Modeling Software Characteristics and Their Correlations in A Specific Domain by Comparing Existing Similar Systems

Akira Osada Daigo Ozawa Haruhiko Kaiya Kenji Kaijiri
Graduate School of Science and Technology, Shinshu University
4-17-1 Wakasato, Nagano 380-8553, Japan
csada@cs.shinshu-u.ac.jp t05a516@amail.shinshu-u.ac.jp
kaiya@cs.shinshu-u.ac.jp kaijiri@cs.shinshu-u.ac.jp

Abstract

Software in a specific domain has several characteristics and each characteristic should be fixed when the software requirements are specified. In addition, these characteristics sometimes correlate with each other. However, we sometimes forget to specify several characteristics and/or to take their correlations into account during requirements elicitation. In this paper, we propose a meta-model for representing such characteristics and their correlations, and also propose a method to build a model for a specific domain by using documents about existing software systems. By using our model for a domain, a requirements specification for a system in the domain could be complete and unambiguous because requirements analysts can check the characteristics that should be decided. The specification could be also correct and consistent because the analysts can know side effects of a requirement change by using correlation among the characteristics. We have applied our methods to a case study for confirming the usefulness of such model and the methods.

Keywords: Requirements Engineering, Requirements elicitation, Domain Modeling.

1. Introduction

We can derive knowledge of a specific domain from resources of existing systems belonging to the domain, and the knowledge can be used to elicit and define requirements of a new system in the domain. For example, when a requirements analyst elicits requirements about a user identification system for a specific library, the analyst will refer similar systems and/or their documents implicitly or explicitly. In this paper, we will propose a model how to explicitly manage such knowledge, a method how to use such knowledge and a method how to build the model. The expected results of the model and the methods are as follows.

- We can avoid forgetting to specify software characteristics that should be decided in a specific domain.
- We can avoid specifying several characteristics mutually contradicting each other.
- We can avoid incorrect realization of such characteristics.

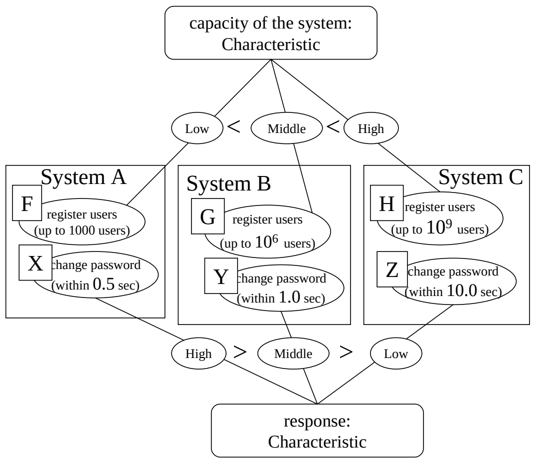

capacity of the system: Characteristic
Low
<
Middle
<
High
System A
F
register users (up to 1000 users)
X
change password (within 0.5 sec)
System B
G
register users (up to 10^6 users)
Y
change password (within 1.0 sec)
System C
H
register users (up to 10^9 users)
Z
change password (within 10.0 sec)
High
>
Middle
>
Low
response: Characteristic

Figure 1. Example of Comparison and Ordering between Similar Systems

The main ideas of our model and methods are comparison and ordering between similar functions and attributes that belong to different existing systems. For example in Figure 1, suppose we have three similar but different systems for user identification, and each system has two kinds of function; "register users" and "change password". By comparing three functions labeled by "F", "G" and "H" in the Figure, we can know "the maximal number of users" is one of the characteristics that should be specified in the requirements. However, we sometimes miss such characteristics by referring only one function, e.g., only "F". In addition, we can put these three functions in order with respect to the number, and we can assume this ordering specifies a preference or a criterion about this kind of function. In this case, "larger number of users is preferred" is a criterion.

The comparison is also useful to identify mandatory and/or optional characteristics. For example, suppose a characteristic about "change password". If all functions about the characteristic work within 0.5 seconds, this attribute value (i.e., within 0.5 seconds) will be mandatory requirement. On the other hand, if functions have different values as shown in Figure 1, it will not be mandatory requirement about this characteristic, and even we may assume this characteristic is unstable.

We also try to detect incorrectness and inconsistency among characteristics by using our model and methods. When customers request to change some characteristics, we should reconsider related characteristics that are not explicitly mentioned. If we do not reconsider such related characteristics, to-be-system would not meet needs of the customers. Thus, we have to know which characteristics are mutually related with each other. Correlations are systematically calculated by using the ordering among functions or attributes, and we assume missing effects could be found by such correlations. Figure 1 shows a simple example of a correlation. For a function "register users", instances of the function are put in order as "H", "G" and "F", with respect to a criteria that "large number is preferred". On the other hand, instances of "change password" are put in order as "X", "Y" and "Z", with respect to a criteria that "quick response is preferred". The orderings about the two functions are reverse, thus we may assume two characteristics corresponding to the functions will negatively correlate with each other. When this correlation is suggested under the change request about "register users", we can confirm whether characteristics about "change password" should be reconsidered or not. If we do not reconsider it, the characteristics about "change password" could be decided against the needs of customers, e.g., the response capability becomes bad. As a result, we cannot avoid incorrectness related to "change password".

The rest of this paper is as follows. In section 2, we explain how to represent knowledge about existing systems as shown in Figure 1, i.e., a meta-model for such knowledge. In section 3, we explain the usage of a model for a specific domain and advantages of the model. In section 4, we present how to develop a model for a specific domain. In section 5, we show a case study for confirming the usefulness of our model and methods. Finally, we discuss related works and summarize current results and future works.

2. Model of Software Characteristics and Their Correlation

In this section, we first show the meta-model, i.e., the data structure of a model for knowledge in a specific domain. Then, we show an example of this model to show that we can represent characteristics and their correlations by using the models.

2.1 Meta-Model

Figure 2 is the meta-model for a knowledge representation in a specific domain, written in a class diagram of UML. We explain the meta-model as follows.

- For an application domain, there should be more than two instances of existing systems.

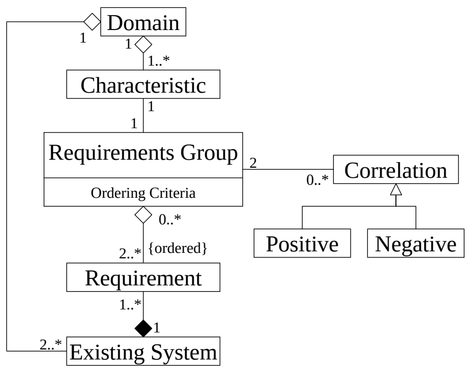


Figure 2. Meta-Model

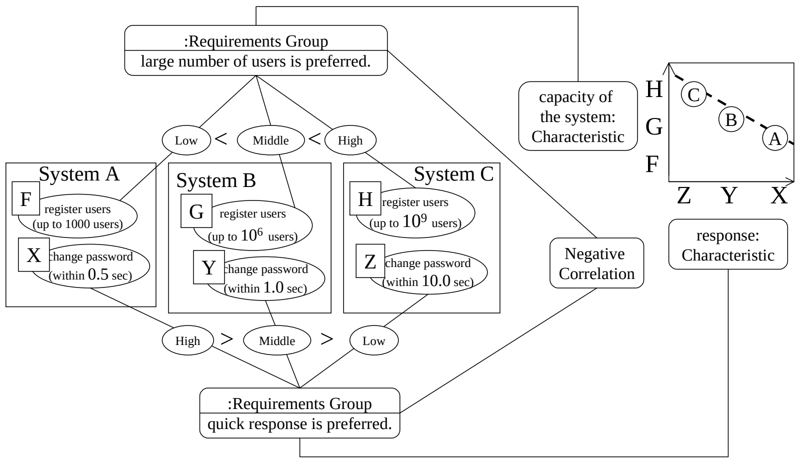
- Each system consists of several requirement statements, e.g., written as sentences, or written as use cases.
- A requirement group has an ordered set of requirements, and the group also has a criterion how to put the requirements in order. The cardinal number of the set should be more than two.
- A requirement group corresponds to a characteristic. This means that the characteristic is specified by the ordered set of concrete requirements.
- A domain has several characteristics that should be specified or taken into account.
- Two requirements groups sometimes correlate with each other negatively or positively.

By using this meta-model, we can easily access the concrete descriptions of requirements when we consider characteristics that should be specified. In addition, we can understand the characteristics well by comparing the requirements. When we describe a model for a specific domain, we can name an instance of "characteristic" but we do not have to do so. This means we do not have to abstract characteristics by giving names like usability, reliability and so on.

Currently, we do not take care about intervals and ratios among requirements but simply put the requirements into order. For example, when we compare requirements, we never mention that one requirement is twice better than another. The main reason is that it is difficult to introduce intervals and ratios for comparing requirements in general. On the other hand, we have several techniques to put requirements in order as mentioned in section 4.2.

2.2 Example of a Model

We show an example of a model based on the meta-model above. In Figure 3, we show a typical example of a model and also show how to express a correlation. For convenience, we use use-case diagrams for representing existing systems and their requirements. We also annotate a two-dimensional chart as a supplementary explanation for the correlation. Figure 3 is an extended version of Figure 1, thus the application domain of Figure 3 is the same as the domain in Figure 1. We added two instances of "requirements group" and an instance of "negative correlation". Be-



:Requirements Group
large number of users is preferred.
Low
Middle
High
System A
F
register users (up to 1000 users)
X
change password (within 0.5 sec)
System B
G
register users (up to 10^6 users)
Y
change password (within 1.0 sec)
System C
H
register users (up to 10^9 users)
Z
change password (within 10.0 sec)
capacity of the system: Characteristic
H
G
F
C
B
A
Z Y X
Negative Correlation
response: Characteristic
High
Middle
Low
:Requirements Group
quick response is preferred.

Figure 3. Typical Example (Extended version in Figure 1)

cause each instance of "requirements group" has an ordering criterion like "large number of users is preferred", we can easily understand what is regarded as important or preferred about the group. Because each instance of "requirements group" also corresponds to an instance of "characteristic", we can well understand each characteristic. An instance of "correlation" is explicitly represented in Figure 3, and its intuitive meaning is a two-dimensional chart in the Figure. In this case, two requirements groups are mutually correlated, thus we should reconsider a characteristic about "change password" when the characteristic about "register users" is changed

We explain how to decide whether two requirements groups correlate or not by using the example in Figure 3. In the example, requirements F, G and H are put in order as $H > G > F$ with respect to a criterion "large number of users are preferred". Thus, we line up the requirements on the vertical axis in the chart according to the ordering. We also line up X, Y and Z in the same way on the horizontal axis. Because F and X are parts of System A, an intersection point of F and X on the chart denotes System A. Other intersection points also denote the corresponding Systems. As we can see, the two ordered sets $\{F, G, H\}$ and $\{X, Y, Z\}$ appear to be closely related. Thus we assume two requirements groups corresponding to the sets correlate with each other. Note that we do not calculate and test correlation coefficient strictly because plots here are based not on interval/ratio scale but on ordinal scale.

3. Usage of a Model

We explain how to use a model for a specific domain.

3.1 Elicitation in General

In Figure 4, we depict how to elicit requirements for a new system with the help of the model. Because the model includes the list (or the set) of characteristics that should be specified or taken into account in the application domain,

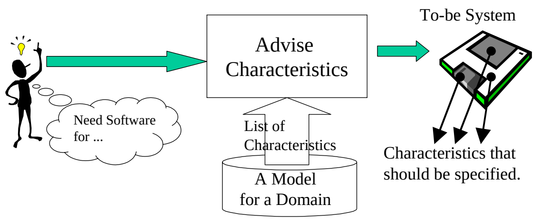


Figure 4. Requirements Elicitation by using Characteristics

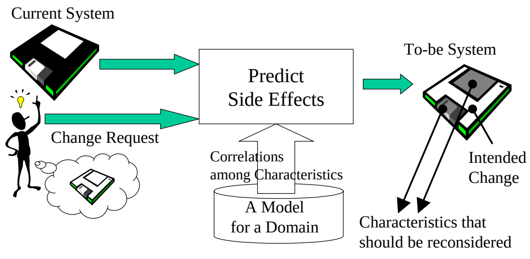


Figure 5. Predicting Side Effects by using Correlations

at least requirements for such characteristics should be discussed during requirements elicitation. When stakeholders such as customers and/or users have not clearly understood their own needs and/or preferences yet, the criterion in each requirements group will help them to understand and decide them. In addition, stakeholders can know each characteristic is stable or unstable, mandatory or optional because they can know the current tendency for each characteristic to be decided by referring a set of requirements related to each characteristic.

3.2 Handling Change Requests

In Figure 5, we depict how to handle change requests from stakeholders by using a metaphor. In the Figure, the body color of the diskette is requested to change black to white, and the request should be satisfied. In addition, we should explore side effects caused by the change request.

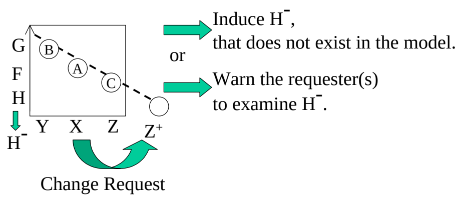


Figure 6. Induction by a Correlation

In this Figure, two side effects are depicted; the color of a label and the color of a shutter.

If such side effects are not taken into account, two kinds of problems could occur. One is about incorrectness. For some reasons, e.g., logical, physical, political, legal and/or organizational constraints, one change requires another changes. For example in Figure 1, if the number of users is requested to change 1000 to 10^9, the response capability for changing passwords will largely decline. If analysts and stakeholders overlook this point, the to-be-system could have bad response capability and such characteristic will not meet the needs for stakeholders. We can avoid such bad capability by several ways, for example, funding more money or introducing excellent technologies. However, in the requirements analysis step, we have to recognize such situation and have to decide how to handle it. Our model and method will contribute to such recognition. Another is about inconsistency. If two contradict or incompatible requirements are requested, the to-be-system cannot be realized or it takes a lot of costs and/or efforts to realize the system. Thus, one of such requirements should be given up or we have to explore some compromise or backup plan in the requirements analysis phase.

The correlations among characteristics will help us to find problems about incorrectness and inconsistency, because the correlations are calculated from the existing systems in a domain and a to-be-system will have similar characteristics.

3.3 Handling Changes by Induction

We finally introduce induction about characteristics. In general, a requirement about a characteristic does not directly correspond to an existing requirement in a model. For example in Figure 1, the number of users could be requested as 3000, 30 or 10^{30}, and there is no such requirements in the model directly corresponding the request. Thus, we cannot predict side effects caused by such request now.

For overcoming such cases, we introduce induction about characteristics. Figure 6 depicts an example of induction. As shown in the Figure, we can propose two kinds of results, inducing side effects or warning the requester to examine the effects.

Suppose a change request, say Z^+, does not correspond to existing requirements in the model as shown in the Figure 6. Then we induce expected requirements, say H^- along with the procedure of linear regression as shown in Figure 6. If H^- is an impossible requirement without any special considerations, we have to warn the requester that Z^+ could cause impossible effect(s). When the requester has received such warnings, he/she and analysts have to explore alternatives or extra, e.g., additional machines, to resolve the problem. If H^- can be possible, we also report that induced side effect is H^- and make stakeholders to accept the effect or not.

Because we do not use interval scale for ordering, we are not allowed to use linear regression procedure. However, we use the procedure because our goal is to explore the possibility and tendency about side effects and we do not have to exactly induce the effects.

4. Method to Build a Model

Whether we can elicit requirements effectively or not largely depends on the availability of a model. For example about such availability, new knowledge should be easily added in an existing model, i.e., the model should be scalable with respect to the number of existing systems. Here we explain how to develop a model and show that a model can be developed easily and the model is scalable.

4.1 The Procedure

Basically, a model is developed along with the following steps.

1. Collect resources about each system belonging to a domain. The examples of resources are help files, manuals and the system itself.
2. Identify requirements such as functions or attributes of each system. Writing use-case diagrams is useful for this step. We may use requirements that are already identified in other systems when identifying them in a system that is not processed.
3. Make an instance of a requirement group with a set of requirements. As mentioned in step 2, already identified requirements are used to identify them in a system, thus, similar requirements are already grouped in general.
4. Find a criterion to distinguish between such requirements. We normally represent a criterion in the form of “XXX is preferred”. In step 2, we have already identified such criterion in general because we compare a requirement with similar requirements.
5. Make an instance of a characteristic and relate the instance to an instance of a requirement group. We may give a name to the instance of a characteristic so as to easily identify the characteristic, e.g., usability, reliability and so on.
6. Put requirements in order with respect to the identified criterion.
7. Calculate correlations among the instances of requirements groups by comparing the ordering of each requirements set.

4.2 Techniques for Comparison and Ordering

Comparison between requirements plays an important role in this method. Currently, we use the following techniques for comparison.

- *Focusing on the numeric attributes in requirements*: When each requirement has numeric attributes, we can naturally compare a requirement with other similar requirements. The typical examples are "the number of users" and "the response time" in Figure 3.
- *Surroundings in a use case* [8]: When use case diagrams for existing systems can be available, we can use structural similarity about a requirement (a use case).
- *Specification matching* [12], [7]: When we regard each requirement as a function, the requirement has pre-conditions and post-conditions. We use logical implication among pre-conditions and/or post-conditions for comparison.
- *Use Case Points* [11]: When we want to focus on the usability about some functionality, behavioral or interactive aspect is important. Because use case points mainly focus on the number and extent of interaction between a system and external things, we can use use case points technique to compare requirements with respect to interactive aspect.
- *AHP* [10]: We sometimes want to put requirements in order with respect to a preference of specific users. In such a case, AHP technique is useful because we can put requirements in order subjectively and systematically.

We may of course use several techniques together for a requirements group. We may also use the other techniques for comparison. From our experiences, identification of requirements and comparison between them mutually complement with each other. In other words, we can effectively identify requirements in existing systems when we compare the resources of each system.

Consequently, we mainly focus on so-called non-functional aspects when we put requirements in order. However, we do not emphasize that point because our ordering framework can be applied to functional aspects in requirements. Actually, functional and non-functional characteristics are not distinguished in ISO quality standards [6].

4.3 Scalability

Apparently, the characteristics that should be considered would increase when knowledge in new existing systems is added to an existing model. Especially, we could explore new criteria for evaluating existing functionalities or attributes when we compare a function with other similar functions.

If we calculate correlations between characteristics by using small number of systems, there are many correlated pairs that have no causal or semantic relationship in fact. If there are a lot of such pairs, we cannot efficiently check side effects among characteristics. However, such kind of pairs could be reduced when the number of systems increases. The reason is that coincidental correlations would be reduced by the many facts in existing systems. We will confirm this point in the next section. Hopefully, correlations between characteristics can be calculated systematically, we do not mind that the number of characteristics increases. Even if the number of such pairs is not reduced, we may reject the consequences from the pairs. We intend to detect causal relationships or side effects not automatically but interactively.

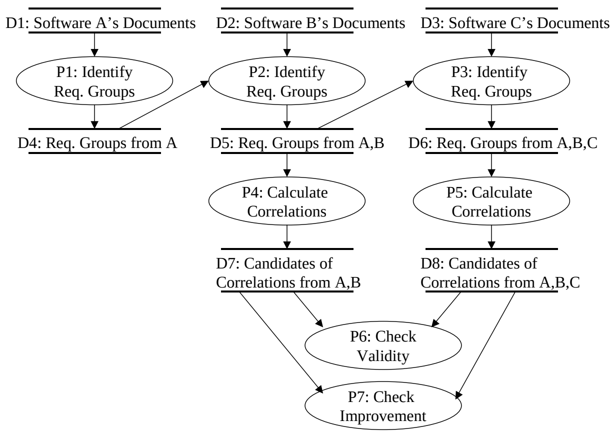

D1: Software A's Documents
D2: Software B's Documents
D3: Software C's Documents
P1: Identify Req. Groups
P2: Identify Req. Groups
P3: Identify Req. Groups
D4: Req. Groups from A
D5: Req. Groups from A,B
D6: Req. Groups from A,B,C
P4: Calculate Correlations
P5: Calculate Correlations
D7: Candidates of Correlations from A,B
D8: Candidates of Correlations from A,B,C
P6: Check Validity
P7: Check Improvement

Figure 7. Work Flow of Case Study

5. Case Study

We want to confirm the following hypotheses.

- It will be easy to develop a model by comparing several existing systems.
- The model will contribute to know mandatory, optional and unstable characteristics in a domain.
- Correlations between characteristics will be useful to find causal relationships among the characteristics.
- Quality of correlations will be improved when the number of existing systems increases.

The case study here is designed to confirm the hypotheses above.

5.1 Design of A Case Study

Figure 7 shows the workflow of this case study written in Data Flow Diagram. We mainly explain each process in the workflow as follows.

- An analyst obtains documents of three pieces of software, which belong to the same application domain. We call these documents as D1, D2 and D3.
- (Process P1) By reading D1, the analyst identifies requirements groups and their corresponding criteria and characteristics. As a result, the analyst obtains several requirements groups, so called D4. Because D4 is generated from only one piece of software, these are no orderings in each requirements group. Thus, we cannot discuss correlations in this step.

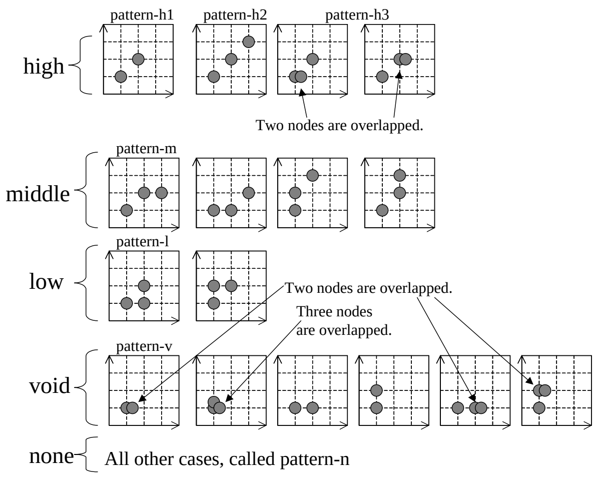
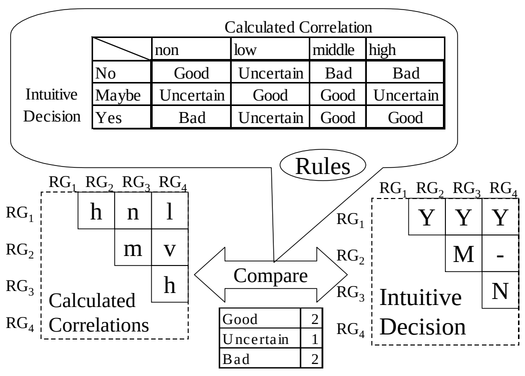

pattern-h1
pattern-h2
pattern-h3
high
Two nodes are overlapped.
pattern-m
middle
pattern-l
low
Two nodes are overlapped.
Three nodes are overlapped.
pattern-v
void
none
All other cases, called pattern-n

Figure 8. Patterns of Requirements Groups' Pairs (Positive Cases Only)

- (Process P2) By reading D2 and D4, the analyst does the same tasks above. As a result, the analyst obtains extended requirements groups, so called D5.
- (Process P3) By reading D5 and D3, the analyst also does the same tasks above. As a result, the analyst obtains more extended requirements groups, so called D6.
- (Process P4) By using D5, the analyst calculates correlations about all pairs of requirements groups. We call the data of such correlations as D7. Because D5 is generated from two pieces of software, patterns of correlations are pattern-h1, pattern-n or pattern-v in Figure 8. We will explain Figure 8 in detail below.
- (Process P5) By using D6, the analyst calculates correlations about all pairs of requirements groups. We call the data of such correlations as D8. Because D6 is generated from three pieces of software, patterns of correlations are pattern-h2, pattern-h3, pattern-m, pattern-l, pattern-n or pattern-v in Figure 8.
- (Process P6) By reading the contents of D7 and D8, the analyst checks whether generated correlations meet our intuition or not, as shown in Figure 9.
- (Process P7) By comparing D7 and D8, the analyst checks whether the quality of such correlations is improved or not. We will explain how to check it by using the Figure 10 below.

We regard charts in Figure 8 as patterns of requirements groups' pairs, and patterns corresponding to the first 10 charts (pattern-h1 to pattern-l) are regarded as correlations. Negative correlations are also represented in the same way. These are 16 charts in this Figure and each chart is written in the same way as a chart in Figure 3. Along with the progress of P2 and P3, some correlated pairs could become uncorrelated pairs (pattern-n). In addition, some uncorrelated pairs in D5 could also become correlated pairs because we regard patterns like pattern-h3, pattern-m and pattern-l


Calculated Correlation

Intuitive Decision

	non	low	middle	high
No	Good	Uncertain	Bad	Bad
Maybe	Uncertain	Good	Good	Uncertain
Yes	Bad	Uncertain	Good	Good

Good	2
Uncertain	1
Bad	2

Figure 9. Validation of Correlations

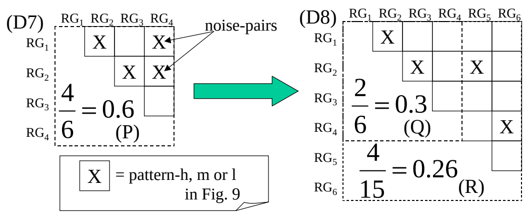


Figure 10. Noise Reduction in Correlations

as correlated pairs.

We will check whether calculated correlations are valid or not in the same way as an example in Figure 9. As shown in Figure 8, correlations are categorized into high, middle, low, void or none. Separately from the correlations, some domain expert evaluates each pair of requirements groups whether the pair is really related (Yes), never related (No) or else (Maybe). By comparing the category of correlations with intuitive decision by the expert according to the rules in Figure 9, we validate the correlations. Note that we do not take care correlations in pattern-v.

Finally, we explain how to check the improvement of correlations' quality along with the progress of P4 and P5 by using an example in Figure 10. We assume there are many correlated pairs that have no causal or semantic relationship in fact. We call such pairs as *noise-pairs*. If there are many noise-pairs, the efficiency for detecting incorrectness and inconsistency goes down. We also assume the number of such noise-pairs will decrease when the number of existing systems increases. Therefore, we focus on the changes of the correlated pairs' rate for confirming the improvement of the quality of correlations. In this example, D5 has four requirements groups and D6 has six because two new groups are found in P3. We have to check 60% (labeled by (P) in Figure 10) of all possible pairs in D7, whether each pair is related semantically or not. On the other hand, we may only check 30% or 26% (labeled by (Q) or (R)) because two noise-pairs are reduced in D8. There are two interpretations about this noise reduction. In the first interpretation, we focus on the changes from (P) to (Q), and rate of noise-pairs that have already existed in D7 are reduced. In the second interpretation, we focus on the

Table 1. Sizes of Software D1, D2 and D3

	D1	D2	D3
Number of Pages	4	20	102

Table 2. Numbers of Requirements Groups: D4, D5, D6

		not dispersed
D4 (from D1)	17	17
D5 (from D4 and D2)	25	4
D6 (from D5 and D3)	33	3

changes from (P) to (R), and rate of noise-pairs out of all possible pairs are reduced.

5.2 Result

In this case study, an application domain about software-music-player was selected. One of the authors played a role of an analyst in this case study. He was a graduate school student and well knew the domain. He downloaded three pieces of such software via a web site [1]. We could use these players without a fee on the windows PC. Each piece of software included user manuals written in windows help format (.hlp or .chm), and he used the manuals as D1, D2 and D3. For convenience, he converted the manuals into PDF format. He also wrote use case diagrams for each player to understand them well. The sizes of the manuals are shown in Table 1. Each manual had descriptions how to use the player and descriptions of changes and updates.

- **result 1**: Table 2 shows the results about D4, D5 and D6, the numbers of identified requirements groups in each step. The right most columns in the Table show the numbers of groups where requirements were not dispersed. Dispersed requirements here mean at least one requirement is different from others. Thus a requirements groups' pair is categorized into pattern-v in Figure 8 when one of the requirements groups in the pair does not have dispersed requirements. Apparently, all groups in D4 could not have dispersed requirements because only one piece of software was handled in D4. In the cases of D5 and D6, there were a few groups without dispersed requirements.
- **result 2**: The analyst could easily identify requirements groups in P2 and P3 because he could compare a requirement with others. In other words, he could know characteristics and criteria corresponding to each group. On the other hand, he only listed up main functionalities in P1. He always used the techniques in Section 4.2 and the most used one was "surroundings".
- **result 3**: He found several requirements (functions) that could not be related to any requirements groups because the requirements were supported only one piece of software. Note that the number of requirements in a requirement group should be more than two by definition in Figure 2. At least, he found seven examples of such requirements.

[1]http://www.vector.co.jp/download/
We can get many kinds of software via this web site without a fee.

Table 3. Validation of Correlation

	D7 (from D1, D2)	D8 (from D1, D2, D3)
Num. of Groups	25	33
Number of Possible Pairs (α)	300	528
Num. of Pattern-v in Fig. 9 (β)	90	93
$\alpha - \beta \Rightarrow$ Total	210 (100%)	435 (100%)
Good	50 (24%)	137 (32%)
Uncertain	32 (15%)	158 (36%)
Bad	128 (61%)	140 (32%)

- **result 4**: As mentioned in result 1 and Table 2, there were several requirements groups without dispersed requirements. The analyst investigated the contents of requirements and found that characteristics and criteria corresponding to such groups seemed to be mandatory in this domain. Examples are "playing a tune at once" and "supporting a list of tunes".
- **result 5**: On the contrary, the analyst found that there were groups where all requirements were completely different. In many cases, two requirements were similar but another was different with respect to a criterion. However, he found all three requirements were different with each other. He investigated the contents of such cases and found that characteristics corresponding such requirements groups seemed to be unstable in this domain. Examples were "displaying the status of software (e.g., name of the tune of time passed)" and "look and feel (usually called skin)".
- **result 6**: The analyst investigated the contents of pairs of requirements groups that had a correlation. Most pairs were seemed to have causal relationships with each other. For example, a pair "displaying the status of software is preferred to be complicated" and "support for quick operations (usually called shortcut) are preferred" was correlated with each other, and these characteristics seemed to have causal relationships.

  In the same way in Figure 9, the analyst validated the correlations. Table 3 showed the results. Correlations in D8 were better than those in D7 because the ratio of "Good" increased (from 24% to 32%) and the ratio of "Bad" decreased (from 61% to 32%).
- **result 7**: According to the example in Figure 10, the analyst checked whether correlations's quality was improved along with the increase of the number of existing systems. The results are summarized in Table 4. He did not apply any statistical tests about this result, but he could know the tendency that the quality was improved ($(\mathrm{P}) > (\mathrm{Q}), (\mathrm{P}) > (\mathrm{R})$). He checked the contents of noise-pairs and almost all the noise-pairs seemed to be unrelated semantically or causally.

Table 4. Resuls of P7: Noise Reduction about Correlations (See Fig. 7 and 10)

	D7 (from D1, D2)	D8 (from D1, D2, D3)	
Num. of Groups	25		33
Number of Possible Pairs (α)	300		528
Number of Correlated Pairs (γ)	210	155	362
Rate of Correlations γ/α	0.70 (P)	0.52 (Q)	0.69 (R)

5.3 Discussion

From the result of this case study, we can intuitively confirm most hypotheses. Especially, comparison among similar functions or attributes, that is one of the main ideas of this study, seems to be useful. Techniques for ordering requirements in Section 4.2 are also useful. Because use case diagrams were written in this case study, the technique of "surroundings" would be frequently used. However, we find a problem. We sometimes cannot put requirements in total order but in partial order. As shown in our meta-model in Figure 2, requirements are basically put in total order. According to the result in this case study, we will modify our meta-model.

Correlations between characteristics seem to be also useful to find causal relationships or side effects as shown in the result 6, especially correlations calculated from three existing systems. From the results 6 and 7, correlations seem to be scalable because the validity and quality are improved along with the increase of the number of existing systems.

6. Related Work

One of the famous researches for handling software characteristics is QFD [5]. In QFD, relationships among stakeholders' requirements and software characteristics are related for developing suitable software. We think it is not so easy to identify such characteristics, thus we focus on exploring such characteristics.

Our approach consequently deals with software quality characteristics. Actually, we inspired the description in ISO standards [6]. Our approach also consequently handles non-functional requirements [2]. However, we do not mentioned in this point because our approach also handles functional requirements. The selection for comparison techniques such as in Section 4.2 will depend on whether a characteristic relates to functional aspect or non-functional one.

There are several researches about domain knowledge. For example in FODA [9], method to identify the distinctive characteristics (features) of a specific domain was proposed and one of the basic concepts of FODA was abstraction. On the other hand in our approach, our basic concept is comparison. LEL [4] also focused on the acquisition for domain knowledge, and an approach based on natural language processing was used. Our method is mainly designed for handling correctness, completeness, consistency and unambiguity, but LEL was not explicitly intended for a specific purposes.

Prioritization techniques such as DDP [3] and WinWin [1] will be also incorporated with ours because we need various kinds of comparison and ordering techniques.

7. Conclusion

In this paper, we propose the model and the methods for using and constructing a domain model. With the model, we can elicit requirements correctly, consistently, completely and unambiguously. Our model and methods are based on the comparison among existing systems in the same domain, and the method to build the model is scalable. In other words, validity and quality of the model can be improved along with the increase of the number of existing systems. From a case study, we partially confirmed the usefulness of our model and methods. Because there are several procedures that could be automatically calculated in our methods, CASE tools could be useful for our methods. Actually, the analyst in the case study felt that supporting tools were required.

References

[1] B. Boehm, et al. Developing Groupware for Requirements Negotiation: Lessons Learned. *IEEE Software*, 18(3):46–55, May/Jun. 2001.

[2] L. Chung, et al. *Non-functional Requirements in Software Engineering*. Kluwer Academic Publishers, 2000.

[3] S. L. Cornford, M. S. Feather, J. C. Kelly, T. W. Larson, B. Sigal, and J. D. Kiper. Design and Development Assessment. In *Proceedings of IWSSD'00*, pages 105–114, 2000.

[4] J. C. S. do Prado Leite and A. P. M. Franco. A Strategy for Conceptual Model Acquisition. In *Proceedings of RE93*, pages 243–246, 1993.

[5] G. Herzwurm, S. Schockert, and W. Pietsch. QFD for Customer-Focused Requirements Engineering. In *Proceedings of RE'03*, pages 330–338, Sep. 2003.

[6] International Standard ISO/IEC 9126-1. Software engineering - Product quality - Part 1: Quality model, 2001.

[7] H. Kaiya and K. Kaijiri. Conducting Requirements Evolution by Replacing Components in the Current System. In *Proceedings of APSEC'99*, pages 224–227. Dec. 1999.

[8] H. Kaiya, A. Osada, and K. Kaijiri. Identifying Stakeholders and Their Preferences about NFR by Comparing Use Case Diagrams of Several Existing Systems. In *Proceedings RE'04*, pages 112–121, Sep. 2004.

[9] K. C. Kang, et al. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21 ESD-90-TR-222, CMU, Nov. 1990.

[10] T. L. Saaty. *The analytic hierarchy process: planning, priority setting, resource allocation*. RWS, 1990.

[11] G. Schneider and J. P. Winters. *Applying Use Cases: A Practical Guide*. Addison-Wesley, 2nd edition, 2001.

[12] A. M. Zaremski and J. M. Wing. Specification Matching of Software Components. *ACM Trans. Software Eng. and Methodology*, 6(4):333–369, Oct. 1997.