So/M: A Requirements Definition Tool using Characteristics of Existing Similar Systems

Naoyuki Kitazawa, Akira Osada, Kazuyuki Kamijo, Haruhiko Kaiya, Kenji Kaijiri
Shinshu University, Nagano, JAPAN
`http://www.cs.shinshu-u.ac.jp/~kaiya/`

Abstract

During a software system development, existing similar systems are useful because such systems and their documents help developers to understand and reuse problems and solutions of the new system. Especially, such helps are effective if developers are not familiar with the new system. In this paper, we present a supporting tool called "So/M" for an analyst to define software requirements. By using So/M, the analyst can easily refer functions of existing similar systems, their commonality and the relationships among them. So/M also enables the analyst to choose the candidate of requirements, and to generate the skeleton of requirements. During a comparative experiment, we have confirmed that So/M significantly helped analysts unfamiliar with a system to be developed.

1. Introduction

Software systems are applied to various kinds of areas, thus software engineers especially requirements analysts have to well understand such areas, e.g., healthcare management, military support, consumer electronics, stock markets and so on. Apparently, it is extremely hard to understand them well because each area requires significant expertise. Fortunately, there already exist some kinds of software systems to support activities in such areas. Therefore, it is reasonable idea to refer existing similar software systems when a new system is developed. Several methods and tools with complicated functions were already introduced [10], [11], [8], [18], but there is no discussion what is the fundamental and useful functionalities about such methods and tools. There is also no experimental or empirical validation about their effectiveness.

We assume that an analyst unfamiliar with an area can define requirements effectively and efficiently even when he/she can refer and analyze information about existing systems in simple ways, e.g., commonalities and differences among the systems. In this paper, we introduce a requirements definition tool called "So/M" [1], that provides information about existing systems in simple ways. By using So/M, we conducted a comparative experiment to confirm the effectiveness of simple support. As a result, we confirmed that even a simple support largely improved the quality of defined requirements and efficiency to define them.

[1]"So/M" is a nickname of the main developer of this tool.

The rest of this paper is organized as follows. In the next section, we investigate and discuss what is a good requirements specification and how to develop it by using information about existing systems. In section 3, we introduce a requirements definition tool called "So/M", that is developed based on the discussion in section 2. In section 4, we report an experiment, which purpose is to confirm the effectiveness of supporting tools like So/M tool. Finally, we summarize our current result and show the future works.

2. Requirements Definition Method and Process Based on Existing Systems

2.1 Requirements Definition and Specification

Before defining software requirements, we have to understand what are the good software requirements. Most textbooks [6, 14, 13] mentioned about this issue and they referred to the IEEE standard for Software Requirements Specifications [3]. In this standard, characteristics of a good SRS (Software Requirements Specification) are listed as follows.

1. Correct
2. Unambiguous
3. Complete
4. Consistent
5. Ranked for importance and/or stability (priority)
6. Verifiable
7. Modifiable
8. Traceable

All of them are of course important, and we think the most important characteristic is completeness. If missing requirements exist, i.e., the requirements are incomplete, other characteristics cannot be confirmed appropriately. Correctness is the second important characteristic because whether developed software meets stakeholders' needs depends on the correctness of SRS. It is not so easy to confirm correctness in early phase in software development, and techniques such as interview, workshops and/or prototype review are used. Unfortunately, these techniques are not cost-effective in general. In a practical point of view, rank of importance is very important because not all requirements can be implemented under limited budget and schedule. Rank of importance is sometimes referred as prioritization or triage [7]. Traceability is also important in practice because requirements are frequently changed thus impacts within requirements and to the other artifacts by the changes should be found efficiently and correctly. The most fundamental task for traceablity is a task for labeling all requirements for referring from the others.

2.2 Information about Existing Systems

One of the ways to develop good SRS is using information about already existing systems and their problems. When the number of referred systems increases, the number of characteristics such as functions and qualities that can be used in a new system also increases. Therefore, information about several systems contributes to completeness of the requirements for the new system. Even if all information of a system cannot be extracted, information extracted from other systems can complement total information.

In a software product line research, domain engineering is a set of techniques for this kinds of information [16]. In domain engineering, commonality and variability among similar systems are focused. This idea can be applied not only to products in a family but also to similar products. For example, by comparing existing similar systems with respect to its functionalities, functional commonality and variability can be identified and non-functional characteristics such as performance and usability can be also found [15].

Several types of data-models for representing such information are proposed [10], [11], [8], [18]. They define data-models for problems, functionalities, qualities and/or tasks, and provide several means of making software requirements better. For example, several inference rules are provided to improve correctness, completeness, unambiguity and so on in [10]. Most of these kinds of researches lack empirical and/or experimental validation of confirming its effectiveness.

RDF based standard notation will also contribute to making this kinds of information reusable. For example, libraries and tools exists based on this technology [2], [1]. For requirements definition, there is no standard notation and/or tool yet, thus it is a little bit hard to try using these technologies.

One of the obstacles for using information about existing systems is the difficulty to create such information. It is very important for requirements analysis to focus on both problems, solutions and their relationships [9], but information about problems, i.e., the reason why the systems are required, is hard to be gathered. On the other hand, information about solutions seems to be gathered semi-automatically. For example, NLP (Natural Language Processing) techniques were used to create domain ontology [12]. Genetic algorithm was used to classify software components based on requirements [4], thus solutions, i.e., components, can be structured based on requirements. Formal concept analysis can be also used for this purpose [17].

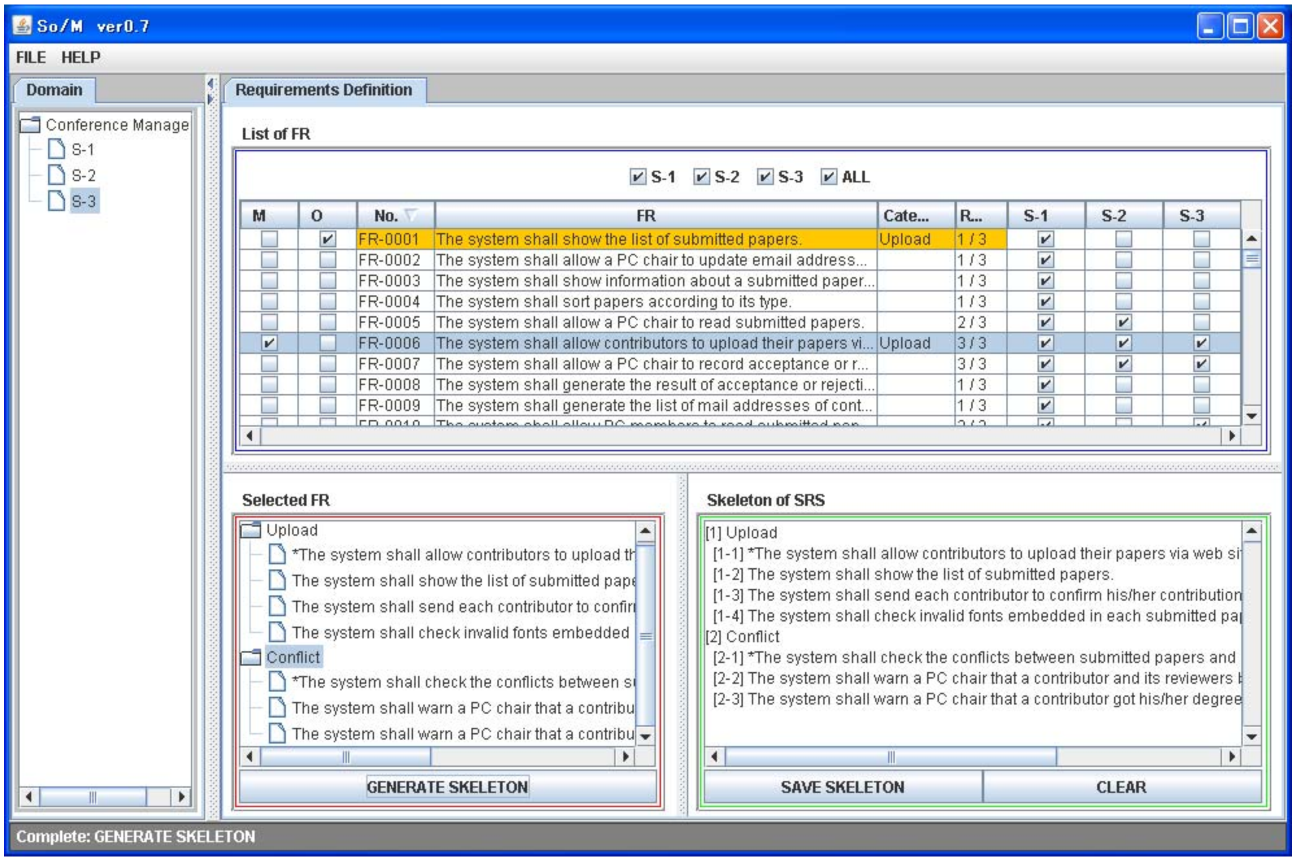
3. Supporting Tool

3.1 Requirements for Supporting Tools

As mentioned in last section, there are several methods and/or tools to improve software requirements. However, their effectiveness is not frequently validated through experimental or empirical studies. One of the reasons is the difficulty to create such information, but there are several means of creating information about solutions as mentioned in last section.

We will design and implement a requirements definition tool to validate fundamental idea about using information about existing systems. Methods and tools mentioned in last section provide various kinds of state-of-the-art means of analyzing characteristics of existing systems and to describe requirements for requirements analysts. Their fundamental and common means are not so complicated. Here we clarify such fundamental and common means as following requirements for methods and tools.

1. Methods and tools shall provide the list of existing systems that are similar to each other.
2. Methods and tools shall provide the list of characteristics such as functions or quality attributes.
3. Methods and tools shall provide commonalities and differences about such characteristics among existing similar systems.
4. Methods and tools shall provide relationships among such characteristics that are useful for choosing collocated characteristics.
5. Methods and tools shall enable analysts to choose and/or create requirements for solving new problems.
6. Methods and tools shall enable the analysts to make the quality of requirements better, e.g., more complete, correct, traceable and prioritized.
7. Methods and tools shall help analysts to perform their tasks efficiently.



M	O	No.	FR	Cate...	R...	S-1	S-2	S-3
	✓	FR-0001	The system shall show the list of submitted papers.	Upload	1 / 3	✓		
		FR-0002	The system shall allow a PC chair to update email address...		1 / 3	✓		
		FR-0003	The system shall show information about a submitted paper...		1 / 3	✓		
		FR-0004	The system shall sort papers according to its type.		1 / 3	✓		
		FR-0005	The system shall allow a PC chair to read submitted papers.		2 / 3	✓	✓	
✓		FR-0006	The system shall allow contributors to upload their papers vi...	Upload	3 / 3	✓	✓	✓
		FR-0007	The system shall allow a PC chair to record acceptance or r...		3 / 3	✓	✓	✓
		FR-0008	The system shall generate the result of acceptance or rejecti...		1 / 3	✓		
		FR-0009	The system shall generate the list of mail addresses of cont...		1 / 3	✓		

Selected FR

Upload
- *The system shall allow contributors to upload th
- The system shall show the list of submitted pape
- The system shall send each contributor to confir
- The system shall check invalid fonts embedded

Conflict
- *The system shall check the conflicts between s
- The system shall warn a PC chair that a contribu
- The system shall warn a PC chair that a contribu

Skeleton of SRS

[1] Upload
[1-1] *The system shall allow contributors to upload their papers via web si
[1-2] The system shall show the list of submitted papers.
[1-3] The system shall send each contributor to confirm his/her contribution
[1-4] The system shall check invalid fonts embedded in each submitted pa
[2] Conflict
[2-1] *The system shall check the conflicts between submitted papers and
[2-2] The system shall warn a PC chair that a contributor and its reviewers
[2-3] The system shall warn a PC chair that a contributor got his/her degree

Figure 1. Snapshot of So/M Tool

3.2 So/M tool: its Design and Implementation

Based on the requirements above, we developed a supporting tool called "So/M" for requirements definition. Because So/M is used in experiments to validate the effectiveness of fundamental means mentioned above, functionalities of So/M are limited. By using Figure 1, we will explain such functionalities.

Review information about existing systems. So/M provides the list of existing systems as shown in the left side in Figure 1. In this figure, three different conference management systems, S-1, S-2 and S-3, are listed. At the top right area, So/M provides the list of characteristics (functions) for each system, and their commonalities and differences. For example, all three systems have characteristics corresponding to FR-006 and FR-007, but FR-0005 is supported by only S-1 and S-2. The other characteristics shown in this figure are supported only by S-1. When its user wants to focus on characteristics only in S-1, he/she may simply puts or removes check boxes, i.e., "☑ S-1 ☐ S-2 ☐ S-3 ☐ ALL", in the top of figure. By choosing a line corresponding to a characteristic, So/M tells other characteristics related to the characteristic to the user. In this figure, the user knows that FR-0001 is related to FR006 by their color changes.

Choose candidates of requirements. So/M enables its user to choose existing characteristics as candidates of requirements for a new system. When the user chooses a characteristic, he/she should specify whether it is mandatory or optional. For example in Figure 1, FR-006 is chosen as mandatory requirement and FR-001 is chosen as optional. So/M enables the user to create groups, we call "category", of requirements. For example in this figure, a category named "Upload" was created, and at least FR-0001 and FR0006 are categorized into this group.

Create skeleton of requirements specification. Chosen candidates are listed at the bottom middle, labeled "Selected FR", in Figure 1. By bushing the button labeled "GENERATE SKELETON", a skeleton of requirements is generated at the right bottom area in this figure. All requirements in the skeleton are labeled by its own identifier, and mandatory requirements are prefixed by "*". By using this skeleton, the user may create a SRS according to some format.

Create information about existing systems. So/M tool has another mode for creating information about existing systems from documents such as manuals and help files semi-automatically. In the mode, a morphological analyzer and simple word matching technique are used to identify characteristics in each system and commonalities and difference between several systems. Issues about this mode is out of scope in this paper.

4. Experiment

4.1 Experiment Design

4.1.1 Purpose and Hypotheses

We formulate the goal of this experiment according to the Goal-Question-Metric paradigm [5] as follows.

Analyze So/M tool for requirements definition
for the purpose of investigating their effectiveness
with respect to the quality of defined requirements list and effort
from the perspective of domain expert
in the context of university students at a computer science course that are unfamiliar with the problem and its domain but are provided information about its solutions.

As mentioned in Section 3, So/M enables a requirements analyst to analyze information about existing systems for similar problem that he/she wants to be solved. Therefore, we expect the analyst can define requirements completely, correctly and efficiently, and the requirements may be traceable and prioritized. We then formulate null hypotheses as follows.

- $H1$ (Correctness): There is no difference between the correctness of defined requirements with So/M and without So/M.
- $H2$ (Completeness): There is no difference between the completeness of defined requirements with So/M and without So/M.
- $H3$ (Traceability): Whether requirements are traceable or not is independent of requirements definition process with So/M or without So/M.
- $H4$ (Priority): Whether requirements are prioritized or not is independent of requirements definition process with So/M or without So/M.
- $H5$ (Efficiency): There is no difference between the efficiency of a requirements definition process with and without So/M.

4.1.2 Design

The purpose of this experiment is to investigate the effect of means of analyzing existing systems provided by So/M. Therefore, the treatment is to apply requirements analysis tasks with and without So/M. We define the following two treatment types.

NoTool : no tool support for analyzing existing systems. The subjects use So/M to refer and analyze characteristics of existing systems to define requirements for a new system.

Tool : tool support for analyzing existing systems. The subjects refer and analyze such characteristics to define requirements for a new system.

Each subject carried out the experimental task alone. We have randomly assigned subjects to treatments. This allows us to assume independence between the treatments. Each subject performed the task for one treatment type.

4.1.3 Subjects, Objects and Tasks

About 25 bachelor students in computer science course in their third year participated in this experiment, which was conducted within a course in winter term of 2007 at our university. All subjects had already taken courses for C and Java programming, software engineering fundamentals. The students were motivated to perform well in this task because it was part of an assignment that was mandatory to pass the course. In addition, each of them may choose other two topics (embedded system design and DSP programming) for this mandatory course, but the 30 students intentionally choose requirements engineering.

The task of each subject was to define requirements for a system to support a program chair of a conference such as COMPSAC. We expected this task was unfamiliar to subjects. The task for each treatment type is depicted in Figure 2. Problems of a program chair are given as a short document as follows.

- It is hard to manage submissions.
- It is hard to remind reviewers who did not submit results on schedule to submit the results.
- It is hard to assign suitable reviewers to each submission.
- It is hard to gather suitable reviewers in advance.
- It is hard to collect camera ready papers and to check their validity.
- It is hard to decide candidates of accepted papers.
- It is hard to forget taking the minutes of PC meetings.
- I am worried about the number of submission and the progress of review.

Each subject can refer characteristics of three existing systems for conference management [2] in each way. 75, 62 and

[2]confman http://www.ifi.uio.no/confman/ABOUT-ConfMan/

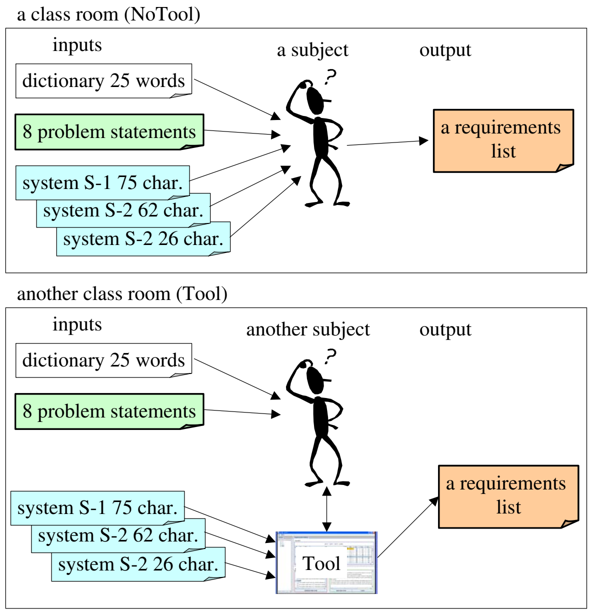


Figure 2. Tasks for each Treatment type

26 characteristics, especially functions of each system, for each system are used. Because these characteristics were extracted from specific documents for each system, all characteristics in an actual system could not be provided for our subjects. One of authors, who was an expert of conference management checked the validity of these characteristics. The total number of characteristics is 137 when duplicated characteristics are eliminated. In addition, each subject can refer a term dictionary with 25 domain specific words and each explanation, e.g., camera ready, review, PC member and so on.

The task of subjects with treatment NoTool was to choose the requirements out of the characteristics by using a text editor in his/her own laptop. The task was completed in class, less than 3 hours. We call the list of chosen requirements as "a requirements list". Each subject can refer the characteristics of each system as a text file respectively. He/she can also refer the problems of a chair and the term dictionary as a text files and printed documents respectively. The subjects were not allowed to gather information via Internet. We asked subject to upload his/her requirements list just after completing his/her task. To improve subjects' motivation to choose requirements, each subject was asked to complete a requirements specification at home after class.

The task of subjects with treatment Tool was also to choose the requirements out of the characteristics. Each subject can refer the characteristics via So/M tool and also choose requirements by using So/M tool. The other things were the same as the task with treatment NoTool.

openconf http://www.zakongroup.com/technology/openconf.shtml
and cyberchiar http://borbala.com/cyberchair/

4.1.4 Operation

Table 1. Operation Overview

Date	group X	group Y
Nov. 14, 21, 28	lecture and lesson for req. def.	
Dec. 05	NoTool	lesson for tool
Dec. 12	lesson for tool	Tool
Jan. 09	exercise with tool	exercise without tool
Jan. 16	feedback, questionnaire, workshop	

In Table 1, we show the schedule of our experiment. We spent 7 weeks for this experiment. For each week, at least 3 hours were used in class. To improve comprehension of subjects for requirements definition, first 3 weeks were spent and each subject wrote a requirements list by him/herself and revised each list after review by a lecturer. The problems were about library systems. During the lecture, subjects learned characteristics of a good SRS, e.g., completeness, correctness and so on. Therefore, all subjects had already learned and experienced requirements definition at least till Dec. 05.

In Dec. 05, 12 and Jan. 09, we divided subjects into two groups randomly and subjects in each group achieved lesson or exercise for requirements definition in different rooms. As shown in Table 1, we regard tasks in Dec. 05 by group X as NoTool, and tasks in Dec. 12 by group Y as Tool. Because subjects of treatment NoTool did not know So/M tool, they never achieve their tasks according to the means provided by So/M tool. Because subjects of treatment Tool spent learning how to use So/M tool in Dec. 05 by analyzing software music player like iTunes or Windows media player, most of them became skilled in using So/M tool. For equality of learning, we let subject to achieve an exercise with or without So/M tool to solve problems in another domain, i.e., web browser in Jan. 09.

In Jan 16, we show the statistical results of our experiment to our subjects, and then gave questionnaire. Finally, subjects and we had a workshop to discuss how to define requirements in high quality.

4.1.5 Data Collection and Analysis Technique

We focus on the requirements lists and spending time to complete them. Because a requirement in a requirements list is a characteristic of existing systems, in other words, a function provided by one or more systems, we simply identify the requirement and count the number of requirements. Because the lists were written in class and gathered via our web site, we could collect spending time for each subject.

We used a questionnaire to collect data about the subjects' impression and attitude towards the task. The 23 questions and inquiries of the questionnaire were distributed and collected after showing the results in Tables 2 and 3 in Jan 16 as shown in Table 1. Within the questionnaire, we confirmed whether his/her data may be used or not in our research. We also gathered the answers whether each subject understands the differences of requirements quality with or without So/M and the reasons via the questionnaire.

As mentioned in section 4.1.1, we want to know the quality of requirements list and its effort. An author of this paper who was an expert of conference management tasks developed a correct requirements list. The correct list consists of 83 requirements.

By comparing the correct requirements list and a requirements list written by each subject, the quality of each requirements list can be measured. We use the following metrics for quality. For simplicity, we abbreviate the correct requirements list to "$CORR$", and a requirements list written by a subject x to "$SUBJ(x)$". Because $CORR$ and $SUBJ(x)$ are sets of requirements, we can apply general set operations, e.g., cardinality of S as $|S|$, to $CORR$ and $SUBJ(x)$.

- $Precision(x) = \dfrac{|SUBJ(x) \cap CORR|}{|SUBJ(x)|}$
  This metric almost corresponds to correctness of requirements.
- $Recall(x) = \dfrac{|SUBJ(x) \cap CORR|}{|CORR|}$
  This metric almost corresponds to completeness of requirements.
- $Traceable(x) =$ if ($\forall i \in SUBJ(x)$: i has own label) then return $true$ else return $false$ fi
  Traceablity is achieved only when each requirement in a requirements list has its own label, such as 1, 2 and 3. Therefore, this metric will measure most fundamental aspect of traceability.
- $Prioritize(x) =$ if ($\exists i \in SUBJ(x)$: i is prioritized) then return $true$ else return $false$ fi
  Ideally, all requirements in a requirements list should be prioritized, e.g., labeled with mandatory or optional mark. However, requirements at only special level could be marked for specifying priority. Thus, this metric focuses on existence of prioritized requirements.
- $Performance(x) = \dfrac{\text{time spent by } x}{|SUBJ(x)|}$
  We do not use $|SUBJ(x) \cap CORR|$, i.e., the number of correct requirements chosen by x, as a denominator of this equation because both correct and incorrect requirements consume subjects' time.

For each metric except $Traceable(x)$ and $Prioritize(x)$, we calculate the average of its value in each type of treatment. Then, we achieve t-test to check whether the difference is statistically significant or not. For $Traceable(x)$ and $Prioritize(x)$, we simply count the number of $true$ value in each type of treatment, and check its difference.

4.2 Results

4.2.1 Overview

Table 2. Overview of Results

	NoTool	Tool
Num. of subjects . . . . (a)	13	10
Average num. of req's	24.6	94.8
Average num. of correct req's	22.2	62.3
Num. of traceable list . . . . (b)	1	10
b/a	7.7 %	100.0 %
Num. of prioritized list . . (c)	4	10
c/a	30.7 %	100.0 %
Average time spent (min.)	101	116
Average performance (min.)	4.9	1.3

Table 2 shows the overview of results. Because several students gave up this credit or did not agree with the research usage of his/her data, the numbers of subjects in two treatments are not completely the same. Note that the number of subjects is the same as the number of requirements list because each subject wrote one list.

4.2.2 H1: Correctness

As shown in Table 3, H1 is rejected because the average precision of NoTool is statistically different from the average of Tool. Therefore, we may regard there is significant difference between NoTool and Tool with respect to correctness. However, the average precision of NoTool is better than the average of Tool against our expectation.

We assume subjects in NoTool chose requirements too carefully and the number of requirements became too small because there was no clear guidance to choose requirements. On the other hand, we assume subjects in Tool tended to choose some characteristics in existing systems without deep consideration because So/M tool enables subjects to choose such characteristics easily.

Latter assumption was validated by the answers of our questionnaire. With respect to practical usage, low precision is not so big problem because analysts may reject incorrect requirements by reviewing the list. According to the result of questionnaire, most of all subjects understand the result about H1. Several subjects pointed out that So/M tool enabled them to choose characteristics easily, thus they tended to fix their choices without enough consideration.

Table 3. Statistical Results (Note: The number of req's in correct list is 83.)

	NoTool	Tool	significant ($p < 0.05$)	p value
Average num. of requirements	24.6	94.8	yes	2.35×10^{-8}
Average time spent (min.)	101	116	no	2.724×10^{-1}
Average performance (min.)	4.9	1.3	yes	4.645×10^{-3}
Average precision	90.3%	66.3%	yes	1.4×10^{-10}
Average recall	26.8%	75.1%	yes	2.1×10^{-9}

4.2.3 H2: Completeness

As shown in Table 3, H2 is rejected because the average recall of NoTool is statistically different from the average of Tool. Fortunately, the average recall of Tool is largely better than the average of NoTool, thus we may regard So/M tool is effective for completeness. Even though the subjects did not know this problem domain, i.e., conference management, and they were only bachelor students, the average recall was more than 70%.

According to the result of questionnaire, most of all subjects understand the result about H2. Several subjects pointed out that So/M told them related characteristics, thus they can relatively chose complete requirements. They also pointed out that So/M enabled them to confirm whether a characteristic was chosen or not, thus they could avoid missing requirements.

4.2.4 H3: Traceability

As shown in line (b) in Table 2, all subjects of Tool put labels for each requirement because So/M tool automatically put such labels. The important result is that only one subject of NoTool put such labels. As shown in Table 1, there were three classes before our experiment and all subjects learned and experienced the importance of traceablity, but the result was terrible. This result strongly supported the necessity for tool support about traceablity.

According to the result of questionnaire, most of all subjects understand the result about H3. They of course pointed out that results in Tool treatment were a natural result because So/M automatically put labels. They also pointed out that labeling requirements was tedious task, thus they did not want to do that in NoTool treatment.

4.2.5 H4: Priority

As shown in line (c) in Table 2, all subjects of Tool prioritize requirements because So/M tool enforced prioritization. In the same way as tracablity, all subjects learned and experienced the importance of prioritization during three classes before our experiment, but the result was not so good in the treatment of NoTool. This result also supported the necessity for tool support about prioritization. We forgot to ask their understandings and reasons about H4, but this result is also natural because So/M enforce a choice of mandatory or optional on its user.

4.2.6 H5: Efficiency

As shown in Table 3, H5 is rejected thus the average performance in NoTool is statistically different the average in Tool. According to the definition, performance depends on the number of defined requirements and spending time. As shown in Table 3, spending times with or without So/M were almost the same, thus the performance largely depends on the numbers of defined requirements.

We assume easy operation of So/M tool to choose characteristics caused this result, and we asked this point in questionnaire and during our workshop. Against our assumption, most subjects answered easiness of choosing was not the cause of this result because they spent most in investigating the problems and solutions. They also commented that So/M tool helps them to distinguish chosen characteristics from others, to identify commonality among existing systems. Therefore, we may regard So/M provided easiness for not only operation but also argumentation for requirements definition.

4.3 Threats to Validity

Internal Validity. Threats to internal validity can affect the independent variables of an experiment. In this experiment, independent variables are defined requirements list and spending time. As shown in Table 1, NoTool treatment was achieved in Dec. 05 and Tool treatment was achieved in Dec. 12, thus some subjects in NoTool could leak the contents of exercise to subjects in Tool. Beforehand, we sent informal questionnaire to bachelor students about their communication and topics. They answered they rarely did not discuss the contents of classes.

External Validity. Threats to external validity reduce the generalizability of the results. To increase generalizability, we ask homework to complete arequirements specification based on the requirements list. We use students as subject, which might be a large threat to external validity. However, all students in this experiment learned programming languages and software engineering in advance, and they learned and experienced requirements definition just before this experiment. According to the answers of questionnaire, no subjects knew the task for conference management, thus our results are enough general in the context of requirements definition for unfamiliar tasks.

Construct Validity. Construct validity is the degree to which the variables measure the concepts they are to measure. We assume most metrics in Section 4.1.5 seem to be suitable to measure correctness, completeness, prioritization and performance. Metric for traceablity in this paper is a little bit weak because labeling is only focused and there are a lot of other factors for traceability. The correct requirements list is one of the key factors in construct validity. This correct list was written by one of authors, who was PC member of several conferences and PC chair of one symposium, thus the list will be correct.

Conclusion Validity. Conclusion validity is concerned with the relation between the treatment and the outcome. Except traceablity and prioritization, we achieve statistical test, thus at least tested results were reliable.

5. Conclusion

In this paper, we introduce a requirements definition tool called So/M, that provides fundamental functionalities based on information about existing systems. We had an experiment to confirm the effectiveness of such functionalities by using So/M. As a result, So/M contributes to improving completeness, traceablity and prioritization about defined requirements and the efficiency for defining them, but hindered improving correctness. According to the results of questionnaire, the following functionalities contributed to them; showing related characteristics among existing systems, distinguishing chosen characteristics from others, achieving monotonous tasks, e.g., labeling, automatically. This kind of supporting tool should enable its user to review their choice and/or decision for improving the correctness of defined requirements.

As mentioned in section 2, there are a lot of state-of-the-art functionalities to improving the quality of requirements. We have to compare effectiveness by So/M and effectiveness by other complicated functionalities. For example, So/M do not provide types of relationships among characteristics of existing systems. If So/M is extended for providing typed relationships and its effectiveness are getting better, we can confirm the usefulness of typed relationships objectively.

In our current work, we do not mention the quality and quantity of information about existing systems. Supporting tools should be scaleable with respect to the amount of information, but we do not have confirmed this point yet. Therefore, we would like to investigate this point. Supporting tools cannot work validly with information of bad quality, thus information quality problem depends on how to create such information. Therefore, we assume the quality of information is high enough in the context of researches about using such information.

Acknowledgements

This work is partially supported by International Information Science Foundation (IISF), Japan.

References

[1] DAML Ontology Library. http://www.daml.org/ontologies/.

[2] KAON Tool Suite. http://kaon.semanticweb.org/.

[3] IEEE Recommended Practice for Software Requirements Specifications, 1998. IEEE Std. 830-1998.

[4] A. S. Andreou, D. G. Vogiatzis, and G. A. Papadopoulos. Intelligent Classification and Retrieval of Software Components. In *COMPSAC 2006*, pages 37–40, 2006.

[5] V. R. Basili and D. M. Weiss. A Methodology for Collecting Valid Software Engineering Data. *IEEE Transactions on Software Engineering*, SE-10(6):728–738, Nov. 1984.

[6] A. Davis. *Software Requirements*. Prentice-Hall, 1990.

[7] A. M. Davis. The Art of Requirements Triage. *Computer*, pages 42–49, Mar. 2003.

[8] J. C. S. do Prado Leite and A. P. M. Franco. A Strategy for Conceptual Model Acquisition. In *Proceedings of First IEEE International Symposium on Requirements Engineering*, pages 243–246, 1993.

[9] M. Jackson. *Problem Frames, Analyzing and structuring software development problems*. Addison-Wesley, 2000.

[10] H. Kaiya and M. Saeki. Using Domain Ontology as Domain Knowledge for Requirements Elicitation. In *Proc. of RE06*, pages 189–198, Sep. 2006. IEEE CS.

[11] J. Kato, M. Saeki, A. Ohnishi, M. Nagata, H. Kaiya, S. Komiya, S. Yamamoto, H. Horai, and K. Watahiki. PAORE: Package Oriented Requirements Elicitation. In *Proc. of APSEC 2003*, pages 17–26, Dec. 2003. IEEE Computer Society Press.

[12] M. Kitamura, R. Hasegawa, H. Kaiya, and M. Saeki. An Integrated Tool For Supporting Ontology Driven Requirements Elicitation. In *ICSOFT 2007*, pages 73–80, Jul. 2007.

[13] G. Kotonya and I. Sommerville. *Requirements Engineering Process and techniques*. Wiley, 1998.

[14] S. Lauesen. *Requirements Engineering Styles and Techniques*. Addison-wesley, 2002.

[15] A. Osada, D. Ozawa, H. Kaiya, and K. Kaijiri. Modeling Software Characteristics and Their Correlations in A Specific Domain by Comparing Existing Similar Systems. In *QSIC 2005, Proc. of QSIC05*, pages 215–222, Sep. 2005. IEEE Computer Society.

[16] K. Pohl, G. Bockle, and F. V. D. Linden. *Software Product Line Engineering: Foundations, Principles And Techniques*. Springer-Verlag New York Inc, 2005.

[17] W. ZHOU, Z. tian LIU, and Y. ZHAO. Ontology Learning by Clustering Based on Fuzzy Formal Concept Analysis. In *COMPSAC 2007*, pages 204–210, 2007.

[18] L. Zong-yong, W. Zhi-xue, Y. Ying-ying, W. Yue, and L. Ying. Towards a Multiple Ontology Framework for Requirements Elicitation and Reuse. In *COMPSAC 2007*, pages 189–195, 2007.