A method of knowledge sharing for controlling hardware in automotive software development

Kaori Hayashi
Technical Resources Dept.
DENSO CREATE INC.
Aichi, Japan
hayashi_ka@dcinc.co.jp

Kenji Fujimoto
Modeling Department
Change Vision, Inc.
Fukui, Japan
kenji.fujimoto@change-vision.com

Midori Daida
Modeling Department
Change Vision, Inc.
Fukui, Japan
midori.daida@change-vision.com

Nobuhide Kobayashi
Business Development Dept.
DENSO CREATE INC.
Aichi, Japan
nobuhide@dcinc.co.jp

Shuji Morisaki
Graduate School of Informatics
Nagoya University
Aichi, Japan
0000-0002-8290-0584

Shuichiro Yamamoto
Graduate School of Informatics
Nagoya University
Aichi, Japan
syamamoto@acm.org

***Abstract*—Skilled engineers often formalized their useful experiences as a checklist, but it might not be effectively utilized by non-skilled engineers. In order to solve this issue, this paper proposes the method of knowledge sharing that can provide the necessary and sufficient knowledge to non-skilled engineers. It consists of the highly reusable knowledge structure and the description pattern of formalizing knowledge. In order to realize this method, this paper shows the proposed knowledge classification method using embedded system as the reference model. Also the effectiveness of this method was confirmed based on the evaluation result.**

Keywords—automotive, description pattern, knowledge structure, knowledge sharing, reusable

I. INTRODUCTION

Automotive software engineers are always expected to reduce development time, even if the scale and complexity of automotive systems are significantly increasing. As a result, skilled engineers have no enough time to consider the best way of formalizing their experience as reusable knowledge. For this reason, the formalized knowledge might not be necessarily appropriate for non-skilled engineers, and is not effectively utilized by them. This paper clarifies the following as issues on utilizing conventional knowledge from the result of analyzing a check list and so on used by actual automotive software development projects.

- Issue 1) Checklists often indicate only engineer's works corresponding to "means" like confirming a resister condition, ones do not indicate completed software state as the result of their works corresponding to "purpose". Therefore, it is difficult for non-skilled engineers to understand the purpose of works from conventional knowledge. It is necessary to clarify the relationship between why, what, and how from the viewpoint of software state.
- Issue 2) Checklists are often written with various viewpoints such as software viewpoint or hardware viewpoint, ones cause confusion to understand for non-skilled engineers. It is necessary to adopt the description pattern that can uniform the expression of formalized knowledge.

This paper proposes the highly reusable structure of knowledge and the description pattern derived from actual development results.

The rest of this paper is structured as follows. Section II describes related works for knowledge sharing. Section III proposes the method used to formalize knowledge. Section IV explains the evaluation result of the proposed method. Section V concludes the paper and discusses future work.

II. RELATED WORK

Regarding issue 1, It is important to clarify the way of thinking on automotive software development in order to define the reusable knowledge structure. The Open Group standardizes ArchiMate[1] as the modeling language of enterprise architecture. ArchiMate provides Motivation viewpoint of marshalling the relationship between stakeholder's goal, issues for their goal, and solutions for each issue. It is likely to be useful, so the method adopted as a reference model to formalize knowledge is proposed [2]. However, it has not sufficiently considered the relationship of why, what, and how, in automotive software development. And it has not also considered unification of knowledge expression. As other ways of marshalling engineer's thinking are Requirement diagrams of SysML[3] and USDM[4]. Those methods can also define the relationship between the above mentioned elements as well as Motivation view. However, Motivation view is more appropriate on the viewpoint of defining concrete knowledge, because the abstraction level of notation provided with Motivation view is more concrete than other methods. Regarding issue 2, as the method to unify descriptions,

978-1-7281-2627-2/19/$31.00 ©2019 IEEE
DOI 10.1109/IIAI-AAI.2019.00171

Authorized licensed use limited to: Universidade Tecnologica Federal do Parana. Downloaded on January 06,2024 at 16:21:28 UTC from IEEE Xplore. Restrictions apply.

EARS(Easy Approach to Requirements Syntax) is so famous[5]. But it has not been adopted to knowledge description.

Additionally, it is important to clarify the target structure related to knowledge dealt with in this paper in order to cope with skilled engineer's thinking. Seven Samurai framework[6] shows necessary and sufficient elements, and their relationship to be considered in system development. Also, automotive system development field, EAST-ADL[7] is standardized as the reference model that can correspond to all development processes. However, it has too many elements and complex relationships, it is difficult for non-skilled engineers to understand it.

As it is important that this proposal can be handled by non-skilled engineers, it is necessary to concentrate the knowledge scope that is expected by actual development projects. Therefore, it adopts the simplified model of the embedded reference model[8] that defines the relationship of embedded software, sensor, actuator and so on, shown in Fig. 1. Because it can express the overview of EAST-ADL quite simply.

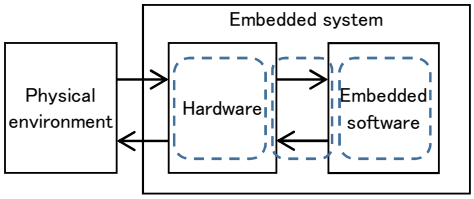

Fig. 1. The simplified reference model of embedded system

This paper explains the appropriate knowledge structure considered from the combination of the way of thinking and target structure, in section III A. Additionally, it explains the description pattern to uniform the formalized knowledge defined from the development results in actual development projects.

III. PROPOSAL METHOD

This section explains the knowledge structure and the description patterns of its elements. Section A shows the knowledge structure which can solve issue 1 shown in section I, and section B shows the description pattern for knowledge which can solve issue 2 shown in section I. Those methods can support non-skilled engineers to effectively share knowledge in automotive software development.

A. The knowledge structure

As shown in Fig. 1, an embedded system is composed of hardware and embedded software, and embedded software inputs changes in the physical environment via hardware, and outputs the corresponding actions to the physical environment via hardware. To get highly reusable knowledge of embedded software development, it is necessary to marshal skilled engineer's implicit knowledge related to hardware, the relationship between hardware and embedded software, and embedded software. The following shows the classification to marshal its implicit knowledge.

- Events on hardware
- Impacts on embedded software
- Countermeasures on embedded software

Events on hardware mean characteristics of hardware that negatively affect system control. Impacts on embedded software mean negative effects on embedded software caused by hardware events. Countermeasures on embedded software mean ways to avoid impacts on embedded software.

TABLE I. shows the necessary classification for effective sharing knowledge in automotive software development. Applied condition is used to narrow only knowledge corresponding to a target product from many formalized knowledge, it is defined based on the meta-model that is marshaled characteristics of the target product group. Fig. 2 shows the basic structure of the meta-model in this paper. It defines a functional requirement with reference to the requirement structure[9], it consists of Input Data, Output Data, Input Trigger, Output Trigger, and processing. Input Data connects to Output Data of other functional requirement. Input Trigger connects to Output Trigger of other functional requirement. Additionally, each element in the meta-model has parameters related to Connected Target and Connected Method if ones connect to Hardware or out of scope of development target.

Finally, the meta-model needs to be refined according to characteristics of target product in each development organization when this method is introduced.

TABLE I. KNOWLEDGE CLASSIFICATION FOR EFFECTIVE SHARING

Classification	Explanation
Applied condition	Characteristics of embedded software corresponding to each knowledge
Events on hardware	Uncontrollable characteristics on hardware, it sometimes negatively affects embedded software.
Impacts on embedded software	Negative effects for embedded software, caused by uncontrollable characteristics on hardware
Countermeasures on embedded software	Processing to be implemented to avoid the above mentioned impacts on embedded software.

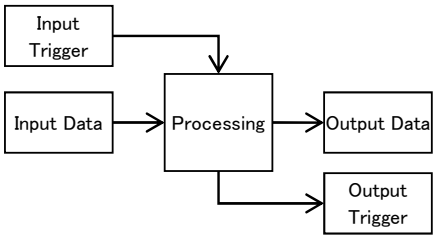

Fig. 2. Basic structure of the meta-model dealt with this paper

B. The description pattern for knowledge

As shown in Fig. 3, the formalized knowledge is used in two steps. In the first step, the user refers to it. In the second step, the user implements it to software. The following shows technical points corresponding to each step.

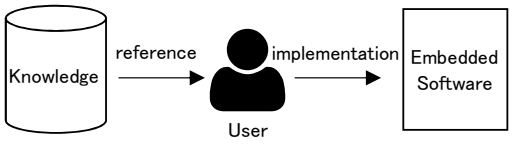

Fig. 3. The usage scene of knowledge

Authorized licensed use limited to: Universidade Tecnologica Federal do Parana. Downloaded on January 06,2024 at 16:21:28 UTC from IEEE Xplore. Restrictions apply.

The first point is the adoption of description pattern for the reference in Fig. 3, in order to prevent the following issues. Knowledge is formalized by more than one person. But it is difficult to read it, if its expression varies depending on each person.

In this paper, necessary information for the expression of formalized knowledge is extracted from analysis results for 15 cases, and the description pattern is defined as a pattern that can describe the information. The following shows the description pattern for each classification of TABLE I. . Additionally, in the following section, < > means a term that is replaced with a concrete term in a target product. (Term A/Term B/…) means to select one from the terms.

1) Event on hardware

The following shows the description pattern for Event on hardware. <Hardware> means a target controlled by embedded software. <Event> means a characteristic event that occurs in the physical environment and affects implementations on embedded software. And <Trigger> means a timing when expected <Event> does not occur or unexpected <Event> occurs. Finally, <Matter> means a reason for causing <Event>.

- <Hardware> (causes/might cause) <Event>.
- When <Trigger> occurs, <Hardware> (causes/might cause) <Event>.
- When <Trigger> occurs, <Hardware> (does not cause/might not cause) <Event>.
- <Hardware> (causes/might cause) <Event> by <Matter>.

  <Hardware> (does not cause/might not cause) <Event> by <Matter>.

2) Impacts on embedded software

The following shows the description pattern for Impacts on embedded software. <Impact> means undesirable processing executed by embedded software, <Matter> means a reason for causing <Impact>.

- Embedded software (performs/might perform) <Impact>.
- Embedded software (does not perform /might not perform) <Impact>.
- Embedded software (can not perform/might not be able to perform) <Impact>.
- Embedded software (performs/might perform) <Impact> caused by <Matter>.
- Embedded software (does not perform/might not perform) <Impact> caused by <Matter>.

  Embedded software (can not perform/might not be able to perform) <Impact> caused by <Matter>.

3) Countermeasures on embedded software

The following shows the description pattern for Countermeasures on embedded software. <Countermeasure> means implementations on embedded software to prevent <Impact>. <Trigger> means a timing when embedded software should execute <Countermeasure>, <Condition> means a time period or a status to execute <Countermeasure>.

- Embedded software should execute <Countermeasure>.
- Embedded software should execute <Countermeasure> before <Trigger>.
- Embedded software should execute <Countermeasure> after <Trigger>.
- Embedded software should execute <Countermeasure> during <Condition>.

The second point is to describe no actions by an engineer but the status of embedded software that addressed the impact, for the implementation in Fig. 3, in order to prevent the following issues. It might be difficult to implement appropriate countermeasures to embedded software because actions of an engineer such as “Confirm a register value” is not clear about what to do for embedded software.

C. The procedure of formalizing knowledge

This section shows the procedure of formalizing knowledge based on the structure shown in TABLE I. from existing knowledge assets such as checklists. And TABLE II. shows the example of knowledge is formalized based on this procedure.

1) To identify hardware targeted for knowledge

Identify a target of formalizing knowledge in order to clarify what to marshal information for.

2) To marshal information based on knowledge structure

Marshal information based on knowledge structure, and detect the lack of information.

3) To add missing information

As the result of Step 2), if the lack of information exists, add information from interviews with skilled engineers and own experience to formalize information based on knowledge structure. On the other hand, if the lack of information does not exist, this step might be omitted.

4) To adopt the description pattern

Knowledge is described according to the description pattern shown in section B. And the terms in described knowledge are used from the terms in the meta-model if ones are defined in the meta-model.

5) To connect to applied condition

Connect formalized knowledge to applied condition defined in TABLE I. in order to provide only sufficient and appropriate knowledge according to the characteristics of a target.

IV. Evaluation

This section explains the evaluation of the effectiveness for this method. The following shows viewpoints.

- Viewpoint A: This method can provide the appropriate structure and the expression for formalized knowledge.
- Viewpoint B: The formalized knowledge assets adopting this method can be effectively used in actual development.

Authorized licensed use limited to: Universidade Tecnologica Federal do Parana. Downloaded on January 06,2024 at 16:21:28 UTC from IEEE Xplore. Restrictions apply.

A. *Viewpoint A: The evaluation result for the structure and the expression of formalized knowledge*

We adopted this method to 10 cases of existing knowledge, and confirmed whether it could detect the lack of necessary information in highly reusable knowledge. As the result of this evaluation, it could detect 18 lacks in 10 cases of existing knowledge. The breakdown of detected lacks is 6 for Event on hardware, 7 for Impacts on embedded software and 5 for Countermeasures on embedded software. Also the validity of this detection result was confirmed by skilled engineers.

Regarding the expression, it could be confirmed that the proposed description pattern was adopted to 15 cases different from the cases described in section III.B, and all cases could be described using its pattern.

B. *Viewpoint B: The effectiveness of this method in actual development*

To confirm the validity in actual development, the questionnaire was conducted on 26 engineers who have developed automotive software development in Japan. The questions of the questionnaire are shown below.

- This method proposes the structure of knowledge and its description pattern in the viewpoint of reusability. Do you think this structure and the knowledge expression using the description pattern is effective in your development team?
- This method can provide the only formalized knowledge corresponding to a target product, and indicate the corresponding location of each formalized knowledge. Do you think the feature of this method is effective in your development team?

As the result of this questionnaire, 13 research participants answered effectively, the rest answered a bit of effective. No participant answered not a bit of effective or not effective. The reasons of the above result are introduced below.

- Information that engineers want to know is separated into classifications, so it is easy to understand.
- Conventional methods often leak the reason for a countermeasure, but this method necessarily describes it.

V. CONCLUSION AND FUTURE WORK

It is often difficult for non-skilled engineers to effectively utilize conventional knowledge, such as checklists that are written by skilled engineers. Because they have no time to consider the knowledge structure suitable for effective utilization, as a result, existing knowledge often lacks necessary information for actual development. For this issue, this paper proposed the knowledge structure and the description pattern in order to define the effective knowledge used by non-skilled engineers. The effectiveness of those methods is confirmed based on the following two viewpoints. First is the ability to detect the lack of necessary information for effective knowledge assets. It was confirmed with the exercise using 10 cases of existing knowledge of automotive software development in Japan. Second viewpoint is the effectiveness in actual development, it is confirmed by the questionnaire for 26 engineers. Additionally, as the limitation of this paper, the effectiveness of this method is confirmed in only Japanese automotive software company.

As the future works, it is necessary to compare the formalized knowledge by adopting this proposal with conventional knowledge assets such as checklists on the viewpoint of detecting failures in actual automotive software development. As a result, we would like to quantitatively show the effectiveness of our proposal.

REFERENCES

[1] The Open Group, "ArchiMate® 3.0.1 Specification," 2017.

[2] K. Hayashi, H. Yamada, and N. Kobayashi, "The proposal of improving reusability for quality criteria using ArchiMate," *SIG-KSN*, vol. 21, 2017.

[3] Object Management Group, "OMG Systems Modeling Language version 1.4," 2015.

[4] T. Saruwatari and S. Yamamoto, "A conversion method USDM to SysML," *KBSE*, vol. 111, pp. 73–78, 2011.

[5] A. Mavin, P. Wilkinson, A. Harwood, and M. Novak, "EARS (Easy Approach to Requirements Syntax)," *Proc. IEEE Int. Conf. Requir. Eng.*, no. November 2016, pp. 317–322, 2009.

[6] J. N. Martin, "The Seven Samurai of Systems Engineering: Dealing with the Complexity of 7 Interrelated Systems," *INCOSE Int. Symp.*, vol. 14, no. 1, pp. 459–470, 2004.

[7] Atesst2 Consortium, "EAST-ADL Domain Model Specification Version V2.1.12," *Proj. Deliv.*, 2010.

[8] S. Yamamoto, "Assurance case for reference model," *Business communication*, Apr-2013.

[9] S. Yamamoto, *Basic knowledge of Requirement engineering*. Kindai Kagaku Sha, 2019.

TABLE II. EXAMPLE OF FORMALIZING KNOWLEDGE

ID.	Applied condition	Event on Hardware	Impacts on embedded software	Countermeasures on embedded software
1.	Element: Input Data - Connected Target: Sensor - Connected Method: IO	Sensor might cause chattering of output signal.	Embedded software might perform unexpected processing by changing input signal that changes in shorter than expected time.	Embedded software should execute filtering of input signal before using it.
2.	Element: Input Data - Connected Target: Nonvolatile memory - Connected Method: Serial Communication	Nonvolatile memory causes data loss by sudden power down during access processing that takes a long time.	Embedded software performs processing using undefined data caused by data loss.	Embedded software should execute confirming a condition of data before using data in Nonvolatile memory.

Authorized licensed use limited to: Universidade Tecnologica Federal do Parana. Downloaded on January 06,2024 at 16:21:28 UTC from IEEE Xplore. Restrictions apply.