Application of Requirements Prioritization Decision Rules in Software Product Line Evolution

Mari Inoki
Faculty of Informatics
Kogakuin University
Tokyo, Japan
m_inoki@cc.kogakuin.ac.jp

Takayuki Kitagawa
Software Engineering and Technology Center
Toshiba Solutions Corporation
Tokyo, Japan
kitagawa.takayuki@toshiba-sol.co.jp

Shinichi Honiden
National Institute of Informatics
Tokyo, Japan
honiden@nii.ac.jp

***Abstract*—An application of a method for prioritizing requirements to an actual project is reported. The project where one of the authors participated as a project member developed in -house software development support tools based on a software product line. In the development of a software product line, a project needs to evolve core assets in accordance with changes to the environment, the market, and technology. The concerns of stakeholders may also change the process of evolving core assets over the years, and even if stakeholders change, the concept of the target product line should be maintained. In order to effectively evolve core assets, it is important for the project to prepare and utilize a standardized method for prioritizing requirements. In this paper, we analyzed the evolution of core assets in relation to an actual project. Tacit knowledge for prioritizing requirements was extracted. Such knowledge was made explicit and defined to develop a method for prioritizing requirements. The method consists of the rules and processes for applying the rules. We also defined a meta-model for prioritizing requirements and incorporated the concept of the improvement of rules into the meta-model. According to the evaluation of the method, the following issues were clarified: (a) different stakeholders smoothly and efficiently reached agreement using the method, and (b) the method is effective for reducing lead time and costs for defining requirements.**

***Index Terms*—requirements definition, requirements prioritization, decision rules, software product line, core assets, software evolution.**

I. Introduction

Software product line development technology has been an emerging paradigm for developing a software product continuously at lower cost with higher quality in a shorter time so as to meet customer requirements or strengthen its competitive market position [1] [2]. In a software product line paradigm, an organization constructs a product roadmap, prepares software assets (core assets), and then develops a software product by reusing core assets. The market, technology, and organization related to a product line change with time, and the requirements for an existing product line—e.g., removing defects or extending core assets—accumulate. A product line development project should maintain and optimize a product line by evolving core assets.

Requirements definition is an important process whereby stakeholders of a software product discover, review, articulate, understand, and document the product requirements and its lifecycle [3] [4] [5]. A project will not always have sufficient time, resources, and budget to implement all requirements determined by the stakeholders. Therefore, requirements should be prioritized, selected, and implemented in an optimized manner.

In software product line development, projects develop core assets, and a software product is developed by reusing core assets. The core assets evolve based on development feedback. Similar to single software product development, requirements definition is also important for software product line development. Numerous studies have examined how to effectively prioritize requirements [4] [6] [7]. However, little is known about requirement prioritization for evolving core assets. The concerned stakeholders change over many years during the evolution of core assets. The basic concept of a product line as defined by the original stakeholders should be maintained, even if the stakeholders change. The requirements for the next model of the core assets should not be prioritized in a self-serving manner. Therefore, a standardized method for prioritizing requirements is required.

Some requirement prioritization methods that focus on software product line engineering have been proposed [8] [9] [10]. However, these are general methods that do not depend on specific domains. To prioritize requirements for the evolution of the core assets, concrete knowledge about what requirements are appropriate for the next model is required. In addition, determining the importance of these requirements is also necessary.

In this paper, by focusing on requirements definition for core asset evolution, we analyze an actual project to evolve core assets and extract and clarify problems for prioritizing requirements. We also devise and propose a requirements prioritization method. We aim to reduce the cost of defining requirements and reduce the time required for specific tasks.

The remainder of this paper is organized as follows. Section II analyzes an actual project relative to core asset evolution. Section III describes and proposes a meta-model driven requirements prioritization method. Section IV evaluates the proposed method, and Section V provides an assessment of effectiveness, appropriateness for stakeholders, and general applicability. Suggestions for future work are also made. Section VI reviews existing related work, and Section VII concludes the paper.

II. Project Analysis of Core Asset Evolution

The target is a project where the first author participated as a project member when she worked for Toshiba Solution Corp.; the project developed in-house software development support tools based on a software product line. The core assets for the product line consist of common and variable artifacts. The developers have developed software products by reusing core assets and delivered products to more than thirty end users. Requirements were also elicited from end users to improve the core assets and products.

A. Stakeholder Relationships

Figure 1 shows the relationships among stakeholders. The primary stakeholders related to the software product line development are the orderers, developers, and end users. Orderers correspond to business unit managers that are in charge of decision making for the investment of core asset development on the basis of a business plan. Developers are classified into two groups—core asset developers and product developers. The product developers reuse the core assets developed by the core asset developers. End users utilize the products delivered by the product developers. The end user corresponds to both individuals and organizations.

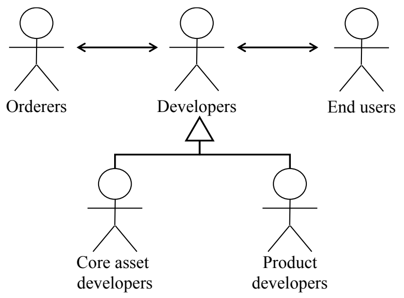


Figure 1 Stakeholders

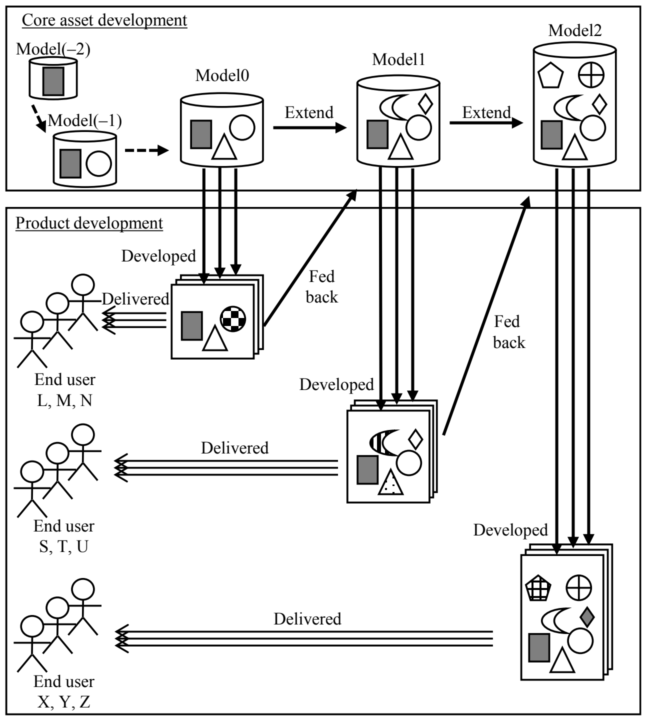


Figure 2 Relationships between core assets and products

B. Core Assets Model Relationships

The project has a product road map that describes a plan to extend core assets and develop products. Fundamentally, a new model of core assets is released according to the product road map. The developers have also considered the requirements elicited by the end users. In the case considered here, the models of the core assets have been released twice.

In the evolution of core assets, requirements definition defines what is to be used in the next model of the core assets. Some examples of such requirements are as shown below:

- Functional or nonfunctional requirements for a product road map that are planned at the beginning of core asset development.
- Functional or nonfunctional requirements for catching up with market and technology change.
- Nonfunctional requirements for improving performance or portability, which are elicited and cumulated by both core assets and product developers.
- Usability requirements elicited from end users.

Figure 2 shows the product road map for the target project. There are five core asset models in this road map. We utilized a model of the core assets to analyze problems. We refer to this model as Model0. Model1 and Model2 are the succeeding models for Model0 and Model1, respectively. As mentioned above, core assets have been extended twice before Model0. Model(−1) and Model(−2) are the precedent models of Model0 and Model(−1), respectively.

Significant architectural change was not performed in this road map. Model0 includes Model(−1) and Model(−2); Model1 and Model2 include Model0 and Model1, respectively. As is shown in Fig. 2, three products for end users L, M, and N were developed by reusing core assets from Model0. The core asset developers considered the requirements elicited from end users L, M, and N, and Model1 was developed by extending Model0. Model2 was developed in the same way as Model1.

C. Requirements Definition Process

Figure 3 shows a requirements definition process for the evolution of the core assets. The process consists of the following sub-processes: elicitation, prioritization, and review and agreement.

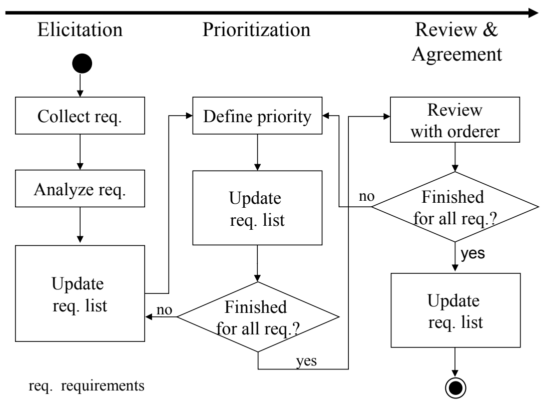


Figure 3 Requirements definition process for core assets evolution

At the beginning of the elicitation sub-process, a core asset developer collects a requirements list, which has been maintain from the launch of the target core assets. The developer analyzes individual requirements of the list. To understand the requirements precisely, this task includes an interview with the stakeholders who provided the requirements. The developer reorganizes and unifies the requirements list by avoiding duplication, or by merging similar requirements to form a single requirement. The requirements list is then updated.

In the prioritization sub-process, the core assets development organization conducts an internal meeting to discuss which requirements are appropriate in the next core asset model. The organization defines the draft priority of a requirement stored in the requirements list.

The developer's organization holds a formal review with orderers regarding the priority of the requirements based on the draft priority. These requirements are reviewed and redefined relative to priority. If the orderers consider the requirement priority inappropriate, the developer reviews and redefines the priority. The formal review is repeated until the involved stakeholders agree upon the priority.

D. Lead Time for Model0

Table I shows the number of requirements and average lead time for defining the requirements. The lead time indicates the time between elicitation and agreement for a requirement, which is expressed in days. The requirements are classified based on the types of requirements and the stakeholders. In this study, we used seven types of requirements; they are reliability, functionality, usability, portability, efficiency, maintainability, and others. The types except for others were defined based on the quality model [11].

Table I Number of requirements and time taken for definition (days) (Model0)

No.	Type	Number of req.	Average duration (days)	Number of req.			Average duration (days)		
				End user	Developer	Orderer	End user	Developer	Orderer
1	Reliability	9	145.44	3	5	1	278.00	94.00	5.00
2	Functionality	31	92.77	4	16	11	121.00	114.00	51.64
3	Usability	14	192.29	5	3	6	289.60	80.00	167.33
4	Portability	9	65.33	1	7	1	242.00	48.86	4.00
5	Efficiency	2	4.00	–	1	1	–	0.00	8.00
6	Maintainability	4	154.75	1	3	–	249.00	123.33	–
7	Others	13	54.69	1	4	8	62.00	59.00	51.63
Total		82	107.35	15	39	28	221.27	89.28	124.36

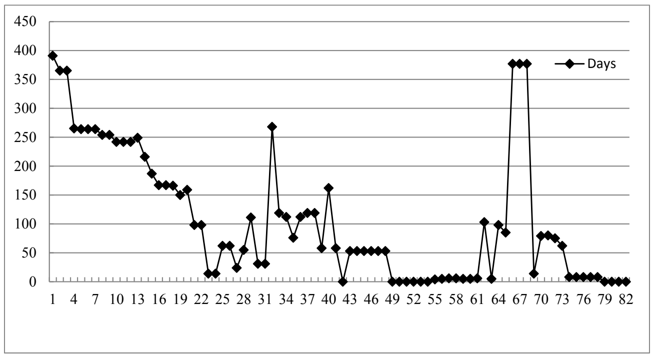


Figure 4 Length of time (days) for defining requirements (Model0)

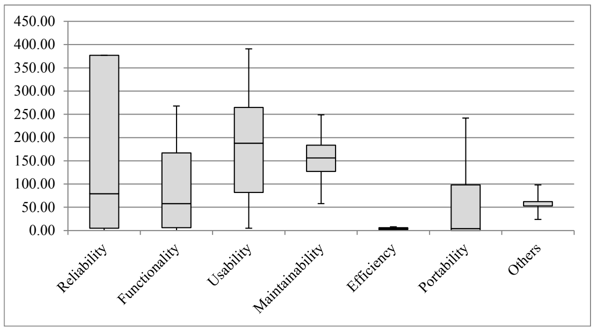


Figure 5 Length of time (days) for defining requirements according to requirements type (Model0)

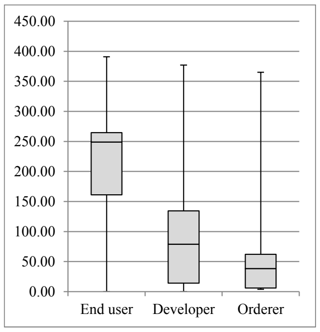


Figure 6 Length of time (days) for defining requirements according to stakeholder type (Model0)

As can be seen in Table I, 82 requirements were elicited and stored in the requirements list. It took an average of 107.35 days to complete the requirement definitions. Figure 4 shows the number of days required for the definition of a requirement. In Fig. 4, the horizontal axis shows the ID of a requirement, and the vertical axis shows time (days). It took a maximum of 391 days to define a requirement.

Figures 5 and 6 show the number of days required to define requirements according to the types of requirements and stakeholders, respectively. Both Table I and Fig. 5 show that definition of reliability, usability, and maintainability requirements took longer than others. According to Fig. 5, time required for the definition of reliability requirements varies widely. In Fig. 6, we see that the requirements elicited from end users also required more time than others.

III. Meta-model Driven Requirements Prioritization Method

A. Problem Analysis

According to the observation of Model0, we discussed with the stakeholders who defined the basic concept of the target product line by utilizing the data of Model0. The following outlines the problems extracted from the stakeholders.

- P1) It took more time to define reliability requirements, and the priority of reliability requirements was lower compared to other types of requirements. If reliability requirements are not satisfied, the risk of interrupting the business operations of the end user increases.
- P2) It is easy for a developer to add certain descriptions to a document such as a user manual. Furthermore, acquiring information about how to use a product from a user manual is effective. However, determining whether such usability requirements were treated was postponed.
- P3) Usability requirements for the very rich graphical user interface are not necessary if there have not been such requirements in the product road map. However, developers did not consider the possibility that such usability requirements were out of project scope. The developers analyzed how to implement the requirements; they analyzed the priority of the requirements. Therefore it took long time to determine that such usability requirements were out of project scope.
- P4) If a product road map has a specific function, you can determine the priority of the function easily and quickly. However, such a check was not performed. This becomes a factor that requires additional cost. If a functional requirement has been described in a product road map, its priority can be determined as high. On the other hand, if no functional requirement is present in the product road map, we can determine that the priority of the requirement is very low.

B. Decision Rules

Stakeholders change as core assets evolve. Thus, different stakeholders tend to forget the basic concept of the target product line and prioritize requirements in a self-serving manner. Therefore, a standardized method for prioritizing requirements is necessary.

We analyzed the problems discussed in III.A and the knowledge for prioritizing requirements. We defined such knowledge according to the following six decision rules. A stakeholder can avoid the situations described in III.A by adhering to the following rules.

- Rule01: A reliability requirement elicited from an end user must be satisfied. This rule was defined in response to P1)
- Rule02: A usability requirement elicited from an end user that results in the addition of a certain description to a document must be satisfied. This rule was defined by considering a solution for P2).
- Rule03: A functional requirement described in the product road map must be satisfied. This rule was defined in order to avoid P4).
- Rule04: If a requirement is a functional or usability requirement not described in the product road map, the requirement does not need to be satisfied. This rule was defined relative to P3) and P4).
- Rule05: If a functional requirement not described in the product road map meets the following conditions, it should be satisfied. This rule was defined as an exception to Rule 04.
  a) It is elicited from an end user.
  b) The end user is using a software product.
  c) The end user can progress their business operations more smoothly with the function.
- Rule06: The priority of other requirements should be individually determined.

C. Meta-model For Requirements Prioritization

The decision rules for prioritizing requirements defined in III.B are specific to the situation described in III.A. To apply a method to another project, we generalized a basic concept and defined a meta-model for prioritizing requirements. We incorporated the concept of rule improvement and extension into the meta-model. By referencing the meta-model, a project that is responsible for core asset evolution can represent uniformly and share the basic concept of the target product line.

Figure 7 shows a meta-model for prioritizing requirements. The meta-model consists of three parts: (a) a knowledge meta-model for prioritizing requirements, (b) a knowledge model for prioritizing requirements, and (c) an instance model of decision rules and processes. The features of the elements are described as follows.

(a) The knowledge meta-model for prioritizing requirements represents a basic structure of a requirements prioritization method. It shows that the method consists of decision rules and processes and that the priority of a requirement is determined from the viewpoint of stakeholder type, product road map, and requirement type.

(b) A knowledge model for prioritizing requirements is a particular example of the right part of the meta-model shown in Fig. 7(a). This indicates instances of stakeholder type, a core asset evolution plan, a requirement type and priority. The priority instances are described by the following three levels.

G1) A requirement that must be satisfied.

G2) A requirement that should be judged individually relative to scope.

G3) A requirement that does not need to be satisfied.

Priority levels G1, G2, and G3 are presented in order of priority. The priority of G2 requirements should be determined depending on the individual situation. Developers must investigate individual requirements. G3 requirements do not need to be satisfied, i.e., they are beyond the scope of the target product line.

(c) An instance model of decision rules and processes is a particular example of the left part of Fig. 7(a). This includes

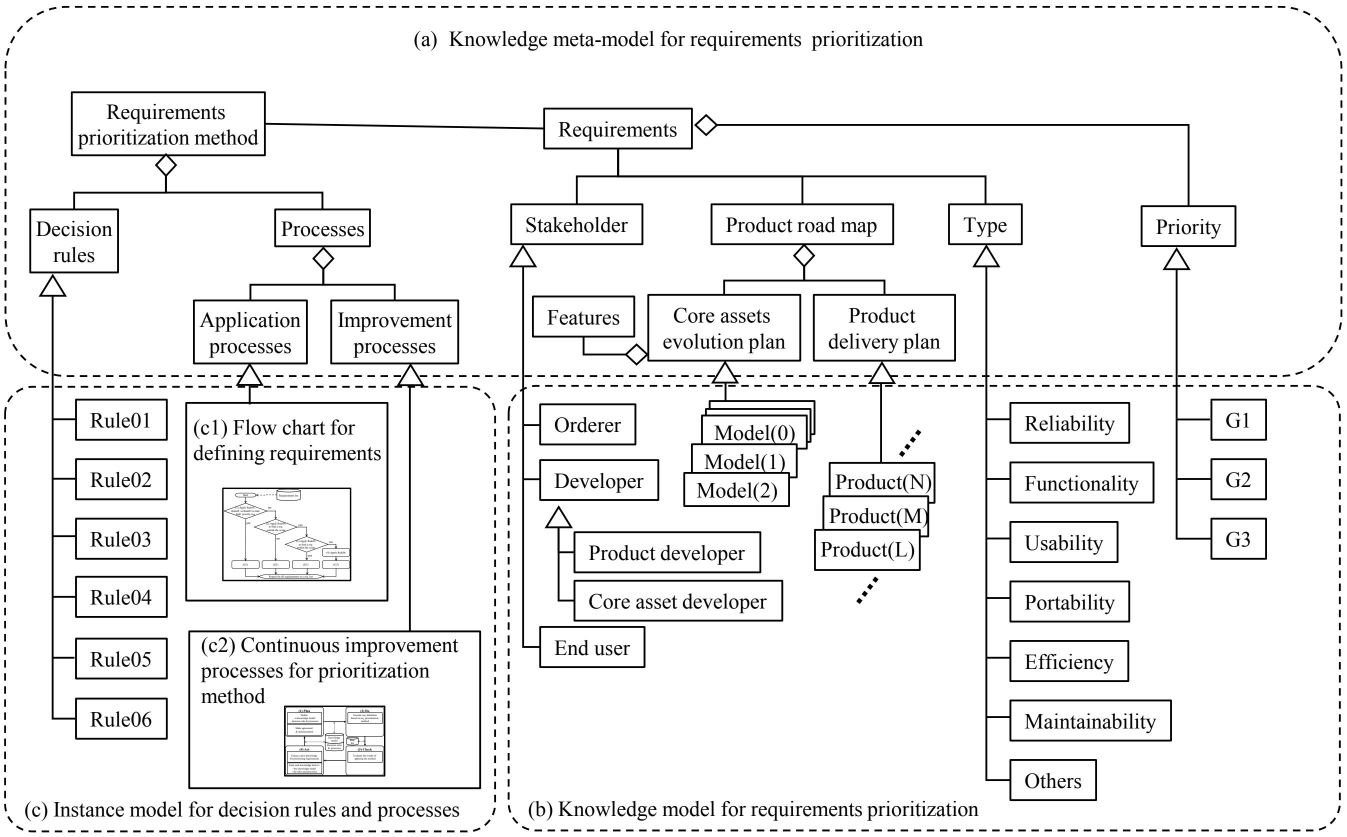
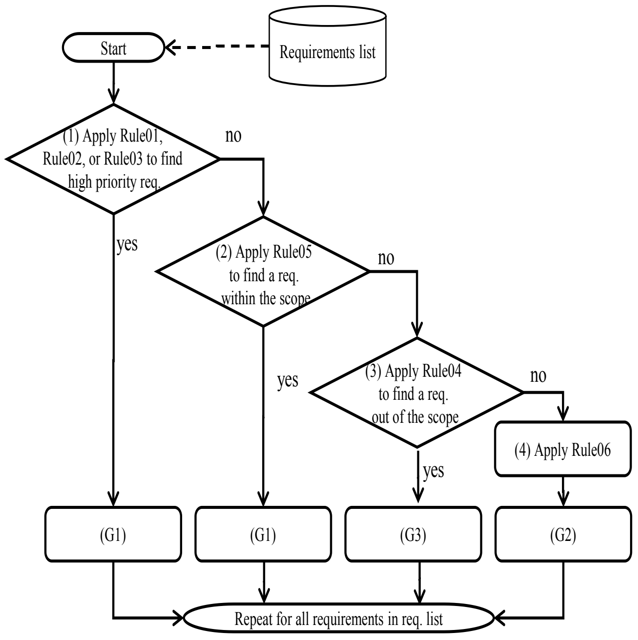


Figure 7 Meta-model for prioritizing requirements

concrete decision rules and their processes. According to the analysis of Model0, we have defined six rules. This instance model depends on domain type, stakeholders, and the concerns they face, etc. Detailed examples of application processes (Fig.7 (c1) flow chart for defining requirements) and improvement processes (Fig.7 (c2) continuous improvement processes for prioritization method) are discussed in the following III.D and III.E, respectively.

D. Application Process

Figure 8 shows a flow chart for defining requirements using the decision rules. On the basis of the decision rules, a developer can divide candidate requirements into three groups, G1, G2, and G3. As is shown in Fig. 8, first, a requirement is selected from the requirements list. It is then investigated to determine if Rule01, Rule02, or Rule03 apply. If Rule01, 02, or 03 apply, the requirement is determined to belong to G1. If these three rules do not apply, the application of Rule05 is investigated. If Rule05 applies, the requirement is classified as G1, i.e., it is within scope. If Rule01, 02, 03, and 05 do not apply to the requirement, the application of Rule04 is investigated. If Rule04 can be applied to the requirement, it is classified as G3, i.e., it is out of scope. Rule05 is applied earlier than Rule04 because Rule05 is an exception to Rule04. Utilizing Rule05 corresponds to rescuing a requirement that is essentially regarded as an unnecessary.



Start
Requirements list
(1) Apply Rule01, Rule02, or Rule03 to find high priority req.
no
yes
(2) Apply Rule05 to find a req. within the scope
no
yes
(3) Apply Rule04 to find a req. out of the scope
no
yes
(4) Apply Rule06
(G1)
(G1)
(G3)
(G2)
Repeat for all requirements in req. list

Figure 8 Flow chart for defining requirements

If any of Rule01−Rule05 is not applicable to the requirement, then Rule06 is applied. The advantage of using decision rules is that a developer can quickly determine if a requirement is necessary or unnecessary.

E. Improvement Process

Here, we define the processes for improving the requirements prioritization method. The processes consist of an iteration of the plan-do-check-act (PDCA) cycle (Figure 9). The PDCA cycle is a management tool that asserts that every managerial action can be improved by careful application of the PDCA sequence. An important contribution to continuous process improvement is the kaizen. Kaizen is a movement that leads to continual and incremental improvement [12]. Through this cycle, an organization improves a knowledge model, decision rules, and processes. The multiplier effects of the kaizen help an organization maintain optimal methods at all times. The functions of PDCA are described as follows.

- The function of "plan" is to carry out tasks; (a) to define the knowledge model, decision rules, and processes, and (b) to come to an agreement and inform all stakeholders.
- The function of "do" is to execute the kaizen, i.e., elicit requirements using the requirements list, prioritize and review them, and establish agreement with stakeholders with respect to which requirements must be in scope.
- The function of "check" is to evaluate the results of applying the method.
- The function of "act" is to extract new knowledge for prioritizing requirements and to feed this new knowledge back to the knowledge model, rules, and processes.

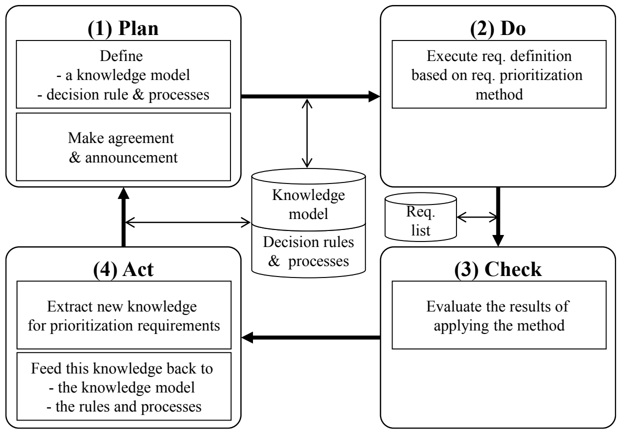

(1) Plan
Define
- a knowledge model
- decision rule & processes
Make agreement
& announcement
(2) Do
Execute req. definition based on req. prioritization method
Knowledge model
Decision rules & processes
Req. list
(4) Act
Extract new knowledge for prioritization requirements
Feed this knowledge back to
- the knowledge model
- the rules and processes
(3) Check
Evaluate the results of applying the method

Figure 9 Continuous improvement processes for prioritization method

IV. Evaluation

A. Evaluation Method

We evaluated the proposed method using two models, Model1 and Model2, which were discussed in Section II. The requirements definition for Model1 was carried out following the processes shown in Fig. 3. By taking Rule01 to Rule06 as the initial rules, each requirement was analyzed to determine if these rules are applicable; thus, the priority of the requirement was defined. On the basis of the improvement processes for this method, improvement of the decision rules was considered. Then, the requirements definition for Model2 was determined.

We evaluated the effectiveness of the six decision rules and their application processes in the evaluation of Model1. The core asset developers of Model2 were different from those of Model1. We evaluated the effectiveness of the Model2 rules and application processes under the condition that different stakeholders utilized them. In addition, the effectiveness of the improvement processes was evaluated.

Those two models and the current model, Model0, were developed in different fiscal years. However, the duration for requirements definition for all three models was six months.

B. Applied Rules

Table II shows the number of decision rules applied to Model1 and Model2. The number of candidate requirements for Model1 and Model2 were 20 and 27, respectively. These candidate requirements were classified into the G1, G2, or G3 requirements shown in Table II. The requirements classified as G2 by application of Rule06 were analyzed. These requirements were classified as G1 or G3 requirements. Since the granularity of some requirements was small, it was difficult to judge whether the product road map included them.

Although Rule05 was not applied to any requirements for Model1, we decided not to modify the six decision rules. We used the decision rules as they were for Model2. According to Table II, all rules were utilized, although the utilization ratio of Rule06 was the highest. It should be acknowledged that additional rules could be added.

Table II Rules applied to Model1 and Model2

	Rule01	Rule02	Rule03	Rule04	Rule05	Rule06		Total
						G2		
	G1	G1	G1	G3	G1	G1	G3	
Model1	0	9	2	1	0	7	1	20
Model2	7	4	5	0	2	6	3	27
Total	7	13	7	1	2	13	4	47
Ratio	14.89%	27.66%	14.89%	2.13%	4.26%	27.66%	8.51%	–

C. Duration for Requirements Definition

1) Duration for the requirements definition of Model1 and Model2: The results of applying the requirements prioritization method to Model1 are shown in Table III, Figure 10, Figure 11, and Figure 12. The results for Model2 are shown in Table IV, Figure13, Figure 14, and Figure 15. As can be seen, an average of 1.15 and 15.19 days was required to define the requirements for Model1 and Model2, respectively.

2) Solution Situation for the Long Duration Problem: According to the information shown in Table III and Table IV, an average of 1.5 days or less was required to define the reliability requirements of Model1 and Model2.

Table III Number of requirements and time taken for definition (days) (Moldel1)

No.	Type	Number of req.	Average duration (days)	Number of req.			Average duration (days)		
				End user	Developer	Orderer	End user	Developer	Orderer
1	Reliability	2	0.50	0	1	1	–	0.00	1.00
2	Functionality	4	1.25	2	1	1	0.50	3.00	1.00
3	Usability	10	1.40	9	0	1	1.44	–	1.00
4	Portability	1	1.00	0	0	1	–	–	1.00
5	Efficiency	0	–	0	0	0	–	–	–
6	Maintainability	1	0.00	0	1	0	–	0.00	–
7	Others	2	1.00	0	0	2	–	–	1.00
Total		20	1.15	11	3	6	1.27	1.00	1.00

Table IV Number of requirements and time taken for definition (days) (Model2)

No.	Type	Number of req.	Average duration (days)	Number of req.			Average duration (days)		
				End user	Developer	Orderer	End user	Developer	Orderer
1	Reliability	8	1.50	7	0	1	1.71	–	0.00
2	Functionality	8	6.50	8	0	0	6.50	–	–
3	Usability	8	19.38	6	1	1	25.83	0.00	0.00
4	Portability	1	0.00	1	0	0	0.00	–	–
5	Efficiency	0	–	0	0	0	–	–	–
6	Maintainability	0	–	0	0	0	–	–	–
7	Others	2	191.00	1	1	0	0.00	191.00	–
Total		27	15.19	23	2	2	9.52	95.50	0.00

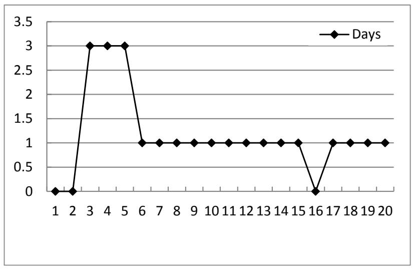


Figure 10 Length of time (days) for defining requirements (Model1)

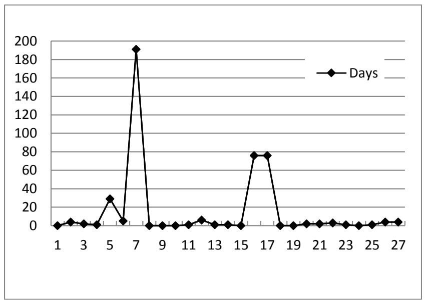


Figure 13 Length of time (days) for defining requirements (Model2)

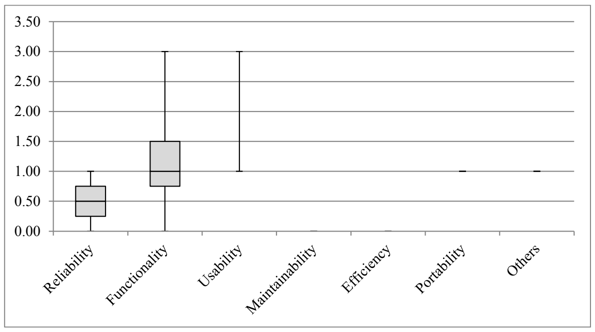


Figure 11 Length of time (days) for defining requirements according to requirement type (Model1)

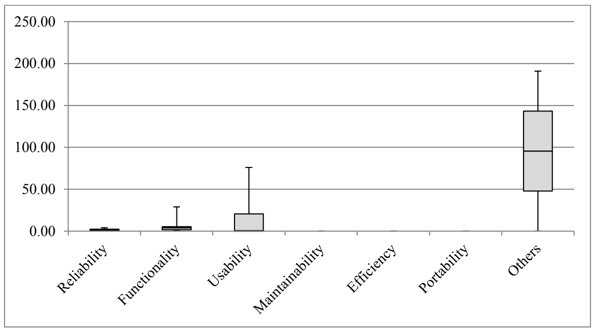


Figure 14 Length of time (days) for defining requirements according to requirements type (Model2)

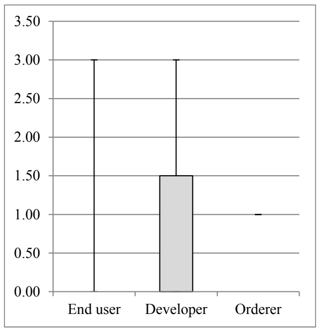


Figure 12 Length of time (days) for defining requirements according to stakeholder type (Model1)

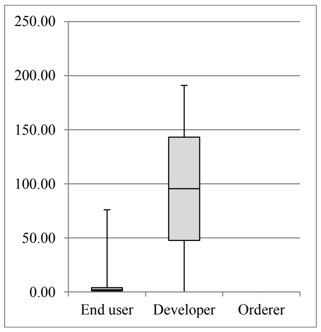


Figure 15 Length of time (days) for defining requirements according to stakeholder type (Model2)

Seven requirements were determined by focusing on the reliability requirements of Model2, which were elicited from end users. It took an average of 1.71 days to define these requirements. Compared to Model0, the duration of the reliability requirements definition for Model1 and Model2 appears to be improved.

The definition of usability requirements for Model1 and Model2 required 1.4 days and 19.38 on average, respectively. Model2 required more time than Model1 due to usability requirements that were difficult for developers to understand. The developers needed to interview end users regarding these requirements. Compared with the duration of the usability requirements for Model0, both Model1 and Model2 appeared to improve.

Regarding the definition of functional requirements, Model1 and Model2 required 1.25 days and 6.5 days on average, respectively. The long duration problem of the functional requirements definition was improved compared to the 92.77 days required for Model0.

Four requirements were classified into others in Table III and Table IV. They were the requirements for the project management. For example, they were about the improvement of staff assignment and resource distribution. There was one requirement that required 191 days to define for Model2. This requirement was related to an in-house project management standard. The long definition period originated in project management and was unrelated to the requirement prioritization method.

D. Cost Analysis

Table V shows the costs to define requirements for Model0, Model1, and Model2. Cost was measured by taking a requirement from each requirement list for Model0, Model1, and Model2. The value of the cost is relative and does not have a specific unit. As is shown in Table V, the costs of requirements definition for Model1 and Model2 were reduced compared to Model0. The period for defining requirements for Model1 and Model2 were shorter than that of Model0 because it was not necessary to repeat requirement understanding, prioritizing, and reviewing activities. Therefore, the costs of the requirements definition for Model1 and Model2 were reduced.

Twenty requirements were determined for Model1. Since the developers could concentrate on the requirements definition in a short time, the definition of these twenty requirements was performed simultaneously. Table VI shows the costs to define the twenty requirements. The twenty requirements definition costs for Model0 and Model2 were calculated by multiplying 20 by an individual cost for the requirements shown in Table V. According to Table VI, the costs for Model1 and Model2 were reduced to 20% and 63% of Model0, respectively.

Table V Cost for defining one requirement

	Model0	Model1	Model2
Elicitation	1.25	0.5	0.7
Prioritization	1	0.42	1
Review & Agreement	0.75	0.42	0.75
Total	3	1.34	2.45

Table VI Cost for defining 20 requirements

	Model0	Model1	Model2
Elicitation	25	1	11
Prioritization	20	2	16
Review &Agreement	15	9	11
Total	60	12	38

E. Improvement of Rules

There is a portability requirement when Rule06 is applied for both Models 1 and 2. A portability requirement for the core assets evolution indicates that a software product developed on the basis of a core assets model can work for a new core assets model. There are several solutions to address the implementation of a portability requirement. One solution is to prepare an automatic transformation mechanism. Another is to provide a document that describes the procedure for converting the product to a new environment. End users must be able to utilize a new core assets model with one of the solutions. The portability requirements must be immediately determined as within the scope of the next model.

In the evaluation, we did not prepare official decision rules related to portability requirements. However, the core assets developers tacitly placed high priority on portability requirements; thus, a significant period for defining requirements for portability requirement was prevented.

V. Discussion

A. Effectiveness

As shown in Section IV, the times required for requirements definition for both Model1 and Model2 were shorter than that for Model0. In addition, the costs of the requirements definition were decreased. Therefore, the method is considered to be effective.

Regarding the reliability requirements elicited from end users, there were seven requirements among Model1 and Model2. These requirements were defined as G1 requirements smoothly; the basic concept of the target product line was maintained by different stakeholders.

Although Model2 required more time than Model1, compared with the duration of the usability requirements for Model0, both Model1 and Model2 appeared to improve. There were usability requirements that were difficult for developers to understand. These usability requirements were elicited on the supposition of the utilization of products under end user specific environments. The developers needed to interview end users regarding these requirements in order to determine if such requirements were necessary or unnecessary for core assets. The decision rules gave stakeholders opportunities for analyzing usability requirements in detail and helped prevent the postponement of the requirements definition.

In addition, the costs of the requirements definition were decreased in proportion to shortening of the time required for requirements definition. In case of Model0, it cost quite a lot to determine priority of requirements that had been stored in a requirements list for a long time. In order to reduce the requirements definition cost, it is helpful to shorten the retention period of requirements stored in the requirements list.

Moreover, the PDCA improvement processes mentioned in III.E enabled us to find new knowledge for the portability requirements that can be represented as a new rule: specifically, a portability requirement that describes that a software product developed on the basis of a model of core assets must work on a new model of core assets. These PDCA improvement processes are imperative for promoting continuous feedback on the knowledge.

B. Appropriateness

The six decision rules for prioritizing requirements are specific to in-house software development support tools based on a software product line; the rules were defined to solve the problems described in III.A. The major cause of the problems was that from the requirements definition for Model0, different stakeholders ignored the basic concept and gave priority to requirements in an ad hoc manner. These problems were solved by utilizing the requirements prioritization method for Model1 and Model2; the basic concept was maintained.

The knowledge of basic concept of the target product line was represented as decision rules; the form of a rule was easy to understand for stakeholders; the method was easy to utilize. The requirements definition was uniformly performed by different stakeholders.

C. Applicability

The six decision rules for prioritizing requirements were devised to maintain the basic concept of existing core assets. The rules take precedence over the reliability and usability issues elicited from end users. If we begin to develop core assets for a new business domain, the current six decision rules would not be appropriate because they do not consider the basic concept for new core assets. Therefore, to apply the method to development of new core assets, new decision rules are necessary. We can create a knowledge model to prioritize requirements models, and instantiate and devise decision rules and processes on the basis of the meta-model. By utilizing the decision rules devised specifically for target core assets, we will be able to evolve core assets smoothly.

D. Future task

Table II shows that approximately 36% of Model1 and Model2 requirements utilized Rule06. It was observed that the utilization ratio of Rule06 was the highest. Among the requirements classified as G2 by the application of Rule06, the granularity of some requirements was too small to judge quickly whether the product road map included such requirements. As previously mentioned, there is some to append additional beyond Rule01 to Rule05.

If numerous decision rules are defined, the application processes of decision rules will be complicated. Furthermore, it will be difficult to utilize our method. Decision rules represent the basic concept of a target product line; they should be simple and easy to understand. It is necessary to optimize and elaborate decision rules continuously.

VI. Related Work

Basic requirement elicitation techniques include use cases [13] [14], scenarios [15], and goal models [16] [17]. These techniques do not directly contribute to requirements prioritization; however, they do contribute to the solution of the problem mentioned in Section V.A. Specifically, the problem is that the granularity of a requirement may be too small to judge quickly whether the requirement was included in the product road map. If requirements specifications are uniformly represented by utilizing use cases, scenarios, and goal models, it will be easy to understand the specifications and the requirement's relationship with a product road map. Accordingly, it can be quickly determined where a new requirement should be positioned in the product road map.

Numerous studies have been conducted on the prioritization of requirements [4] [7]. Current requirements prioritization techniques include the following:

(a) Evaluation by assigning numeric value to the importance of requirements [7] [18].

(b) Evaluation by utilizing two metrics: cost and value [6] [19].

(c) Evaluation on the basis of several metrics, considering the importance and effectiveness of requirements [20].

The techniques (a) include 1-10 ranking [4], MosCow [4], and 100-point test [18]. These are simple and do not require special learning costs and tools; any project will be able to use them easily. For techniques (b) and (c), a requirement is evaluated by several metrics, and the evaluation result is visualized on a coordinate plane. Accordingly, multiple stakeholders can reach a common understanding intuitively.

As has been mentioned previously, concerned stakeholders can change during the evolution of core assets. Different stakeholders can define different priorities over time. It is difficult for different stakeholders to define the priority of requirements uniformly over time by only utilizing existing techniques.

Davis applied "triage," most commonly used to determine medical treatment priorities, to requirements prioritization. This method is referred to as requirements triage [21] [22]. Requirement triage helps developers classify the requirements quickly into unnecessary and necessary requirements by utilizing a vote mechanism. This method corresponds to the process of defining priority; it does not provide concrete knowledge regarding which requirements should be taken as important or their value.

Kukreja et al. presented a method for selection of the most appropriate technique by combining seventeen different prioritization techniques [23] and defined seventeen evaluation criteria, including consideration of ease of realization, ease of use, linkability to end benefits/goals, scalability. Their method was developed by evaluating existing prioritization techniques. This study proposes a meta-level method for selecting a requirement prioritization technique. Defining the priority of a requirement on the basis of existing prioritization techniques is more effective than defining priority in an ad hoc manner. Appropriate techniques depend on the conditions of an organization, stakeholder skills, etc. However, the selection of only a single technique is not necessary; combinations of two or more techniques should be also considered. For example, if we combine a cost-value approach with the proposed prioritization method when a project launches a product line, the evaluation results for requirements will be represented as a cost-value diagram. The cost-value diagram shows a basic concept of the target product line. Such diagrams help stakeholders understand and share the basic concept of the target product line.

Prioritization methods that focus on software product line engineering have also been proposed [8] [9] [10]. These are comprehensive methods and tools that examine how to elicit, analyze, specify, and verify requirements that are specific to a software product. However, they are generalized and do not depend on specific domains. These techniques are applicable to the launch of a new product line; any concrete know-how that focuses on core asset evolution in a concrete domain is not shown. As discussed in this paper, analyzing a concrete example of a core asset evolution project provides additional meaning that clarifies knowledge about requirement priority. Thus, we have proposed a meta-model and processes for generalizing and improving such knowledge.

VII. Conclution

In this paper, an application of a method for prioritizing requirements to an actual project is reported. The method has been proposed for a core asset evolution project. The objective of the method is to help different stakeholders define the priority of a requirement uniformly. The method consists of rules for prioritizing requirements and processes for applying these rules. In addition, we have defined a meta-model for prioritizing requirements and processes for continuous improvement of rules. According to an evaluation of the method, it was clarified that different stakeholders were able to smoothly and efficiently reach an agreement, and the development lead time was shorter. In addition, the cost required to define requirements was reduced. Our requirements prioritization method helps a core asset evolution project continuously acquire and share concrete knowledge about what requirements are appropriate for the next model. This incremental PDCA cycle plays the role of an engine for an organization's growth.

As a future work, we will increase the number of decision rules and business domains continuously; we will contribute to a business growth.

References

[1] Clements, P. and Northrop L., "Software product lines: practice and patterns", Addison-Wesley, 2001.

[2] McGregor, J., Muthig, D., Yoshimura K., and Jensen, P., "Guest editor's introduction: successful software product line practices", IEEE Software, May/June 2010, pp.16 – 21, 2010.

[3] IEEE std. 830-1998, "IEEE recommended practice for software requirements specifications", IEEE, 1998.

[4] International Institute of Business Analysis, "A guide to the business analysis body of knowledge® (BABOK® Guide)", Version 2.0, 2009.

[5] ISO/IEC/IEEE, "ISO/IEC/IEEE 29148, systems and software engineering, life cycle processes, requirements engineering", 2011.

[6] Karlsson, J., Wohlin, C., and Regnell, B., "An evaluation of methods for prioritizing software requirements", Information & Software Technology, 39, pp.939 - 947, 1998.

[7] Mead, N. R., "Requirements prioritization introduction", Carnegie Mellon University, https://buildsecurityin.us-cert.gov /articles/best-practices/requirements-engineering/requirements-prioritization-introduction, 2006 - 2013.

[8] Lee, J, Kang, KC, Sawyer, P & Lee, H., "A holistic approach to feature modeling for product line requirements engineering", Requirements Engineering, 28 September, 2013 Springer, 2013.

[9] Derakhshanmanesh, M., Fox, J., and Ebert, J., "Requirements-driven incremental adoption of variability management techniques and tools: an industrial experience report", Requirements Engineering, 29, September 2013, Springer, 2013.

[10] Alferez, M., Bonifacio, R., Teixeira, L., Accioly, P., Kulesza, U., Moreira, A., Araujo, J., and Borba, P., "Evaluating scenario-based SPL requirements approaches: the case for modularity, stability and expressiveness", Requirements Engineering, 23 October 2013, Springer, 2013.

[11] ISO/IEC, "ISO/IEC 9126-1:2001, software engineering - product quality - part 1: quality model", 2001.

[12] Imai, M., "Kaizen - The key to Japan's competitive success", McGraw-Hill, 1986.

[13] Cockburn, A., "Writing effective use cases", Addison-Wesley Professional, 2000.

[14] Jacobson, I., "Object-oriented software engineering", ACM Press, 1991.

[15] Carroll, J. M., "Making use: scenario-based design of human-computer interactions", The M.I.T. Press, 2003.

[16] Van Lamsweerde, A., "Requirements engineering: from system goals to UML models to software specifications", Wiley, 2009.

[17] Yu, E. S. K., "Towards modelling and reasoning support for early-phase requirements engineering", Proc. of the Third IEEE International Symposium on Requirements Engineering, pp.226 - 235, 1997.

[18] Leffingwell, D. and Widrig, D., "Managing software requirements: a use case approach", 2nd ed. Boston, MA: Addison-Wesley, 2003.

[19] Karlsson, J. and Ryan, K., "A cost-value approach for prioritizing requirements", IEEE Software, Vol.14, No.5, pp.67 - 74, 1997.

[20] Wiegers, K. E., "Software requirements", 2nd ed. Redmond, Wash.: Microsoft Press, 2003.

[21] Davis, A. M., "The art of requirements triage", IEEE Computer, Vol. 36. No.3, pp.42 – 49, 2003.

[22] Davis, A. M., "Just enough requirements management: where software development meets marketing", Dorset House, 2005.

[23] Kukreja, N., Payyavula, S. S., Boehm, B., and Padmanabhuni, S., "Selecting an appropriate framework for value-based requirements prioritization: a case study", 20th IEEE International Requirements Engineering Conference, RE 2012, pp.303 - 308, 2012.