Experiments on Quality Evaluation of Embedded Software in Japan Robot Software Design Contest

Hironori Washizaki  
National Institute of Informatics  
2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo, Japan  
washizaki@acm.org

Yasuhide Kobayashi  
Afrel, Inc.  
3-111, Tonya-cho, Fukui, Japan  
yosu@esm.co.jp

Hiroyuki Watanabe  
Ogis-RI Co., Ltd.  
4-13-23, Shibaura, Minato-ku, Tokyo, Japan  
Watanabe@ogis-ri.co.jp

ABSTRACT

As a practical opportunity for educating Japanese young developers in the field of embedded software development, a software design contest involving the design of software to automatically control a line-trace robot, and conduct running performance tests was held. In this paper, we give the results of the contest from the viewpoint of software quality evaluation. We create a framework for evaluating the software quality which integrated design model quality and the final system performance, and conduct analysis using the framework. As a result of analysis, it is found that the quantitative measurement of the structural complexity of the design models bears a strong relationship to qualitative evaluation of the design conducted by judges. It is also found that there is no strong correlation between design model quality evaluated by the judges and the final system performance. For embedded software development, it is particularly important to estimate and verify reliability and performance in the early stages, using the model. Based on the analysis result, we consider possible remedies with respect to the models submitted, the evaluation methods used and the contest specifications. In order to adequately measure several non-functional quality characteristics including performance on the model, it is necessary to improve the way of developing robot software (such as applying model driven development) and reexamine the evaluation methods.

Categories and Subject Descriptors

D.2.8 [Software Engineering]: Metrics—product metrics, complexity measures, performance measures

General Terms
Measurement

Keywords
Software quality, robot contest, embedded software development, software design, software model

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

1. INTRODUCTION

As a learning opportunity in analysis and design modeling technology in the field of embedded software development, and to provide an opportunity to experience the entire development process in a short period of time, the ET software design robot contest (hereafter contest) was held by the Japan System House Association[1] in Tokyo, 2005. 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, or 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 the code or representation of the problem. As the requirements for embedded software have become more complex, larger scale, and have begun to include higher reliability, the need for education regarding the application of analysis and design modeling in this broader sense has increased. The contest was planned as an opportunity to respond to this demand.

Entrants participated in teams over the approximately three-month period from April to June, first modeled the design of embedded control software for a line-trace robot and then implemented the software based on the model. The robot, built with LEGO Mindstorms blocks[2], was required to move independently and following a dark line. On the contest competition day in July, participants tested the overall adequacy of each robot as an embedded system by verifying the independent operation of the robot. 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 using the robot running performance results (time to complete the regulated course).

In this paper, we report the result of experiments on quality evaluation of embedded software. First, we assumed that the better robot design model quality is, the better robot runtime performance is. Second, we carefully analyzed the relationships between these two evaluation results. Finally, we will discuss the nature of the models that correspond to systems’ performance, and appropriateness of model evaluation methods used in the contest.

As a result of analysis, it is found that the quantitative measurement of the structural complexity of the design models bears a strong relationship to qualitative evaluation of the design judged by us. It is also found that there is no strong correlation between design model quality judged
by us and the final system performance. It is well known that many problems in software development, not limited to embedded software, are introduced during the analysis and design work, and that these problems exert a dominating effect on software and system quality characteristics\cite{3}. For embedded software development where real-time performance is required, it is particularly important to estimate and verify reliability and performance in the early stages, using the model. In order to adequately measure several non-functional quality characteristics including performance on the model, it is necessary to reexamine the evaluation methods. Moreover, we think that participants have to avoid the lack of correspondence between model and program code by applying model driven development and performance patterns to the way of developing robot software.

The remainder of this paper is organized as follows. Next section introduces an overview of the contest. Section 3 describes the model/running performance evaluation methods and results in the contest. Section 4 presents a quality evaluation framework to integrate both model and running performance evaluations. In section 5, we report on the results of analyzing relations among them based on the framework. In the last section, we draw a conclusion and state future works.

2. CONTEST OVERVIEW

The contest has been held four times, once a year from 2002 to 2005. 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. The hardware and software regulations and the evaluation methods are described below.

- Hardware: The hardware is regulated as a robotic vehicle assembled from LEGO Mindstorms blocks. Figure 1 shows an overview of the robot’s hardware. The robot is capable of moving forwards, backwards and turning left or right under motor control, and recognizing a course indicated by a black solid or dashed line using a light sensor to detect brightness on the running surface.

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

- Software: 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 of approximately 20 meters 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 UML\cite{4} diagrams and accompanying descriptions or pictures. Software was implemented in languages capable of controlling Mindstorms robots, including C, C++, and Java.

- Evaluation: Judges evaluated both the models created as a result of software analysis and design process, and the results (run-time and rank) achieved by the robot (the embedded system including hardware and software) in a running performance test.

It has been the intention of this contest that participants learn technology, including object-oriented embedded software analysis and design modeling using UML, through their participation in the competition. Modeling-technology training sessions and workshops to discuss the models and their performance were held, and the contest results were published. It has been clear from questionnaire results after previous contests (held in 2002-2004 as “UML robot contest”) that the previous contests were educational for participants learning these development technologies\cite{5}.

3. EVALUATION OF SOFTWARE QUALITY IN THE CONTEST

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

3.1 Evaluation of models (judging standard)

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

- Content: 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 relations/structures among elements from an object-oriented perspective. For example, we considered the adequacy regarding the problem divide-and-conquer, role division and assignment, level of abstraction, relations, and multiplicity.
  - Logical correctness: We judged the feasibility of requirements described in models. For example, we considered the correctness of algorithm and behavior models.
  - Originality of the algorithm: We judged the uniqueness of the described algorithm such as the dynamic sensor calibration functionality and return functionality (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 correctness: We checked whether the described models correctly follow the UML (or any other diagram) specifications.
  - Clarity: We judged the clarity (understandability) of the model by checking the adequacy regarding the separation of views, layout, etc.
  - Originality and ingenuity: We judged the uniqueness of the described model including the accompanying descriptions/pictures.

Qualitative evaluations were made by the judges, with the final decisions given by a two-step process. First, the individual evaluations from the six judges were tallied, and then a group evaluation was decided after discussion within the group. This process was conducted to eliminate biases of individual judges.

### 3.2 Model evaluation results

Figure 2 shows the number of models receiving each evaluation grade as described above. As shown, most of the models received B or C grades. Fewer models received A or D grades. As examples, the structure part and behavior part of the model which was considered the best in the A group by the judges are shown in the form of UML in Figure 3 and Figure 4 respectively.

Several trends with respect to all of the models during the evaluation were observed and are described below.

- Validity: Most of the models were broken into a number of classes through a divide-and-conquer approach; however, some classes with too much functionality were seen. Further, models which could determine the type of a given segment of the course (e.g. straight line, curved) were considered more complete than those which simply followed the line. For example, Figure 3 shows abstract classes (Course type and Drive method) for course type and driving method, allowing the driving approach used to be set based on 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: There were several different driving strategies 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 the section of the course. There were also models which described and verified their development process in addition to the software and hardware products.
- Presentation: For about half of the models, the UML notation was not sufficiently correct or adequate. For example, there were UML diagrams where attributes or multiplicity was not given, relation or role names were not appropriate, or the target of a state diagram was not specified. In terms of clarity, it was easy 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 otherwise. Also some of the models over-used metaphor[6, 7], with the result that the intention of their analysis and design was obscured.
3.3 Running performance evaluation method

The judges evaluated the robot running performance of all participating teams whether they could complete the course within a specified time. Teams that completed were also evaluated based on their fastest completion time, and their rank based on this time.

3.4 Running performance results

Of the 47 teams which made submissions, the robots from 15 (31.9%) of the teams completed the course. The times and rankings of the teams which completed are shown in Figure 5. The fastest time was 35.4 [sec], and the slowest time was 112.3.

4. INTEGRATION OF MODEL AND RUNNING PERFORMANCE EVALUATIONS

In order to discuss both the model and running performance results uniformly, a framework to integrate the two results was needed. In the following sub-sections, we will discuss the operational process of the contest and its background, and based on this process, construct a framework.

4.1 Contest process

In order to shed light on the relationship between the results of the model evaluation and the running performance, we will model the contest process with a UML activity diagram based on the Software Process Engineering Metamodel (SPEM)[8]. The resulting process model with several additional particular comments is shown in Figure 6. Here, the participating teams, judges and steering committee of the contest are drawn as process participants, along with the activities and inputs that they contribute. For example, each participating team conducted 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.

4.2 Integrated quality evaluation framework

From the viewpoint of software-quality measurement[9], the running performance evaluation described in the process above is a kind of external measurement[10] of the software embedded in the robot. An external measurement is an activity which measures the 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 the running performance can be attributed to differences in the software. On the other hand, the model evaluation is a kind of internal measurement[11] of the software quality. An internal measurement is an activity which measures the software quality by assigning a quantitative value to some property of the intermediate or final software (model or program).

Using the various quality characteristics required by the ISO 9126[9] software quality model, we applied internal and external measurements to build a systematic framework for the software quality evaluation. This framework is shown in Table 1. The software quality characteristics used in the contest are summarized here:

- Functionality: Although not clearly specified as a judging standard, whether the software satisfied the functional requirements was included in the judges’ evaluation of the model validity. Thus, internal measurement of the functionality is considered to consist of the parts of the model evaluation that reflect validity of the model.
- Maintainability: Software maintainability is related to clarity and notational correctness. Internal measurement of maintainability is considered to consist of the parts of the model evaluation related to presentation (particularly clarity and correctness).
- Reliability: Reliability was externally measured by whether the robot was able to complete the test course under the control of the software. It could also be measured internally through an evaluation of logical correctness of the model, but as noted above, we were not able to verify the logical correctness of the models submitted.
- Efficiency: The course completion time and rank of the robot under control of the software give an external measurement of the quantity of time resources it uses.
- 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 contest participants developed new software for a single hardware environment, unrelated to the results of previous or future contests.
Figure 6: Workflow of contest process described in the form of UML activity diagram with SPEM

Table 1: Constructed software quality evaluation framework

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Factor to be shown</th>
<th>Internal measurement (Model evaluation)</th>
<th>External measurement (Running evaluation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality</td>
<td>Required functions are supported</td>
<td>Validity of the content</td>
<td>Whether the course was completed</td>
</tr>
<tr>
<td>Reliability</td>
<td>Functions behave normally under all conditions</td>
<td>Logical correctness</td>
<td></td>
</tr>
<tr>
<td>Usability</td>
<td>Intuitiveness/usability</td>
<td>–</td>
<td>Running time/rank</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Use of time, resources</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>Maintainability</td>
<td>Effort required for maintenance</td>
<td>Presentation</td>
<td>–</td>
</tr>
<tr>
<td>Portability</td>
<td>It can be ported to a different environment</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
5. RELATION BETWEEN THE MODEL EVALUATION AND RUNNING PERFORMANCE EVALUATION

In the framework we created in the previous section, we drew connections between the model and performance evaluation results and various quality characteristics. Note particularly that the reliability and efficiency measurements were based on the running performance evaluation and not 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 the analysis and design work, and that these problems exert a dominating effect on software and system quality characteristics[3]. Therefore, it is expected that the value of external quality based on executable software systems can be estimated from the value of internal quality based on non-executable artifacts including models[12]. Especially for embedded software development where real-time performance is required, it is particularly important to estimate and verify reliability and efficiency/performance in the early stages, using the model.

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

5.1 Relevance of the model evaluation

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

Of all of the possible metrics applicable to a whole object-oriented model, we used the number of classes[13] and the coupling complexity (Coupling Factor: COF[14]) among the classes because all of the submitted models included a class diagram that showed the overall software architecture. If the model included several class diagrams, we looked at the diagram showing the overall driving control method.

The number of classes indicates the scale of the object-oriented software, and was measured by the number of classes in the class diagram that showed the overall architecture. COF indicates the static complexity of object-oriented software based on coupling between classes. It was measured by taking the number of one-way relations (two-way relations are counted twice) between classes, not including inheritance or dependency relationships (dotted arrows), in the class diagram. This value is now normalized for scale to a value from zero to one. The complexity, COF(S), of a collection of related classes, S, is given by the following function where #S denotes the number of classes in S.

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

For example, in Figure 5, 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.

![Figure 7: COF measurement example](image)

The number of classes and the COF measurement value for each of the 47 models submitted are shown in Figure 8. In the figure, all of the models which received an A grade had more than ten classes, and a COF value not more than 0.15. Of the B-grade models, most of them had more than 15 classes and a COF value not more than 0.16. From this we see that for models of a certain size, those with low static complexity were evaluated more highly. Conversely, most of the models with fewer classes and higher COF value were given a C grade. So, we see that small-scale models with higher static complexity received lower evaluations.

Because the roles of each class could not be determined from the class diagram due to lack of adequate explanation, models receiving a D grade can be excluded from examination of the scale and complexity of the static structure. Also, there were some models receiving a B grade but having few classes and higher complexity. These models had other diagrams (e.g. state diagrams) which express the operation of the algorithm carefully from a dynamic perspective in a way that could be understood, so they were taken separately from most of the other B-grade models, which were graded based on the adequacy of the class diagram.

When evaluating the models, we considered both the validity of the model detail based on an object-oriented perspective, as well as the adequacy of the presentation. The smallest unit of structure in an object-oriented design is a class, and the number of classes and COF value can be thought as quantitative measurements of the appropriateness of the class abstractions and the relationships between those classes. Further, 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 scale and structural complexity that are expressed by the number of classes and COF value are implicitly taken into consideration in the qualitative evaluations given by the judges. It is well known that software maintenance costs are significantly affected by the levels of software complexity[15]. Especially in module-based software, module couplings are important metrics for maintainability[16]. In the contest, the judging activity can be thought as a kind of a maintenance activity because the judges as third parties evaluated models after their implementations. Therefore, the given qualitative measurements that implicitly reflect complexity and scale can be seen as almost 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 correlation between quality and the running performance results. However, among the models actually submitted, most of those that prominently showed originality were also evaluated highly on the content, so we see that consideration of originality did not have a significant effect on the values of the final results.

Figure 8: Number of classes and COF value for all models

5.2 Expectations for the relation between model and running performance evaluation results

In the contest, it was our intention to encourage the participants to use embedded-software Round-Trip Engineering (RTE[17]), or Model-Driven Development (MDD[18]) by requiring the participants to submit their model and evaluating it before the running test was held. However, we did not actually investigate 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 that has been getting attention in industry, in which several aspects of target domain/problems are abstracted and expressed in the model, so that the relationship between model and the program code is maintained as development progresses. By applying RTE, increased quality and maintainability (including modifiability) can be expected because any problems are detected early on the abstract model due to the close relationship between the model and its implementation.

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

As shown in Table 1, the results of external measurements of the software reliability and efficiency come from the results of the running performance test. Thus, if the participants used RTE or MDD and created a model and program with an adequate relationship, then the robots of participants whose models were evaluated better for both reliability and efficiency quality attributes would be expected to actually complete the regulated course more quickly.

5.3 Verification of the relationship

Based on the above expectations, we examined the relationship between the model evaluation results and the running performance results, assuming that the performance results represent an external measurement of reliability and efficiency.

The evaluation method discussed in section 3.1, did not explicitly include evaluation from an efficiency perspective. Neither could we determine whether the models submitted were likely to show high reliability by evaluating their logical correctness. However, it might be still possible that the evaluations which considered validity of the model content and adequacy of presentation reflect their 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 scale, structure, inter-object communication or state transitions. This type of excess complexity can ultimately lead to decreases in reliability and efficiency of the software.

Thus, we have verified the relationship between the scale and static complexity in the model evaluation results, and the running performance results. The COF values and the performance ranks are shown in Figure 9 along with evaluation grades for the 47 models submitted. Similarly, the number of classes and performance rank are shown in Figure 10. Data for teams that did not complete the test course are shown under “Non-completing”.

Figure 9: COF and running result

Figure 10: Number of classes and running result
In Figure 9, there is no recognizable relationship between the performance ranks of completing teams and the COF values. No trends in the COF values can be seen which distinguish the completing from the non-completing teams. Further, more than half of the models which received an A grade did not complete the course, and for the ones that did, rankings were low compared to others which received B and C grades. There was also no recognizable relationship between performance rank or completion vs. non-completion for models receiving B, C and D grades. Similarly, Figure 10 shows no recognizable relationship between model scale and performance rank or evaluation grade.

The correlation matrix for the number of classes (denoted as “N. classes”), COF value, model evaluation (denoted as “Eval. grade”) and running performance rank (denoted as “Run result”) for the 15 completing teams are shown in Table 2. To calculate the correlation coefficients between the grades A to D and the other values, they were assigned the numbers 1 (grade A) to 4 (grade D), respectively. From Table 2, we can see that the number of classes, which gives an internal measure of a software quality characteristic, has a strong correlation with both the COF value and the evaluation grade. The weak correlation between COF value and evaluation grade is attributed to the fact that many of the models with a low COF value (good in terms of complexity) did not complete the driving course. There was no recognizable strong correlation between the performance rank and any of the other measurements (number of classes, COF value or evaluation grade).

From the above results we see that there was no clear relationship between the qualitative evaluation or structural characteristics of the software models in the contest, and the running performance results of the embedded system software. Thus we conclude that the evaluation methods and measurements of structural characteristics (number of classes and COF) of the models do not reflect reliability or efficiency of the software, and so, do not provide adequate internal measurements of these quality characteristics.

Table 2: Correlation matrix for model-evaluation results and running results for completing teams

<table>
<thead>
<tr>
<th>N. classes</th>
<th>COF</th>
<th>Eval. grade</th>
<th>Run result</th>
</tr>
</thead>
<tbody>
<tr>
<td>N. classes</td>
<td>1</td>
<td>-0.732</td>
<td>1</td>
</tr>
<tr>
<td>COF</td>
<td>-0.711</td>
<td>0.288</td>
<td>1</td>
</tr>
<tr>
<td>Eval. grade</td>
<td>0.353</td>
<td>-0.564</td>
<td>0.093</td>
</tr>
</tbody>
</table>

5.4 Reasons and remedies

We will now consider reasons why we were not able to measure software reliability and efficiency from the models using internal measurements, and possible remedies with respect to the models submitted, the evaluation methods used and the contest specifications. The relationships between the remedies suggested by this examination are shown in a feature model[19] in Figure 11.

The remedies can be divided into three types: (1) by participants, (2) by judges, and (3) by steering committee. Details are shown below.

1. Reasons related to the models submitted, and remedies by participants are the followings.

- Dynamic models were inadequate.
  
  **Reason:** In many of the models submitted, the hardware control and driving strategy were not modeled, and logical correctness that would likely be reflected in high reliability could not be judged.
  
  **Remedy:** Participants can express dynamic software behavior and algorithms.

- There was a lack of correspondence between model and program.
  
  **Reason:** In typical embedded software development, often efficiency is refined by iterating through the implementation and testing work. It is possible that contest participants adopted this kind of approach. However, without maintaining the applied relationship between the software implementation (the program) and the model, the result of this type of iterative work will not be reflected in the submitted model.
  
  **Remedy:** Each team could adopt RTE or MDD as their basic development approach in order to maintain the relationship between model and program while generating the final program. Further, MDD encourages derivation of 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 generation of more reliable programs.

- Non-functional characteristics were not specified.
  
  **Reason:** None of the models submitted 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 is not effective for expressing non-functional characteristics.
  
  **Remedy:** UML performance profile[20] or other performance frameworks[21, 22] can be used to add special annotations regarding non-functional characteristics into the standard models. For example, Woodside[23] proposes a technique for adding performance annotations to UML models and transforming them into performance models for which performance can be verified.

- Techniques for handling hardware characteristics were not specified.
  
  **Reason:** While the software design, hardware characteristics, control algorithm, or combinations of these had an effect on efficiency, most participants did not know how to deal with it, or if they did, they were not able to express how they dealt with it in their models.
  
  **Remedy:** We could record some software patterns[24, 25] or anti-patterns[26] that take the hardware characteristics into consideration, and encapsulate knowledge gained from software analysis and design that successfully increased efficiency (or work-arounds for
Developing models consistent with performance

Correspondence between model and program

Specifying non-functional characteristics

Specifying how to handle hardware characteristics

Enrich dynamic models

Quantitative evaluation of performance

Multi-faceted evaluations

More complex requirements and scale enlargement

Solution for conducting practical/educational contest

Figure 11: Contest remedy feature model

(2) Reasons related to the model evaluation method and remedies by the judges are the followings.

- Evaluation of efficiency was not emphasized or quantified.
  
  **Reason**: Evaluation of efficiency was not explicitly included in the model evaluation.
  
  **Remedy**: We can emphasize evaluation of efficiency on models that also show dynamic aspects, and 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[28, 29, 30] which measure the complexity of the dynamic behavior of an object-oriented design. In order to do this, dynamic aspects of all submitted models would have to be expressed fully, and would have to be planned consistently so that they could be compared.

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

(3) A reason due to the competition specifications and its remedy by the steering committee are the followings.

- Requirements were so few and simple that participants could develop software without adequate modeling.
  
  **Reason**: The functional and non-functional requirements imposed on the software by the specifications of the contest were few and simple, and the scale of the required software was small. Further, many of the participants were evaluated highly on the running performance results, but poorly on the model. This suggests that the requirements were met through a more traditional approach to development, in which implementation was attempted without adequate analysis and design on the model.
  
  **Remedy**: Participants could be asked to develop larger-scale software by adding more functions or 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 presented the results of evaluating the embedded software models submitted to the ET Software Design Robot Contest, and the results of evaluating the performance of the systems using this software. We created a framework for evaluating the software quality which integrated these two results, and using that framework, examined whether there was a relationship between them. It was found that the quantitative measurement of the structural complexity of the design models bears a strong relationship to qualitative evaluation of the design judged by us. However, it was also found that the quantitative evaluation and structural characteristics of the models had no clear relationship with the results of testing the performance of the embedded systems incorporating the software which was purported 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 not adequate to give an internal measure its reliability or efficiency. In order to adequately measure these non-functional characteristics on the model, it will be necessary to reexamine the evaluation methods. Moreover, we think that participants have to avoid the lack of correspondence between model and program code by applying model driven development and performance patterns to the way of developing robot software.

We are planning to continue to hold similar contests in the following years. Based on lessons learned (including the reasons why we were not able to measure software reliability and efficiency from the models, and the corresponding remedy...
diaries) from the evaluation experiments, we will consider ways to run the contest that will make it more practical and educational, including the appropriate relationship between the model and actual performance as discussed in this paper.

Also, by examining similar model contests and embedded system development projects, we have a plan to investigate the generality of the issues considered in this paper. Although this paper has dealt with relatively simple robot systems, we expect that many of these issues and lessons learned can be also applied to more complicated embedded systems because the development of such robot systems covers common activities in embedded system development (such as controlling hardware devices and using external sensors) except for designing hardware itself.

7. ACKNOWLEDGMENTS

Our thanks go to all of staff and participants in ET Robocon for giving us this opportunity to conduct experimental evaluations.

8. ADDITIONAL AUTHORS

Additional authors: Eiji Nakajima (Mamezou Co., Ltd., email: eiji@mamezou.com), Yuji Hagiwara (CATS CO., LTD., email: hagiwara@zipc.com), Kenji Hiranae (Change Vision, Inc., email: kenji.hiranabe@change-vision.com) and Kazuya Fukuda (Toyoko System Corporation, email: fukuda@toyoko-sys.co.jp).

9. REFERENCES