R&D Project Report

Quality Evaluation of Embedded Software in Robot Software Design Contest

Hironori WASHIZAKI[1], Yasuhide KOBAYASHI[2], Hiroyuki WATANABE[3], Eiji NAKAJIMA[4], Yuji HAGIWARA[5], Kenji HIRANABE[6] and Kazuya FUKUDA[7]

[1]*National Institute of Informatics*
[2]*Afrel, Inc.*
[3]*OGIS-RI Co.*
[4]*Mamezou Co.*
[5]*CATS CO.*
[6]*Change Vision, Inc.*
[7]*Toyoko System Corp.*

ABSTRACT

A robot design contest, called the "Embedded Technology (ET) Software Design Robot Contest," which involves designing software to automatically control a line-trace robot, was held in Tokyo, in 2005. The contest was intended to provide a practical opportunity to educate young Japanese developers in the field of embedded software development. In this paper, we give the results of the contest from the viewpoint of software quality evaluation. We created a framework for evaluating software quality, which integrated the design model quality and the final system performance, and we conducted an analysis using this framework. As a result of the analysis, the quantitative measurement of the structural complexity of the design model was found to have a strong relationship to the qualitative evaluation of the design by the contest judges. On the other hand, no strong correlation between the design model quality evaluated by the judges and the final system performance was found. For embedded software development, it is particularly important to estimate and verify reliability and performance in the early stages, according to the design and analysis models. Based on the results, we consider possible remedies with respect to the models submitted, the evaluation methods used, and the contest specifications. To adequately measure several quality characteristics, including performance, in terms of a model, it is necessary to improve the approach to developing robot software (for example, by applying model-driven development) and to reexamine the evaluation methods.

KEYWORDS

Software quality, product metrics, robot contest, embedded software, software design, software model

1 Introduction

A robot design contest, called the "Embedded Technology (ET) Software Design Robot Contest" [2] (hereafter, simply "the contest"), was held by the Japan Embedded Systems Technology Association [3] in Tokyo, in 2005. The contest was intended to provide two opportunities for young Japanese developers: a learning opportunity in analysis and design modeling technology in the field of embedded software development, and an opportunity to experience the entire development process in a short period of time.

Received September 21, 2006; Revised March 20, 2007; Accepted March 27, 2007.
[1)]*washizaki@nii.ac.jp*, [2)]*kobayashi-y0222@afrel.co.jp*,
[3)]*Watanabe@ogis-ri.co.jp*, [4)]*eiji@mamezou.com*,
[5)]*hagiwara@zipc.com*, [6)]*kenji.hiranabe@change-vision.com*,
[7)]*fukuda@toyoko-sys.co.jp*
DOI: 10.2201/NiiPi.2007.4.6

In a narrow sense, analysis and design modeling is a technology for organizing and expressing the results of interpreting target domains, requirements, system-integration techniques, and implementation techniques as models by taking an abstraction from a particular point of view. In a broader sense, it includes the use of models to examine particular properties, make revisions, or change code. The need for education on analysis and design modeling in this broader sense has increased, because the requirements for embedded software have become more complex and larger scale and have begun to include higher reliability. The contest was planned as an opportunity to respond to this demand.

The entrants participated in teams over an approximately three-month period from April to June. They first modeled the design of embedded control software for a line-trace robot and then implemented the software according to the model. The robot, built with LEGO Mindstorms blocks, [4] was required to move independently while following a dark line. On the contest competition day in July, the participants tested the overall adequacy of each robot as an embedded system by demonstrating its independent operation. As judges of the competition, we evaluated the embedded software designs submitted by the participants before the robot tests. Then, the overall implementations were evaluated independently of the model according to the robot performance results, in terms of the time required to complete the test course.

In this paper, we report the results of experiments on quality evaluation of embedded software[1)].

- First, we assumed that the better the quality of the robot design model is, the better the robot's runtime performance is. This assumption was based on the well-known principle that internal quality attributes influence external quality attributes, and the external quality attributes influence the attributes of quality in use attribute (from Figure 2 in [5]) [2)]. In other words, *evaluating and improving product quality is one means of improving quality in use*, [5] and, *appropriate internal attributes of software are a prerequisite for achieving the required external behavior*. [5] Therefore, we expect that a value of external quality for an executable software system can be estimated from a value of internal quality based on non-executable artifacts, including models. [6]
- Second, we created a framework for evaluating both the model quality and the final system performance in a systematic way, and using this framework, we carefully analyzed the relationships between these two sets of evaluation results. From this analysis, we found that the quantitative measurement of the structural complexity of the design models had a strong relationship to our qualitative evaluation of the design. We also found that there was no strong correlation between the design model quality, as judged by us, and the final system performance.
- Finally, we considered the nature of the models that correspond to a system's performance and the appropriateness of the model evaluation methods used in the contest. We concluded that it is necessary to reexamine the evaluation methods in order to adequately measure several quality characteristics, including performance, in terms of the model. Moreover, we found the participants have to avoid the lack of correspondence between the model and the program code by applying model-driven development or other techniques.

The remainder of this paper is organized as follows. The next section gives an overview of the contest. Section 3 describes the model and performance evaluation methods and gives the results of the contest. Section 4 presents a quality evaluation framework integrating both the model and performance evaluations. In section 5, we report the results of analyzing the relations obtained using this framework. In the last section, we draw our conclusions and discuss future works.

2 Contest overview

2.1 Contest statistics

The contest has been held five times, once a year from 2002 to 2006. Table 1 lists the time period, number of teams, and participants in the contest each year.

The intention of this contest is that participants learn modeling technology, including object-oriented embedded software analysis and design modeling using UML, [7] through their participation. Modeling-technology training sessions and workshops to discuss the models and their performance were held, and the contest re-

Table 1 Scale of the contest.

Time	Period	Num. teams / participants
1st	February–March 2002	24 / 110
2nd	February–April 2003	21 / 110
3rd	February–April 2004	41 / 210
4th	April–July 2005	53 / 250
5th	April–July 2006	108 / 500

[1)] A preliminary conference paper based on this work can be found in [1].

[2)] This general principle is not specific to reliability and efficiency among all quality characteristics; however, we assumed it is applicable to those two characteristics.

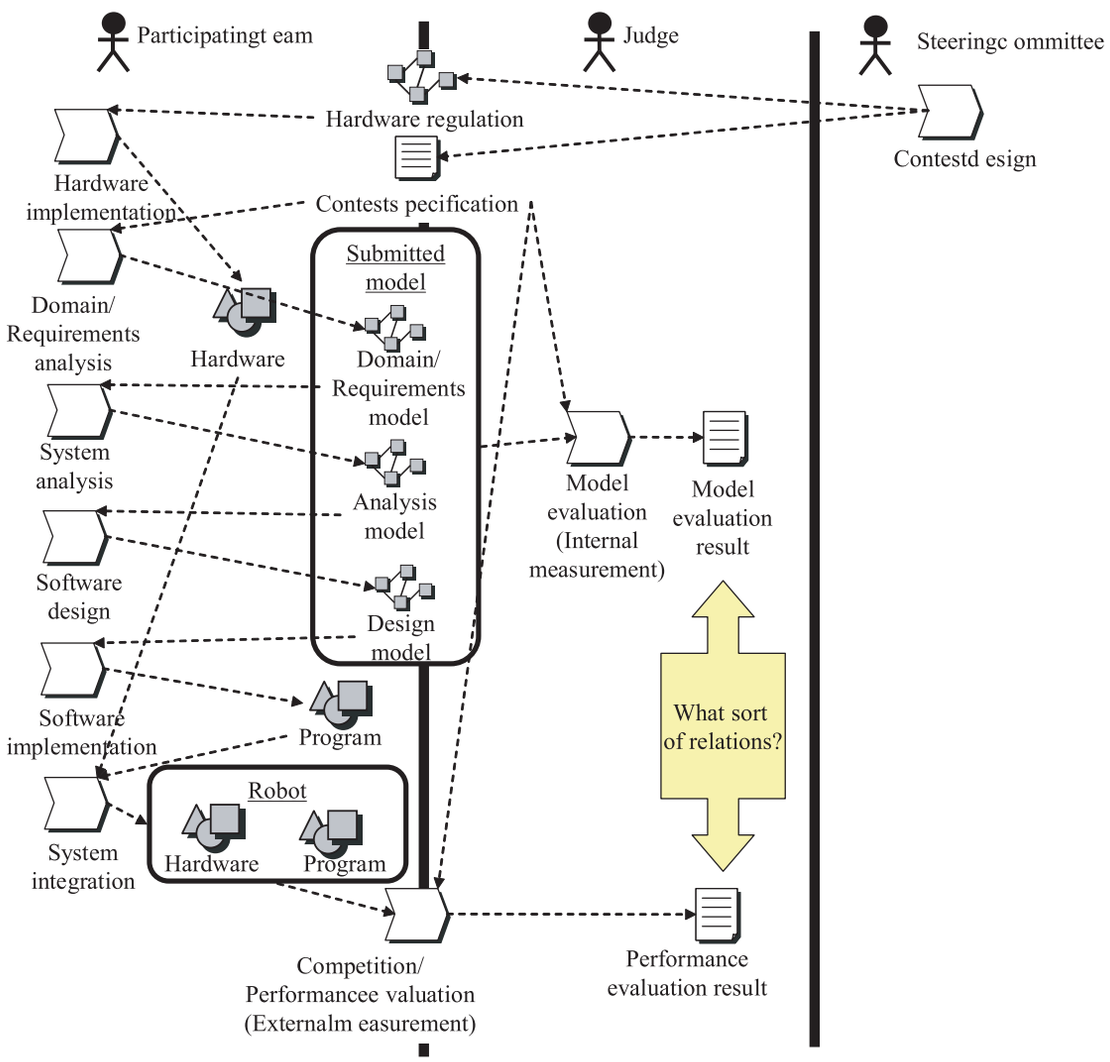

Fig. 1 Workflow of the contest process.

sults were published. It has been clear from questionnaire results after previous contests (held in 2002-2004 as "UML robot contest" [8]) that they were highly educational for the participants learning these development technologies. [9] Because of this educational effect, the number of participants has been increasing (as shown in Table 1).

The 2005 contest, which we examine in this paper, had approximately 250 participants in a total of 53 teams. Of those, we looked at the models and test results from the 47 teams that submitted entries within the contest period.

2.2 Contest process

Fig. 1 illustrates the contest process in the form of a UML activity diagram based on the Software Process Engineering Metamodel (SPEM) [10] profile, with several additional particular comments. Here, the participating teams, the judges, and the steering committee are drawn as process participants, along with the activities and inputs that they contribute. For example, each participating team went through a series of activities, including hardware implementation, domain/requirements analysis, system analysis, software design, software implementation, and system integration. The hardware regulations and contest specifications were actually received from the steering committee of the contest.

2.3 Contest regulations

The hardware and software regulations and the evaluation methods are described below.

- Hardware: The hardware consists of a robotic vehicle assembled from LEGO Mindstorms blocks. Fig. 2 shows an overview of the robot hardware. The robot is capable of moving forward and backward and turning left or right under motor control, and of recognizing a course indicated by a black solid or dashed line by using a light sensor to detect brightness on the running surface.

  To provide an opportunity for participants to compete purely on the basis of software quality, they were not asked to design the hardware.

- Software: The participants analyzed, designed, and implemented software to control the hardware so that it would traverse a course indicated by a black solid or dashed line approximately 20 meters long to reach the finish. The software was required to control the hardware automatically, adjusting for course conditions. No communication with the robot was permitted once it had begun the course.

  The format of the requirements, analysis, and design model was not regulated, but all models were submitted with diagrams in various versions of UML [7] (such as UML version 1.1, 1.5, or 2.0) and accompanying descriptions or pictures. The software was implemented in languages capable of controlling Mindstorms robots, including C, C++, and Java.

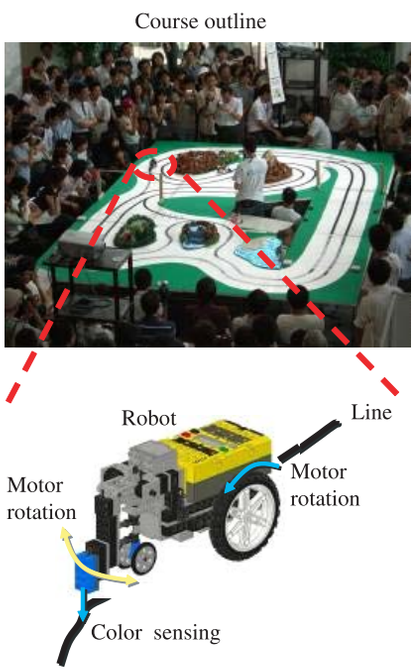

Fig. 2 Overview of the course and a robot.

- Evaluation: The judges evaluated both the submitted models and the results achieved by the robots in a running performance test.

3 Software quality evaluation in the contest

During the contest, the models and the robot performance were evaluated independently. The evaluation methods and the results are described below.

3.1 Model evaluation (judging standards)

The judges qualitatively evaluated the submitted requirements, analysis, and design models for both content and presentation, and they assigned a final grade for each ranging from A (best) to D (worst). Then, from among the models receiving an A or B grade, three particularly excellent models received awards. The details of how content and presentation were evaluated are given below. All of the submitted models were based on object-oriented designs, and most were written in UML, so an evaluation from the object-oriented/UML perspective was also included.

- Content: The validity and correctness of the model content were evaluated. Specifically, the following three aspects of the model were evaluated:
  - Validity: We judged the validity of each model element and the relations/structures among elements from an object-oriented perspective. For example, we considered the adequacy regarding the problems of divide and conquer, role division, role assignment, the level of abstraction, the relations, and multiplicity.
  - Logical correctness: We judged the feasibility of the requirements described in the models. For example, we considered the correctness of the algorithm and behavior models.
  - Algorithm originality: We judged the uniqueness of the described algorithm, such as the dynamic sensor calibration functionality and the return functionality (for when the robot breaks away from the line).

- Presentation: Apart from the correctness of the model, we judged whether the design intention was presented clearly. Specifically, the following three aspects were evaluated:

– Notational validity: We checked whether the described models correctly and appropriately follow the UML (or any other diagram) specifications for model representation. For example, if the target model seemed to use a certain version of UML, we checked its notational validity according to the same version of UML.
– Clarity: We judged the clarity (understandability) of the model by checking its adequacy regarding the separation of views, layout, and so forth.
– Originality and ingenuity: We judged the uniqueness of the described model including the accompanying descriptions/pictures.

The judges made qualitative evaluations with the final decisions obtained by the following two-step process. First, each of the six judges evaluated the models individually. Second, we collected these intermediate evaluations, and conducted a group discussion to reach a final evaluation for each model. This process was conducted to eliminate the biases of individual judges and obtain results with high validity.

3.2 Model evaluation results

Fig. 3 shows the number of models receiving each evaluation grade, as described above. Most of the models received B or C grades, while fewer models received A or D grades. As examples, the structure part and the behavior part of the model considered the best in the A group by the judges are shown in UML form in Figs. 4 and 5, respectively.

Several trends observed in the evaluation are described below.

- Content:
  – Validity: Most of the models consisted of a number of classes defined through a divide-and-conquer approach; however, some classes with too much functionality were seen. Furthermore, models that could determine the type of a given segment of the course (e.g., straight line or curved) were evaluated higher than those that simply followed the line. For example, Fig. 4 shows the abstract classes (`Course type` and `Drive method`) for the course type and driving method, allowing the driving approach used to be set according to the current course conditions.
  – Logical correctness: In many of the models, the hardware control method had not been modeled, so it was not possible to verify the validity and correctness of the algorithm.
  – Originality: Several different driving strategies were described in the models, from those that used the same control strategy to follow both solid and dashed lines to those that changed their control strategy depending on the type of line. There were also models which described their development processes in addition to the products.
- Presentation: Most of the models were correct in terms of diagram notations because these were described with modeling tools (such as JUDE [11] and EclipseUML [12]) that do not allow users to draw incorrect diagrams; however, half of the

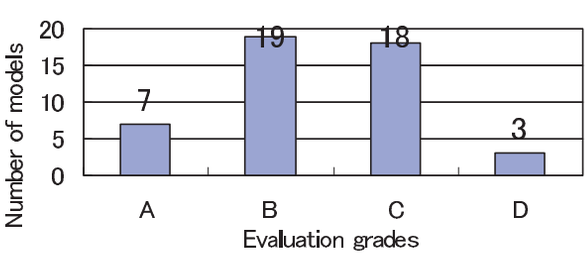

Fig. 3 Model evaluation results.

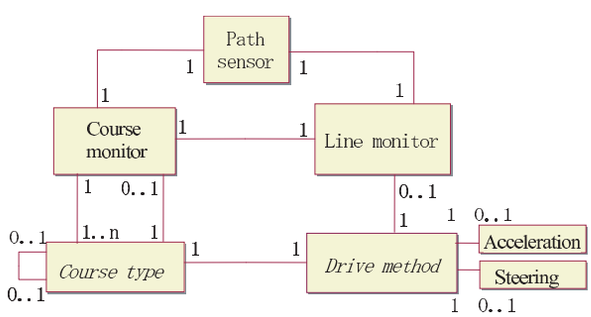

Fig. 4 UML class diagram representing the architecture of the winning model.

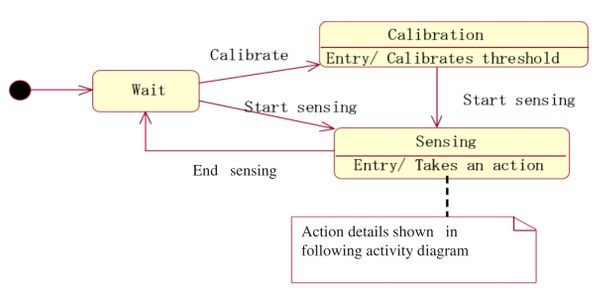

Fig. 5 UML state chart diagram representing the path sensor's behavior in the winning model.

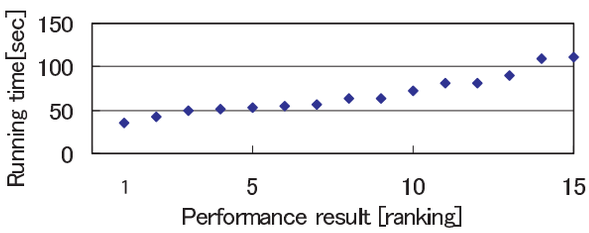

Fig. 6 Times and ranks of the teams whose robots completed the course.

models used UML notation in inappropriate ways. For example, there were UML diagrams in which attributes or multiplicity was not given, relations or role names were inappropriate, or the target of a state diagram was not specified. In terms of clarity, it was easier to read and understand the intention of more of the models that used pictures, text, or non-UML diagrams to explain their control plan than of those that did not. In addition, some of the models overused metaphor, [13], [14] with the result that the intention of their analysis and design was obscured.

3.3 Performance evaluation

In the performance test, each team ran the robot on the regulated course twice. The judges evaluated the robot running performance of all the participating teams in terms of whether the robot could complete the course within a specified time. The teams whose robots completed the course were also evaluated in terms of the fastest completion time, and their ranks were based on this time.

3.4 Running performance results

Of the 47 submissions, the robots from 15 (31.9%) of the teams completed the course. The times and rankings of these teams are shown in Fig. 6. The fastest time was 35.4 sec, while the slowest was 112.3 sec.

4 Integration of model and performance evaluations

To examine both the model and performance results uniformly, a framework to integrate the two results was needed.

From the viewpoint of software-quality measurement, [5] the performance evaluation described in Fig. 1 was a kind of external measurement [15] of the software embedded in the robot. An external measurement is an activity that measures software quality by assigning a quantitative value to some property of the behavior of the system in which the final executable software is installed. In the contest, the hardware was uniform for all teams, so any differences in performance could be attributed to differences in the software. On the other hand, the model evaluation was a kind of internal measurement [16] of the software quality. An internal measurement is an activity that measures the software quality by assigning a quantitative value to some property of the intermediate or final software.

Using the various quality characteristics required by the ISO 9126-1 [5] software quality model, we applied internal and external measurements to build a systematic framework for software quality evaluation. This framework is described in Table 2. The software quality characteristics used in the contest are summarized here:

- Functionality: Although not clearly specified as a judging standard, the software's success in satisfying the functional requirements was included in the judges' evaluation of the model validity. Thus, the internal measurement of the functionality was considered to consist of the model evaluation.
- Maintainability: Software maintainability is related to clarity and notational validity. The internal measurement of maintainability was considered to consist of the model evaluation regarding presentation.
- Reliability: Reliability was externally measured by whether the robot could complete the test course under the control of the software. It could also be measured internally through an evaluation of the model's logical correctness, but as noted above, we could not verify the logical correctness of the submitted models.
- Efficiency: The course completion time and rank of the robot under the control of the software gave an external measurement of the quantity of time resources that it used.
- Usability: Usability of the software by the actual users (the contest participants) was not considered or evaluated in the contest.
- Portability: Portability was not considered or evaluated because the contest participants developed new software for a single hardware environment, unrelated to the results of previous or future contests.

The applicable scope of this framework is not limited to the contest's products or embedded systems; it can be partially applied to other software systems. Since the internal measurements in the framework depend on

Table 2 Constructed software quality evaluation framework.

Characteristics	Factor to be shown	Internal measurement (Model evaluation)	External measurement (Performance evaluation)
Functionality	Required functions are supported	Validity of the content	—
Reliability	Functions behave normally under all conditions	Logical correctness (∗)	Whether the course was completed
Usability	Intuitiveness/usability	—	—
Efficiency	Use of time/resources	—	Course time/rank
Maintainability	Effort required for maintenance	Presentation	—
Portability	Easiness of porting the target to different environment	—	—
		—	—

(∗) In the contest, the logical correctness of the submitted models could not be verified.

only models obtained as a result of software analysis/design, they can be applied to any system if some software design model corresponding to the system is available.

On the other hand, since the external measurements in the framework strongly depend on the contest specifications (including the robot), they would have to be modified or replaced with other quality metrics that can measure and evaluate corresponding external quality attributes of the new target system (even if the target system was an embedded one).

5 Analysis of relation between model evaluation and performance evaluation

In the framework described in the previous section, we drew connections between the model and performance evaluation results and various quality characteristics. Note that the reliability and efficiency measurements were based on the performance evaluation rather than the model evaluation.

It is well known that many problems in software development, not limited to embedded software, are introduced during early stages such as analysis and design and that these problems exert a dominating effect on software and system quality characteristics. Therefore, it is expected that a value for the external quality of executable software systems can be estimated from a value for the internal quality based on non-executable artifacts including models. [6] Especially for embedded software development where real-time performance is required, it is particularly important to estimate and verify the reliability and efficiency in the early stages by using the model.

Below, we first confirm the validity of the qualitative model evaluation results by comparing them against other measurements. Then, after discussing how a correlation between the model evaluation results and the performance evaluation results should be expected, we attempt to verify the reliability and efficiency of the models in the contest at an early stage based on the correlation between these two sets of results.

5.1 Validity of qualitative model evaluation

By looking at the relationships between the evaluation results and the values obtained by applying several quality measurement methods (i.e., product metrics) to the models, the validity of the qualitative evaluation of the models can be shown.

Of all possible metrics applicable to an object-oriented model, we used the number of classes [17] and the coupling complexity (Coupling Factor: COF [18]) among the classes as metrics for measuring structural aspects because all of the submitted models included a class diagram showing the overall software architecture. If the model included several class diagrams, we looked at the diagram showing the overall driving control method. On the other hand, we could not uniformly apply any metrics for measuring behavioral aspects, because each of the submitted models used different diagrams (e.g., a state chart diagram, a sequence diagram, etc.) to represent the behavior of different targets (e.g., whole or part).

The details of the metrics that we used are as follows.

- The number of classes indicates the scale of object-oriented software. It was measured in terms of the number of classes in the class diagram showing the overall architecture.
- COF indicates the static complexity of object-

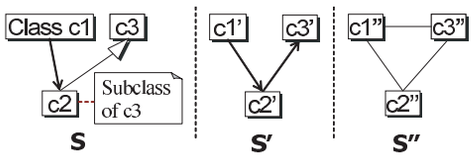

Fig. 7 COF measurement example.

oriented software according to coupling between classes. It was measured by taking the number of one-way relations between classes (two-way relations were counted twice), not including inheritance or dependency relationships (dotted arrows), in the class diagram. This value was then normalized for scale to a value from zero to one. The complexity, $COF(S)$, of a set of related classes, S, is given by the following function where $\#S$ denotes the number of classes in S.

$$COF(S) ::= \frac{\text{(Number of one-way relations excluding inheritance)}}{\#S^2 - \#S - 2(\text{Number of subclasses})}$$

For example, in Fig. 7, the COF values for each of the programs S, S' and S'' built from three classes would be 1/4, 1/3 and 6/6 (=1), respectively. Therefore, S'', with a complete, bi-directional graph formation is measured as the most complex.

The number of classes and the COF measurement value for each of the 47 submitted models are shown in Fig. 8. In the figure, all of the models receiving an A grade had more than 10 classes and a COF value of no more than 0.15. Of the B-grade models, most of them had more than 15 classes and a COF value of no more than 0.16. From this, we can see that for models of a certain size, those with low complexity were evaluated more highly. Conversely, most of the models with fewer classes and a higher COF value were given a C grade. In other words, small-scale models with higher complexity were evaluated lower.

For example, Fig. 9 shows a UML class diagram representing the static software structure of one of the A-grade models[3]. This is not a small-scale model and its complexity is low, because the number of classes and the COF value are 38 and 0.04, respectively. In contrast, Fig. 10 shows a UML class diagram for one of the C-grade models. This is a small-scale model and its complexity is relatively high, because the number of classes and the COF value are 7 and 0.33, respectively.

[3] Figs. 9 and 10 are meant to roughly illustrate structural complexity; we do not discuss the content/meaning of these figures.

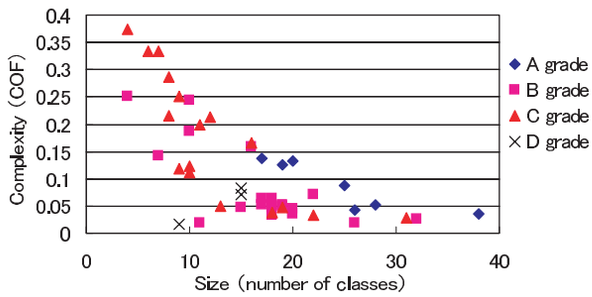

Fig. 8 Number of classes and COF value for all models.

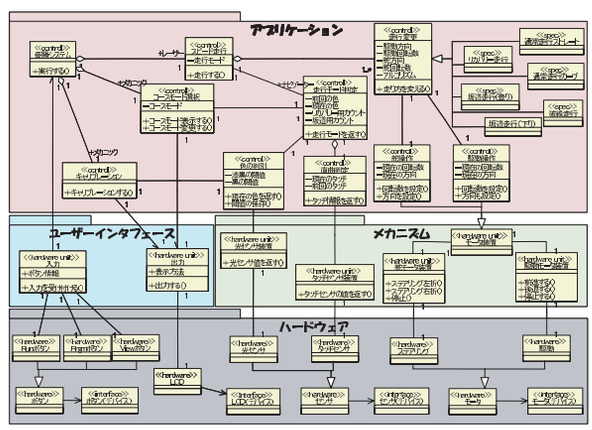

Fig. 9 UML class diagram representing the static structure of an A-grade model.

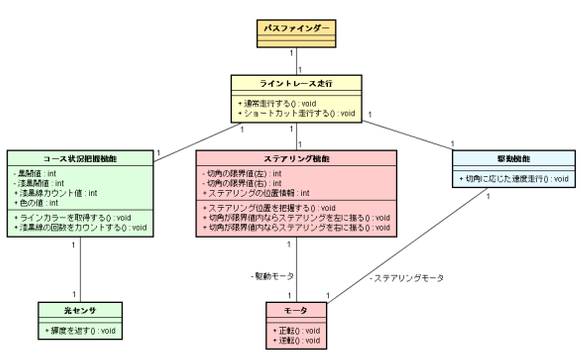

Fig. 10 UML class diagram for a C-grade model.

Since the roles of each class could not be determined from the class diagram because of a lack of adequate explanation, models receiving a D grade were excluded from examining the scale and complexity of the static structure. In addition, some models received a B grade but had few classes and higher complexity. These models had other diagrams (e.g. a state chart diagram) that comprehensibly represented the operation of the algorithm from a dynamic perspective, so they were consid-

ered separately from most of the other B-grade models, which were graded according to the adequacy of the class diagram.

In evaluating the models, we considered both the validity of the model's details from an object-oriented perspective and the adequacy of the presentation. The smallest unit of structure in an object-oriented design is a class, and the number of classes and the COF value can be regarded as quantitative measurements of the appropriateness of the class abstractions and the relationships between those classes. Furthermore, having an appropriate number of classes and COF value can be seen as directly related to the adequacy of the presentation. For example, if the number of classes is extremely large or small, or the COF value is very large, it is very likely that the content and intention will be difficult to understand.

From these observations, it can be recognized that the scale and structural complexity as expressed by the number of classes and the COF value were implicitly considered in the qualitative evaluations by the judges. It is well known that software maintenance costs are significantly affected by the levels of software complexity. [19] Especially in module-based software, module couplings are important metrics for maintainability. [20] In the contest, the judging activity can be thought as a kind of a maintenance activity, because the judges acted as third parties evaluating the models after their implementations. Therefore, the given qualitative measurements that implicitly reflect complexity and scale can be seen as mostly valid.

We also gave some consideration to originality when making qualitative evaluations. Since originality is not related to the essential quality requirements of the software, it could be argued that it must be excluded from the discussion below regarding the correlation between quality and the performance results. Among the models actually submitted, however, most of those that showed significant originality were also evaluated highly for their content, so we can see that the consideration of originality did not have a significant effect on the values of the final results.

5.2 Expectations for relation between model and performance evaluation results

In the contest, it was our intention to encourage the participants to perform Round-Trip Engineering (RTE [21]), or Model-Driven Development (MDD [22]) by requiring them to submit their model for evaluation before the performance test was held. We did not actually investigate, however, whether the program installed in the robot was an adequate implementation reflecting the model submitted (though this was our hope).

- RTE is a development technique in which several aspects of a target domain or problems are abstracted and expressed in a model, so that the relationship between the model and the program code is maintained as development progresses. By applying RTE, software quality is expected to be high because the close relationship between the model and its implementation means that any problems are detected early in the abstract model.

- MDD is a development technique that extends RTE, deriving the program code by repeatedly transforming the model (by hand or with a computer) according to some specific transformation rules. Through the application of MDD, productivity increases in problem domains where such transformation rules have been well verified and accumulated. Using the fact that characteristics of interest are inherited through transformation steps by lower-level models from high-level abstract models, functional and non-functional characteristics can be verified with high precision at an early stage of development.

As shown in Table 2, the results for the external measurements of the software reliability and efficiency came from the results of the performance tests. Thus, if the participants used RTE or MDD and created a model and program with an adequate relationship, then the robots of the participants whose models were evaluated better for both reliability and efficiency would be expected to actually complete the test course more quickly.

5.3 Results of relationship analysis

Based on the above expectations, we examined the relationship between the model evaluation results and the performance results.

The evaluation method discussed in section 3.1 did not explicitly include evaluation from an efficiency perspective. Neither could we determine whether the submitted models were likely to show high reliability by evaluating their logical correctness. It might be still possible, however, that the evaluations, which considered the validity of the model content and the adequacy of presentation, reflected reliability and efficiency. More specifically, it might be possible that object-orientated evaluation of the static architecture and dynamic behavior implicitly evaluates models as inadequate if their designs are overly complex in terms of scale, structure, inter-object communication, or state transitions. This type of excess complexity can ultimately lead to decreases in the reliability and efficiency of software.

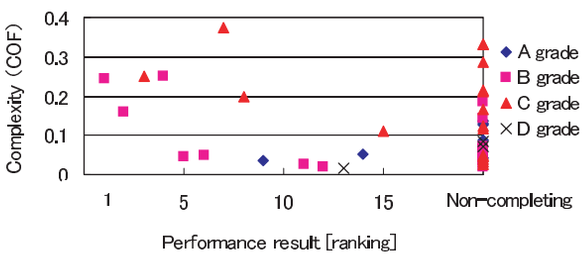

Fig. 11 COF and performance results.

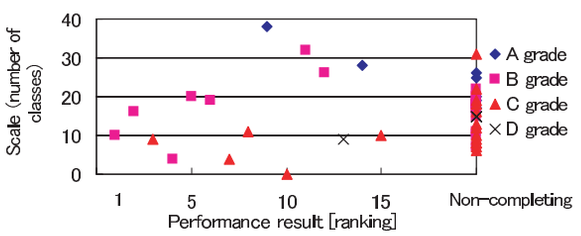

Fig. 12 Number of classes and performance results.

Thus, we verified the relationship between the scale and static complexity in the model evaluation results and the performance results. The COF values and performance ranks are shown in Fig. 11, along with the evaluation grades, for the 47 submitted models. Similarly, the number of classes and the performance rank are shown in Fig. 12. The data for teams whose robots did not complete the test course are shown under "Non-completing."

In Fig. 11, there is no recognizable relationship between the performance ranks of the completing teams and the COF values. No trends in the COF values distinguishing the completing and non-completing teams can be seen. Furthermore, more than half of the models that received an A grade produced robots that did not complete the course, and for those that did, the rankings were low compared to others that received B or C grades. There was also no recognizable relationship between the performance rank or completion and non-completion for the models receiving B, C, or D grades. Similarly, Fig. 12 shows no recognizable relationship between the model scale and the performance rank or evaluation grade.

A correlation matrix for the number of classes (denoted as "Num. classes"), the COF value, the model evaluation (denoted as "Eval. grade"), and the performance rank (denoted as "Run result") is shown in Table 3 for the 15 completing teams. To calculate the correlation coefficients between grades A through D and the other values, the grades were assigned the numbers 1 (grade A) through 4 (grade D), respectively. Regarding the performance rank, there was a weak but positive correlation between the performance rank and the number of classes. Moreover, there was a weak negative correlation between the performance rank and the COF value. These correlations might suggest that a robot whose model was small, with high complexity, would have run faster among the completing teams, but this observation contradicts our expectation. There was a very weak correlation between the COF value and the evaluation grade, however; we attribute this to the fact that many of the models with a low COF value (good in terms of complexity) produced robots that did not complete the course.

Table 3 Correlation matrix for model evaluation results and performance results for completing teams.

	Num. classes	COF	Eval. grade	Run result
Num. classes	1			
COF	−0.732	1		
Eval. grade	−0.711	0.288	1	
Run result	0.353	−0.564	0.093	1

Bringing these contest results together, we can observe the following.

- The qualitative and quantitative model evaluations did not reveal the robot reliability.
- If a robot had high reliability in terms of completing the course, the quantitative model evaluation could somewhat reveal the robot efficiency (i.e., the runtime performance), in contrast to our expectation; the robots whose models were small, with high complexity, ran faster among the completing teams.
- Teams that focused on programming and system testing rather than modeling may have dealt with the hardware characteristics more appropriately, since the development period of the contest was limited. We could not verify this assumption, however, because almost all of the participating teams did not open their programs to the public.

Moreover, we should give some consideration to the situational effect on the performance test results. As mentioned in section 3.3, each team ran its robot only twice on the competition day. It is possible that robots based on models evaluated highly failed the performance test by accident because of environmental factors, such as changes in the course conditions and the room's light intensity during the entire competition. Thus, to confirm the validity of the results obtained from the contest, we examined additional data obtained

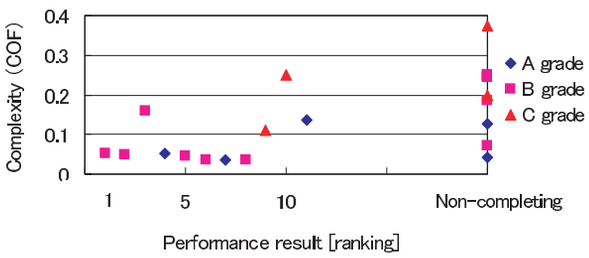

Fig. 13 COF and performance results in the post-contest championship.

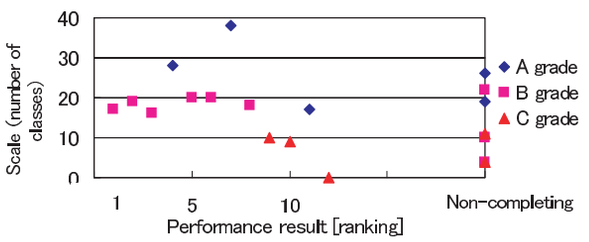

Fig. 14 Number of classes and performance results in the post-contest championship.

Table 4 Correlation matrix for model evaluation results and performance results for completing teams in the championship.

	Num. classes	COF	Eval. grade	Run result
Num. classes	1			
COF	−0.614	1		
Eval. grade	−0.833	0.458	1	
Run result	−0.443	0.449	0.307	1

from the post-contest championship, in addition to the contest results.

In November 2005, a championship was held with 20 selected teams competing in terms of performance on the same course as that of the contest (twice per team). The steering committee selected teams whose models were evaluated highly or whose robots successfully completed the course at the contest. Note that each of the teams was allowed to refine the program but not to modify its model from the contest.

The COF values and the performance ranks in the championship are shown in Fig. 13, along with the evaluation grades for the 20 models. Similarly, the number of classes and the performance rank in the championship are shown in Fig. 14. In addition, the correlation matrix for the 12 completing teams (out of 20) is shown in Table 4. From these figures and the table, there was no strong correlation between the performance rank and any of the other measurements. For the completing teams, however, there was a weak negative correlation between the performance and the number of classes. On the other hand, there were weak positive correlations between the performance and the COF value, and between the performance and the evaluation grade. These correlations might suggest that the robots whose models were large, with low complexity, ran faster among the completing teams; this observation meets our expectation.

Bringing these championship results together we see the following.

- The qualitative and quantitative model evaluations did not reveal the robot reliability.
- If the robot had high reliability in terms of completing the course, the quantitative model evaluation could somewhat reveal the robot efficiency without contradicting our expectation: the robots whose model had a certain size, with low complexity, ran faster.
- Teams that focused on modeling in the contest could have spent more time on programming and testing in the championship, using the same good model. Again, we could not verify this assumption.

From the above results in the contest and the championship, we can see that there was no strong relationship between the qualitative model evaluation or structural characteristics and the system performance results, although there were some weak relationships indicating the possibility of evaluating or estimating system efficiency in terms of the model. Thus, we conclude that the evaluation methods and measurements of structural characteristics (number of classes and COF) of the models do not reflect the reliability (and efficiency) of the software, and therefore, they do not provide adequate internal measurements of quality characteristics.

5.4 Reasons and remedies for lack of strong relationship

We now consider the reasons why we could not clearly measure the software reliability and efficiency from the models by using the internal measurements described above. We also examine possible remedies with respect to the submitted models, the evaluation methods, and the contest specifications. The potential remedies can be divided into three types: (1) by participants, (2) by judges, and (3) by the steering committee. (1) The reasons related to the submitted models and the potential remedies by the participants are as follows.

- The dynamic models were inadequate.

  Reason: In many of the submitted models, the hardware control and the driving strategy were not modeled, and logical correctness, which would likely be reflected in high reliability, could not be judged.

  Remedy: Participants should enrich their behavior models to express dynamic software behavior and algorithms.

- There was a lack of correspondence between the model and program.

  Reason: In typical embedded software development, efficiency is often refined by iterating through the implementation and testing. It is possible that some contest participants adopted this kind of approach. Without maintaining the relationship between the software implementation (the program) and the model, however, the results of this type of iterative work will not be reflected in the submitted model.

  Remedy: Each team should adopt RTE or MDD as their basic development approach in order to maintain the relationship between the model and the program. MDD encourages deriving models whose operation can be verified at an early stage, so simulations using the model and dynamic quality measurements can be easily applied. This should lead to the generation of more reliable programs.

  Note that the difference between the transformation tools and rules used for the same model in MDD-based development might lead to some difference in the program or system quality. Therefore, it is necessary to consider such differences in analyzing the relationship in terms of quality between the model and the program transformed by MDD.

- Non-functional characteristics were not specified.

  Reason: None of the submitted models clearly expressed any of the non-functional characteristics. Conventional standard model notation (e.g., UML) is effective for improving the expression or readability of functional characteristics, but it is not effective for expressing non-functional characteristics.

  Remedy: A UML performance profile [23] or another performance framework [24], [25] can be applied to add special annotations regarding non-functional characteristics into the standard models. For example, Woodside [26] proposes a technique for adding performance annotations to UML models and transforming them into performance models for which performance can be verified.

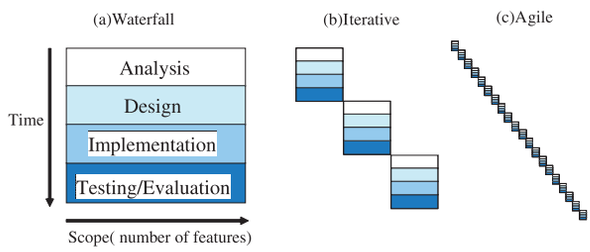

Fig. 15 Iterative processes in terms of time and scope (from [27]).

- The models were not built by considering the hardware and environmental characteristics.

  Reason: The hardware characteristics (such as speed and capacity), the environmental characteristics, the control algorithm, and combinations of them had significant effects on efficiency. Most participants, however, did not know how to deal with these factors or were unable to express how to deal with them in their models.

  Remedy: In embedded software development, the hardware and environmental characteristics tend to become obvious by testing software on the target hardware. Therefore, each team should use some iterative development process to conduct a cycle of modeling, implementation, and testing on the target hardware. This should enhance the model's capability of dealing with the hardware and environmental characteristics. Fig. 15 shows a comparison of three typical process models (waterfall, iterative, and agile) from the viewpoint of time and scope [27]. The waterfall process tries to cover the entire scope in each activity (such as analysis and design) and contains no iteration; it leads to a situation in which the hardware and environmental characteristics are not dealt with in the analysis and design models. On the other hand, the iterative and agile processes divide the entire scope into several smaller fragments and perform several iterations; this leads to a system built step by step, involving the hardware and environmental characteristics.

  Moreover, each team could record some software patterns [28], [29] (or anti-patterns [30]) that take the hardware characteristics into consideration and encapsulate knowledge gained from software analysis and design that successfully increased efficiency (or that failed to increase ef-

ficiency). By explicitly stating which patterns are to be applied to the model, judging the efficiency of the model would become easier, and this should result in the generation of final software that is more efficient. As an example, Smith et al. has proposed performance anti-patterns and workarounds for the problems of performance in traditional object-oriented design. [31]

(2) The reasons related to the model evaluation method and the potential remedies by the judges are as follows.

- The evaluation of efficiency was not emphasized or quantified.

  Reason: Efficiency was not explicitly included in the model evaluation.

  Remedy: We can emphasize efficiency evaluation for models that also show dynamic aspects, and we can, as much as possible, use quantitative measurements. For example, we can evaluate the efficiency annotations in the model as described above, or we can apply dynamic metrics that measure the complexity of the dynamic behavior of an object-oriented design. [32]–[34] To do this, the dynamic aspects of all submitted models would have to be expressed fully by the same kind of diagrams (e.g., a state chart diagram, a sequence diagram, etc.) for the same kind of target (e.g., whole or part), so that they could be compared in the contest.

- The evaluation factors were not varied enough.

  Reason: All of the model evaluation factors evaluated combinations of various aspects of the content or presentation in an integrated way. This made it difficult to evaluate these aspects individually.

  Remedy: Independent evaluation standards for each facet of the content or presentation can be decided ahead of time, and quantitative measurements can be made based on these standards.

(3) The reason due to the competition specifications and its potential remedy by the steering committee are as follows.

- The requirements were so few and simple that the participants could develop the software without adequate modeling.

  Reason: The functional and non-functional requirements imposed on the software by the contest specifications were few and simple, and the scale of the required software was small. This suggests that the requirements could be met through a traditional approach to development, in which implementation was attempted without adequate analysis and design in terms of a model.

  Remedy: The participants could be asked to develop larger-scale software by adding more functions or by making the driving course more complex. It is necessary, though, to give adequate consideration to the level of difficulty for first-time participants, and to the scale of software that the hardware can support.

6 Conclusion and future work

In this paper, we have presented the results of evaluating the embedded software models submitted to the ET Software Design Robot Contest in 2005, and the results of evaluating the performance of the systems using this software. We created a framework for evaluating the software quality that integrated these two results, and by using that framework, we examined whether there was a relationship between them. We found that the quantitative measurement of the structural complexity of the design model had a strong relationship to the qualitative evaluation of the design as judged by us. We also found, however, that the qualitative evaluation and the structural characteristics of the models had no clear relationship with the performance of the embedded systems incorporating the software, which was expected to be based on these models. This suggests that the methods used in the contest to evaluate and measure the structural characteristics of the software were inadequate to obtain an internal measure of its reliability or efficiency. To adequately measure these non-functional characteristics in terms of the model, it will be necessary to reexamine the evaluation methods. Moreover, we think that the participants have to avoid the lack of correspondence between the model and the program code by applying model-driven development and performance patterns in developing the robot software.

In 2006, we held the contest in the same way and obtained similar results (overall, the qualitative model evaluations did not reveal the robots' reliability or efficiency), since the detailed analysis of the 2005 contest presented here was conducted in parallel with running the new contest. We are planning, however, to continue holding similar contests in the following years. Based on the lessons learned (including the reasons why we could not measure software reliability and efficiency from the models, and the corresponding remedies) from these evaluation experiments, we will consider ways to run the contest that will make it more practical and educational, including obtaining an appropriate relationship between the model and the actual performance, as

discussed here.

Also, by examining similar model contests and embedded software/system development projects, we have a plan to investigate the generality of the issues considered in this paper. Although the development in the contest covers many of the common activities in embedded system development (such as controlling hardware devices and using external sensors), except for designing hardware, some of the issues and lessons learned in this paper might not apply with industrial embedded development, because the main objective of the contest was to provide an educational opportunity. Not all of the participants of the contest were fully professional developers, and the product of the contest was not meant for practical or commercial use.

Acknowledgements

This research was partially supported by a research grant from the Japanese Society for Quality Control. Our thanks go to all of the staff and participants in ET Robocon for giving us this opportunity to conduct experimental evaluations. Moreover, we thank the anonymous reviewers for their valuable comments.

References

[1] H. Washizaki, Y. Kobayashi, H. Watanabe, E. Nakajima, Y. Hagiwara, K. Hiranabe and K. Fukuda, Experiments on Quality Evaluation of Embedded Software in Japan Robot Software Design Contest, *Proc. 28th International Conference on Software Engineering* (*ICSE2006*), pp. 551-560, 2006.

[2] Embedded Technology (ET) Software Design Robot Contest; `http://www.etrobo.jp/`

[3] Japan Embedded Systems Technology Association; `http://www.jasa.or.jp/top/en/`

[4] The LEGO Group, LEGO Mindstorms; `http://mindstorms.lego.com/`

[5] ISO/IEC 9126-1, *Software engineering - Product Quality - Part 1: Quality model*, 2001.

[6] K. Lee and S. J. Lee, A Quantitative Software Quality Evaluation Model for the Artifacts of Component Based Development, *Proc. 6th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, and 1st ACIS International Workshop on Self-Assembling Wireless Networks*, 2005.

[7] OMG: Unified Modeling Language (UML); `http://www.uml.org/`

[8] UML Robot Contest; `http://www.otij.org/event/umlforum/2003/robocon/`

[9] Y. Kobayashi, T. Yamazaki, T. Futagami and H. Watanabe, Robot Contest as Software Engineering Education, *Proc. IPSJ/SIGSE Embedded Software Symposium* (*ESS2004*), 2004. (in Japanese)

[10] OMG, *Software Process Engineering Metamodel Specification, Version 1.1*, 2005.

[11] Change Vision, Inc., UML Modeling Tool – JUDE; `http://jude.change-vision.com/jude-web/`

[12] Omondo, EclipseUML; `http://www.eclipse-uml.com/`

[13] D. West, Metaphor, Architecture, and XP; *Proc. 3rd International Conference on Extreme Programming and Agile Processes in Software Engineering*, 2002.

[14] J. Herbsleb, D. Root and J. Tomayko, *The eXtreme Programming (XP) Metaphor and Software Architecture*, CMU-CS-03-167, Carnegie Mellon University, 2003.

[15] ISO/IEC TR 9126-2, *Software engineering - Product Quality - Part 2; External metrics*, 2003.

[16] ISO/IEC TR 9126-3, *Software engineering - Product Quality - Part 3: Internal metrics*, 2003.

[17] M. Lorenz and J. Kidd, *Object-Oriented Software Metrics*, Prentice Hall, 1994.

[18] F.B. Abreu, M. Gonlao and R. Esteves, Toward the Design Quality Evaluation of Object-Oriented Software Systems; *Proc. 5th International Conference on Software Quality*, 1995.

[19] R. Banker, S. Datar, C. Kemerer and D. Zweig, Software Complexity and Maintenance Costs; *Communications of the ACM*, Vol. 36, No. 11, 1993.

[20] W. P. Stevens, G. J. Myers and L. L. Constantine, Structured Design; *IBM Systems Journal*, Vol. 13, No. 2, 1974.

[21] A. Brown, An Introduction to Model Driven Architecture; *IBM developerWorks*, May, 2004; `http://www-128.ibm.com/developerworks/rational/library/3100.html`

[22] B. Selic, The Pragmatics of Model-Driven Development, *IEEE Software*, Vol. 20, No. 5, 2003.

[23] OMG, *UML Profile for Schedulability; Performance, and Time* (*version 1.1*), 2005.

[24] M. Marzolla and S. Balsamo, UML-PSI: The UML Performance SImulator; *Proc. 1st International Conference on the Quantitative Evaluation of Systems*, 2004.

[25] S. Balsamo, A. Di Marco, P. Inverardiand and M. Simeoni, Model-Based Performance Prediction in Software Development; A Survey, *IEEE Transactions on Software Engineering*, Vol. 30, No. 5, 2004.

[26] M. Woodside, D. C. Petriu, D. B. Petriu, H. Shen, T. Israr and J. Merseguer, Performance by Unified Model Analysis (PUMA); *Proc. 5th ACM Workshop on Software and Performance*, 2005.

[27] Y. Nagase, Development methodologies and processes, @IT, January, 2003. (in Japanese), `http://www.atmarkit.co.jp/im/carc/serial/column/nagase02/nagase02.html`

[28] T. Winn and P. Calder, Is This a Pattern?; *IEEE Software*, Vol. 19, No. 1, 2002.

[29] H. Washizaki, A. Kubo, A. Takasu and Y. Fukazawa, Relation Analysis Among Patterns on Software Development Process; *Proc. 6th International Conference on Product Focused Software Process Improvement, LNCS*, Vol. 3547, 2005.

[30] W. J. Brown, R. C. Malveau, H. W. McCormick and T. J. Mowbray, *AntiPatterns: Refactoring Software; Architectures, and Projects in Crisis*, Wiley, 1998.

[31] C. U. Smith and L. G. Williams, Software Performance AntiPatterns; *Proc. 2nd International Workshop on Software and Performance*, 2000.

[32] B. Selic, G. Gullekson and P. Ward, *Real-Time Object Oriented Modeling*, John Wiley & Sons, 1994.

[33] S. M. Yacoub, H. H. Ammar and R. Robinson, Dynamic Metrics for Object Oriented Designs; *Proc. 6th International Software Metrics Symposium*, 1999.

[34] M. Genero, D. Miranda and M. Piattini, Defining Metrics for UML Statechart Diagrams in a Methodological Way; *Proc. International Workshop on Conceptual Modeling Quality*, 2003.

Hironori WASHIZAKI

is an assistant professor at National Institute of Informatics, Tokyo, Japan. He obtained his Doctor's degree in Information and Computer Science from Waseda University in 2003. His research interests include software reuse, quality assurance and patterns. He has published more than 35 research papers in refereed international journals and conferences. He received SES2006 Best Paper Award from IPSJ/SIGSE, and Takahashi Encouraging Award from JSSST in 2004. He has served as member of program committee for several international conferences including REP'04, ASE'06, Profes'04-07 and APSEC'07. He is also a member of IEEE, ACM, IEICE, JSSST, IPSJ and Hillside Group.

Yasuhide KOBAYASHI

is a president of Afrel co., Ltd., Fukui, Japan. He has worked on research, development and provision of software curriculums for educational institutions and engineers. His main research is an effective approach for young people learning software. He has applied software engineering to education, and practiced educational methods, based on pedagogy, with use of robot control and contests. He has served as a chairman on the steering committee of "ET Software Design Robot Contest", an international committee member of "WRO (World Robot Olympiad)", and a contest judge of "MDD (Model Driven Development) Robot Challenge". He is also a member of IPSJ/SIGCE.

Hiroyuki WATANABE

is with the Osaka Gas Information System Research Institute Co.,Ltd. (OGIS-RI Co.), Japan.

Eiji NAKAJIMA

is a chief editor consultant at Mamezou company, in Tokyo, Japan. He obtained his bachelor's degree in information engineering from Toyo University in 1987, and was active as a software engineer at a precision instrument maker. He wrestled with object-oriented development since 1995 and wrestled with productivity improvement in software development. He has experiences in supporting the improvement reform activity at 15 or more companies by using object-oriented development, the development process improvement, and so forth.

Hiroyuki HAGIWARA

is a manager of Software Div of CATS Corporations, Yokohama, Japan. He has passed through development of a CASE tool for embedded system and consultation for design methodology, and engages in marketing. His recent interests include Requirements Engineering and Human Centered Design.

Kenji HIRANABE

is CEO of Change Vision, Inc. Kenji is a Japanese software development consultant and book translator of Lean Software Development, Agile Project Management, and other Agile books. He's also an author of Jude, a UML and MindMap editor software. You can read his blog at http://jude-users.com/en/modules/weblog/index.php?user_id=5.
He specializes in agile development, object-oriented software construction and project facilitation.

Kazuya FUKUDA

is with the Toyoko System Corporation, Japan.