordance with such erroneous or missing information, and accurate function tests were sometimes unable to be implemented. In such event, confirmation whether or not the required functions were achieved and discovery of bugs were left unattended until the acceptance test, and therefore, large rework was generated. Therefore, by reusing the function test patterns created by function developers in the software development process, misinterpretation of requirement specifications and omission of function confirmation have been able to be eliminated with higher accuracy. In addition, comprehensive confirmation of the MILS test has enabled tests and reviews with testing personnel individual differences reduced to a minimum to be conducted by systematic standardization and automatization on the basis of function test items, and the verification accuracy of the mass-production level is able to be secured before coding.

4. 2 Verification of Reduced Rework Hour

This section shows results of comparatively verifying the effects of the measures described in **Section 4. 1** with the conventional process using a rework hour calculation formula. The verification is made with the bug discovery conditions in the past JTEKT mass-production development extracted and the similar bugs assumed to be generated in the recent process. Based on the rework hour calculation formula defined by the Equation (1), comparisons have been made. **Figure 5** shows how the rework hour is summed as the process advances on the basis of the Equaion (1). To the design process, the MILS test is added, and because rework occurs in this process, in the design process, the rework hour increases in terms of the conventional ratio. However, because in this stage, a large number of bugs are found, rework from the unit test overriding processes of conventional specifications review, coding, unit test conducted again, etc. is reduced. Finally, results of reducing rework and reducing the overall rework hour have been obtained as compared to conventional cases.

Rework hour summation in process X

$$= \text{Rework hour summation up to the preceding process} + \sum_{j=1}^{Y} \sum_{i=1}^{Z_j} (T_{ij} \times \eta_{ij}) \quad (1)$$

where,

Y : number of bugs in process X

Z_j : number of rework processes of jth bug in process X

T_{ij} : rework hour in rework process i of jth bug in process X

η_{ij} : (increased) work efficiency ratio from that of conventional cases in rework process i of jth bug in process X

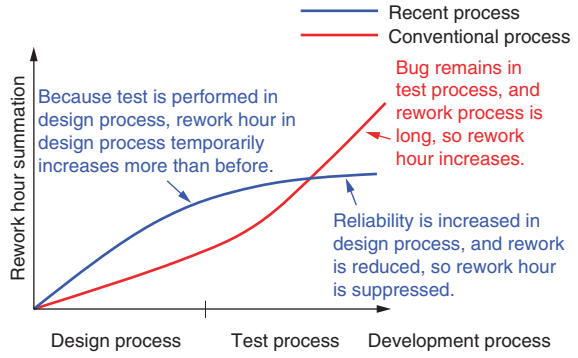

Fig. 5 Rework hour summation as the development process advances

5. Measures for Further Reducing Rework by Model Recombination

In this chapter, discussion will be made on measures for further reducing rework by performing model recombinations in advance in accordance with the performance of generated object anticipated from the model.

In the model-based development, high reliability is secured by auto coding by the use of the model. On the other hand, the performance of the execution object constitutes an important requirement for deciding the cost. **Figure 6** shows the performance ratio to hand coding when part of certain mass-produced software is modeled and auto-coded.

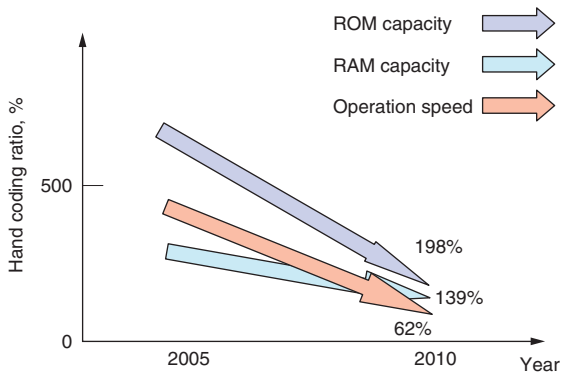

Fig. 6 Performance of execution object

It has been confirmed that the performance of execution object has been greatly improved from the previous performance verification and is in the level of applying to mass production. In this regard, however, the hand coding ratio recently obtained is a result of recombining models several times aiming at the performance matched to the open resources of micro computer. **Figure 7** shows rework generated by code performance.

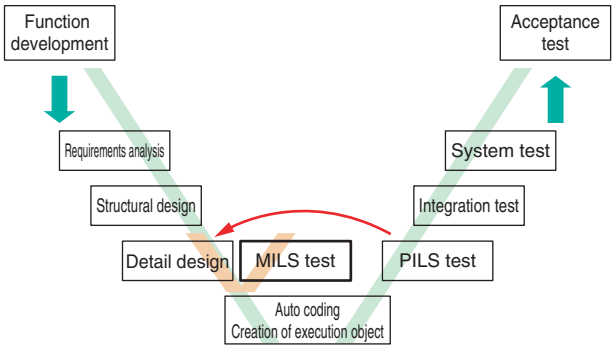

Fig. 7 Rework depending on code performance

While making trial and error of model recombination in this way, performance differences of generated objects by model combination methods have been streamlined and summarized. This has enabled to gradually optimize the model in advance with this trend used as an indicator in the detail design process. Even when models are recombined, the function test is performed by the MILS test, and the reliability is able to be secured, and therefore, rework is able to be reduced without decrease of software reliability.

6. Conclusion

The process with the recently investigated model-based development applied has been confirmed that it is the process that meets the JTEKT development pattern, and application effect has been verified. The process has been also confirmed to maintain the reliability equal to that of software obtained through the conventional development process. In addition, shortened work hours in each process and the effect of improving the development efficiency by securing the high reliability from the design stage have been verified, and it has been confirmed that the development period is able to be shortened. Besides, in the verification of the recently built process, MATLAB available from MathWorks has been used for the model, and TargetLink available from dSPACE for the auto coding and test tool. In addition, in this report, part of joint research results with Toyota Central R&D Labs., Inc., is included, for which the authors will express their thanks.

*1 MATLAB is a trademark of MathWorks.
*2 TargetLink is a trademark of dSPACE.

References

1) CYBERNET SYSTEMS CO., LTD.: MATLAB/Simulink wo Riyoushita Moderubeesukaihatsu no Torendo to Syouraitenbou, Hito to kuruma no Tekunorogii ten Seminaa shiryou (Seminar Material in Automotive Engineering Exposition) (2009) 14.
2) INFORMATION-TECHNOLOGY PROMOTION AGENCY, JAPAN, Software Engineering Center: Kumikomi sofutouea muke Kaihatsu purosesu gaido, SHOEISHA. Co., Ltd. (2007) 65.



Y. KOBAYASHI* H. KATO** K. TOYOZUMI***

* *Electronics Engineering Dept., Steering System Operations Headquarters*
** *System Development Dept., Steering System Operations Headquarters*
*** *Electronics Engineering Dept., Steering System Operations Headquarters, Doctor of Engineering*