Software Requirements Parts for Construction of Software Requirements Specifications

Yoshitaka Iyoda[1] and Atsushi Ohnishi[2]
[1]*Graduate School of Science and Engineering, Ritsumeikan University, 1-1-1 Noji Higashi, Kusatsu 525-8577, Japan*
(Currently, Mr. Iyoda is at Hitachi Ltd., Japan)
[2]*Department of Computer Science, Ritsumeikan University, 1-1-1 Noji Higashi, Kusatsu 525-8577, Japan*

Keywords: Software Requirements Parts, Software Requirements Specification (SRS), Construction of Software Requirements Specification.

Abstract: In software developments, a software requirements specification (SRS) must be correctly specified. An SRS becomes large and complicated when system to be developed become large. It takes a lot of efforts and costs to newly specify a correct SRS. The authors propose a method for generating SRS parts. Using SRS parts an SRS can be easily constructed. First a domain expert decomposes an SRS into functional requirements, and then he/she derives parts of functional requirements from them SRS. In order to improve the reusability, derived SRS parts will be abstracted using a thesaurus. The authors have been developed a prototype system for abstracting SRS parts. The proposed method will be illustrated with examples and evaluated through an experiment.

1 INTRODUCTION

A software requirements specification (SRS) is a final product of software requirements definition process and will be referred in later phases of software development. SRS can be used for users' validation of elicited and specified requirements and for developers' review of SRS. So an SRS should be correct. However, construction of an SRS and guarantee of its correctness need a lot of labours and cost.

A solution of the above problem is reusing existing SRS, but if there exists an SRS database, it is difficult to effectively retrieve a similar SRS from the database. Even if a similar SRS can be detected, some requirements may not be necessary and some should be reviesed, and some should be newly added. In other words, it is not so easy to make a new SRS by reusing an existing SRS.

In this paper, we propose a generation method of SRS parts. In other words, we propose a method of deriving SRS parts from an SRS. Each SRS part represents a functional requirements of a sub-system. In such a way, SRS parts should be highly reusable. We also propose an abstraction method of derived SRS parts. Abstracted SRS parts will be stored into a SRS parts database. In the second process, we will construct an SRS with the SRS parts. In this paper, we focus on the generation of SRS parts.

The paper is organized as follows. The next section will briefly introduce a requirements language named X-JRDL. In section 3, we will describe a generation method of SRS parts. Section 4 presents an experiment for evaluation of the proposed method. In section 5, we will discuss related works. In the last section, we will give concluding remarks.

2 REQUIREMENTS LANGUAGE

We developed requirements model named **Requirements Frame** and a text-base requirements language named **X-JRDL** based on the model (Ohnishi and Agusa, 1991). In this research we adopt X-JRDL as a requirements language, since it is quite easy to transform SRSs with X-JRDL organized differently.

Since X-JRDL aims to specify requirements of file-oriented applications, this language provides 6 noun types (human, function, file, data, control, and device) and 16 concepts including data flow, control

Iyoda Y. and Ohnishi A..
Software Requirements Parts for Construction of Software Requirements Specifications.
DOI: 10.5220/0004479901470153
In *Proceedings of the 8th International Joint Conference on Software Technologies* (ICSOFT-EA-2013), pages 147-153
ISBN: 978-989-8565-68-6
Copyright © 2013 SCITEPRESS (Science and Technology Publications, Lda.)

flow, data creation, file manipulation, data comparison, and structure of data/file/function. The 16 concepts (10 verb type concepts and 6 adjective type concepts) are shown in Table 1.

Table 1: Concepts provided by X-JRDL.

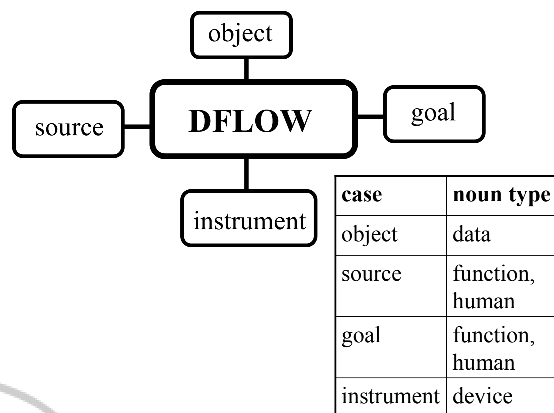
Concept	Meaning
DFLOW	Data flow
CFLOW	Control flow
ANDSUB	And-tree structure
ORSUB	Or-tree structure
GEN	Data creation
RET	Retrieve a record in a file
UPDATE	Update a record in a file
DEL	Delete a record in file
INS	Insert a record in a file
MANIP	File manipulation
EQ, NE, LT, GT, LE, GE	Logical operators

There are several verbs to represent one of these concepts. For example, to specify a concept *data flow*, we can use i*nput, output, print out, display, and send*, and so on. Each concept has its own case structure. The "cases" (Fillmore, 1968) mean concept about agents, objects, goals of the operations (Shank 1977). For example, the *data flow* (DFLOW) concept has objec*t, source, goal*, and *instrument* cases. The object case object corresponds to a data which is transferred from the source case object to the goal case object. So, a noun assigned to the object case should be a data type noun. A noun in the source or goal cases should be either a human or a function type noun. If and only if a human type noun is assigned to source or goal cases, some device type noun should be specified as an instrument case. These are illustrated in Figure 1.

When a user wants to write requirements of another application domain, he may need a verb not categorized into these 16 concepts. In such a case, he can use a new verb if he defines its case structure. In this sense, X-JRDL is extensible.

Since a newly defined verb, its concept, and its case structure can be registered in the verb dictionary, he can use his own verbs as well as provided verbs.

The case structure of each verb enables to detect illegal usages of data and lack of cases. Suppose a requirement sentence, *"A user enters a retrieval command with a terminal."* Since the objective is "*a retrieval command"* that is data type noun, *"enters"* should be categorized into the DFLOW concept. With the case structure of the DFLOW, this sentence will be analyzed as shown in Table 2.



case	noun type
object	data
source	function, human
goal	function, human
instrument	device

Figure 1: Case structure of data flow (DFLOW).

Table 2: Analysis of a requirement sentence *"A user enters a retrieval command with a terminal."*

Concept: DFLOW

object	source	goal	instrument
retrieval command	user	*NOT specified*	terminal

In this sentence the goal case noun is not specified. If indispensable case is not specified, previously specified nouns of the same type become candidates of the omitted case. In this way, a requirement sentence is transformed into an internal representation named **CRD** (Conceptual Requirements Description). CRD is exactly based on the case structures.

X-JRDL provides to use pronouns and omission of nouns. We frequently come across such features in Japanese sentences. The X-JRDL analyzer automatically assigns a concrete word into a pronoun or a lacked case.

The X-JRDL analyzer has a dictionary of nouns, verbs and adjectives. When a requirements definer uses a word which is not appeared in the dictionary, the analyzer guesses a type of new noun and a concept of new verb and adjective with the Requirements Frame (Ohnishi, 1996).

3 GENERATION OF REQUIREMENTS PARTS

3.1 Outline

The outline of a generation method of software parts is illustrated in Figure 2. The first step is deriving a functional requirement from requirements specifications. The second step is generating an SRS part from a functional requirement. The third step is storing SRS parts into SRS parts database. These

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

three steps are shown in Figure 2. The second process is making SRS using SRS parts as shown in Figure 3. In this paper, we focus on the first process only.

Since an existing SRS as is may include unnecessary requirements or customizable requirements, we cannot easily reuse it. In order to improve the reusability, we divide an SRS into single functional requirements and replace concrete nouns in the SRS with abstract nouns. We call a single functional requirement using abstract nouns "a requirements part."

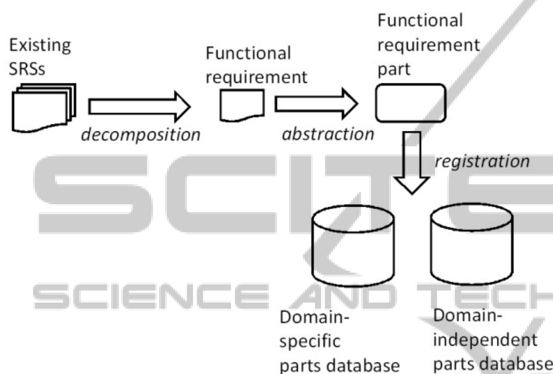


Figure 2: Generation of SRS parts.

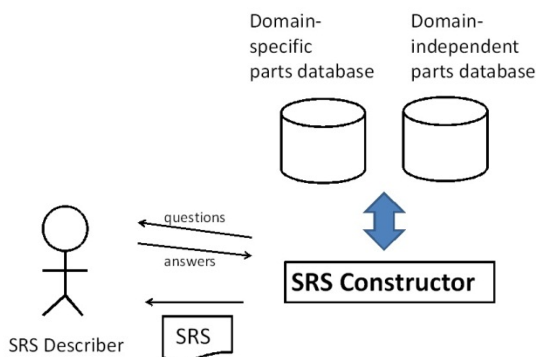


Figure 3: Construction of an SRS.

Requirements parts can be categorized into two types. One is domain dependent parts and the other is domain independent parts. Requirements parts are stored into database in accordance with their types as shown in Figure 2.

3.2 Functional Requirement Parts

A functional requirement part can be generated by decomposing an SRS into functional requirements and by replacing concrete nouns with abstract nouns. We do not replace verbs with more abstract verbs, because we keep a certain abstraction level of SRS by specifying it with a controlled language, X-JRDL.

Both decomposition and abstraction contribute to improve the reusability. For example, an SRS of library system may not be reused for an SRS of CD rental system. However, some functions are similar between two systems. Registration of new books and registration of new CDs are similar each other. Retrieving a book with some keywords is similar with retrieving a CD with some keywords. These functions are included in the SRS of library system. So, decomposing an SRS and deriving a functional requirement from the SRS is a key to improve the reusability.

Books and CDs are different, but they have a common role, that is, rental object. So, we can replace "book" in functions of library system with "rental object" in order to easily reuse the functions. In making an SRS of CD rental system with such functions, we have to replace "rental object" with "CD."

In this way, we can get functional requirement parts by decomposing an SRS into functional requirements and by replacing concrete nouns with more abstract nouns.

Functional requirement parts can be categorized into domain-specific parts and domain-independent parts. Domain-specific parts can be reused for constructing SRS of a certain domain, while domain-independent parts can be reused for constructing SRS of any domain.

3.2.1 Decomposing SRS

Functional requirement parts creator can decompose an SRS into functional requirements by hand. Usually an SRS of a software system consists of several functions. If an SRS is organized by features or sub-systems (IEEE std830 1998), he can easily decompose it into functional requirements.

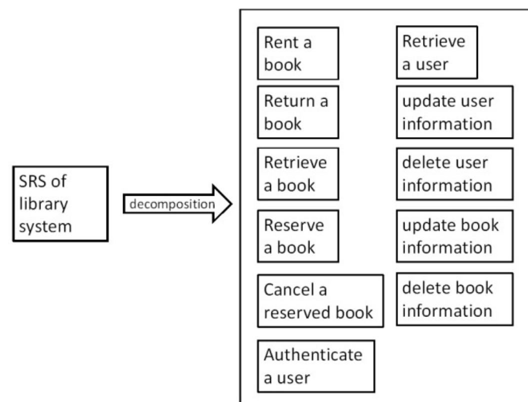


Figure 4: Decomposition of an SRS of library system into 11 functional requirements.

For example, an SRS of library system can be decomposed into eleven functional requirements as shown in Figure 4. A creator of functional requirements parts decompose the SRS by hand.

A functional requirement of "retrieve a book" is shown in Figure 5 and data flow diagram of the same function is shown in Figure 6.

A library retrieval system receives book information from a library information center, searches library data from book database with the book retrieval keywords, receives book id if book data is equal to book retrieval keywords, and passes a copy of a book to a library information center.

Figure 5: A functional requirement of "retrieving a book."

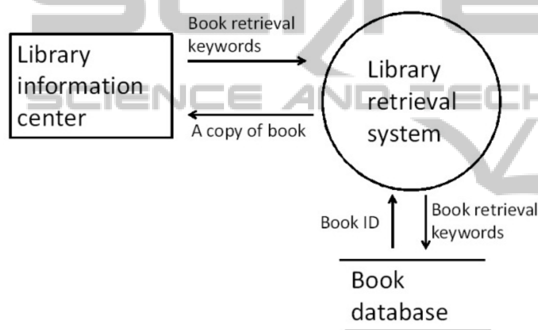


Figure 6: Data flow diagram (DFD) of "retrieving a book."

3.2.2 Replacing Concrete Nouns with Abstract Nouns

The procedure of abstraction of nouns is as follows.

1. Select the most important action. Then concrete nouns corresponding cases of the action can be replaced to the cases name and the action. For example, in case of "retrieving a book," the most important action is "retrieval." "Book" is an object case of retrieval, so "book" can be replaced to "retrieval object." "Library information center" is a user of retrieval, so "library information center" can be replaced to "user."
2. Select a noun. If a selected noun is a compound noun, reduce to essential noun(s). For example, "library retrieval system" is a compound noun, and can be reduced to "retrieval system." If a selected noun is a simple and concrete noun, replace it with abstract noun using thesaurus (Yamaguchi 2006). If a selected noun is abstract, do not replace it. This thesaurus is originally in Japanese, but for readers' convenience we translated in English.

This thesaurus provides 300,000 nouns of different abstraction levels and a hierarchical structure of nouns. By using thesaurus, we can easily get more abstract nouns for a given noun and select an adequate and abstract noun. We can avoid synonyms in abstraction by using a thesaurus.

3. Repeat the above until all of the nouns will be selected.
4. User can modify the results of abstraction if necessary.

Figure 7 shows a part of thesaurus. "Noun" is the most abstract noun in the thesaurus. There exists a structural hierarchy, that is, "noun"-"concrete noun"-"active noun"-"human"-"person (position)"-"staff"-"library staff." If a concrete noun "library staff" appears in a functional requirement, then we can replace it with a more abstract noun, such as "staff."

Figure 8 shows a functional requirement part of retrieving an object. This part is generated by abstracting the functional requirement shown in Figure 5. The underlined nouns are replaced with abstract nouns in Figure 8.

We have developed a supporting system of generating abstracted functional requirements parts with C#. This system supports replacing concrete nouns with abstract nouns. In other words, this system supports to make abstracted functional requirements parts from functional requirements.

1 noun
  2 concrete noun
    3 active noun
      4 human
        318 person (position)
          319 royalty
            320 king
            321 noble
          322 cabinet
          323 chief
          324 vice-chief
          325 director
          326 staff
            assistant, new employee, station employee, policeman, diplomat, officer, professor, library staff, guard, technical staff, teacher, lecturer, detective, ….

Figure 7: A part of thesaurus taken from (Yamaguchi 2006).

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

The number of source code is about 3,000 lines. This system is a 3-person-month product.

A retrieval system receives retrieval object information from a user, searches data from retrieval object database with the retrieval object keywords, receives retrieval object id if retrieval object data is equal to retrieval object keywords, and passes retrieval object to a user.

Figure 8: A functional requirement part of “retrieving an object.” generated from the functional requirement shown in Figure 5.

Figure 9 shows a screenshot of supporting system. This system enables to replace a concrete noun with an abstract noun. Figure 10 shows a part of contents of Figure 9 translated into English.

As shown in the second line of Figure10, a noun “book” is the first target to be abstracted. Using thesaurus, abstract nouns are listed. In this requirement, the most important verb is “rent” and “book” is assigned in the object case of “rent,” so recommended abstract noun “rental object” is also provided in the 17th line of Figure 10. If any candidates of abstract nouns provided by the system are inadequate, user can give his own abstract noun by selecting “addText” in the 15th line of Figure 10.

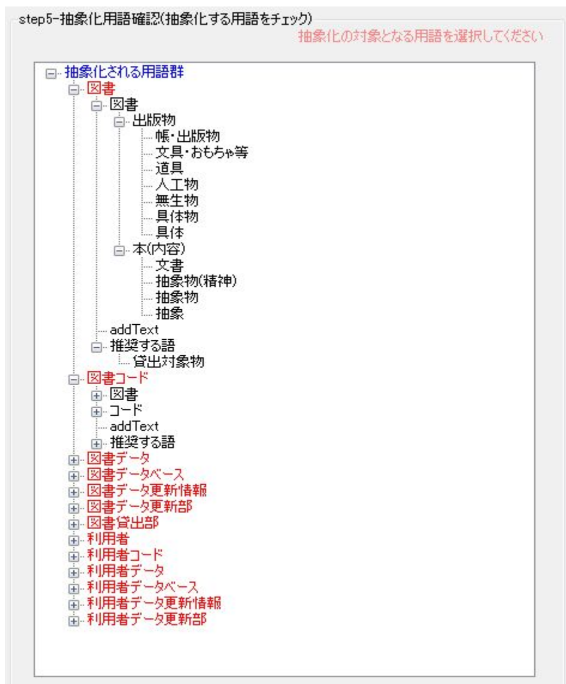


Figure 9: A screenshot of replacing a concrete noun with an abstract noun.

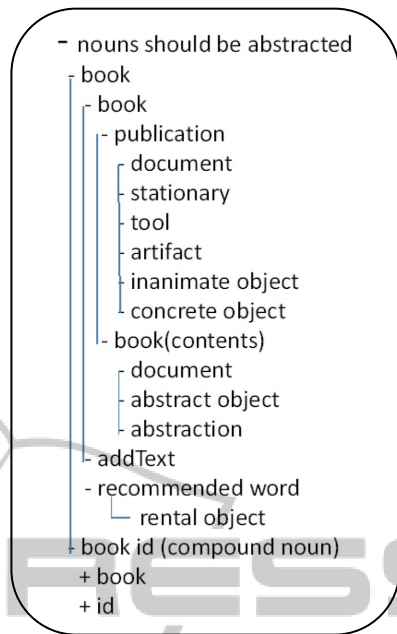


Figure 10: A part of screenshot translated in English.

The next noun to be abstracted is “book id” shown in the 18th line in Figure 10. This noun is a compound noun. A compound noun will be divided into simple nouns and each simple noun will be abstracted. It is difficult to automatically judge which simple noun is important. So, user will reduce the compound noun. In this example, user will judge which simple noun is important. In this case, since “id” is important, just “id” will be selected and abstracted.

4 EXPERIMENT OF EVALUATION OF THE PROPOSED METHOD AND SYSTEM

In order to evaluate the proposed method shown in 3.2.2 and a prototype system based on the method, we performed an experiment for developing functional requirement parts using an SRS of library system. Four subjects (named a, b, c, and d) who are master course students of computer science department can be divided into two groups, say A and B. We give two functional requirements included in the SRS of library system to the subjects. These two functions are “rent a book” and “retrieve a book.”

Subjects a and b of group A replace concrete nouns in the functional requirement of “rent a book”

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

with abstract nouns by hand, while they replace concrete nouns in the functional requirements of “retrieve a book” using prototype system. Subjects c and d of group B replace concrete nouns in the functional requirements of “rent a book” using prototype system, and then replace concrete nouns of the functional requirements of “retrieve a book” by hand.

The number of concrete nouns is 31 and the number of concrete nouns that should be replaced with abstract nouns is 6 in the functional requirement of “rent a book.” The number of concrete nouns that should be replaced is 5 of functional requirement of “retrieve a book.” We prepare correct results in advance and compare the results by subjects and correct ones. Table 3 shows precision and recall values of the abstraction.

Here, the precision is defined as follows.

$$\frac{\text{the number of correctly abstracted nouns by subject}}{\text{the number of abstracted nouns by subject}}$$

The recall is defined as the following equation.

$$\frac{\text{the number of correctly abstracted nouns by subject}}{\text{the number of nouns that should be abstracted.}}$$

For all the subjects, both precision value and recall value in the abstraction with system is greater than the values by hand. This fact means that our method can correctly support the abstraction. In the case of the abstraction by hand by the subject c, recall value is low while precision value is high. This result means that subject c cannot correctly select nouns that should be abstracted. As for the subject d, he cannot abstract functional requirement by hand, but can correctly abstract functional requirement using system.

Table 3: Result of the experiment.

subjects	Precision (abstraction by hand)	Recall (abstraction by hand)	Precision (abstraction with system)	Recall (abstraction with system)
a	0.3	0.3	0.7	0.9
b	0.6	0.6	1	1
c	0.7	0.4	1	1
d	0	0	1	1

As for the subject a using system, the precision value and the recall value is not equal to 1. The reason why the both value is not 1, he modified abstracted noun to the original noun, because he thought that the original noun is abstract enough. To avoid such mistakes, we have to enhance the function of selection of nouns that should be abstracted and provide a function that provide the reason why the abstraction is needed for the selected noun.

5 RELATED WORKS

Buhne et al. proposes a requirements definition method of four different abstraction levels. These levels are software level, function level, system level, and vehicle level. Their proposed method contributes to improve the traceability and the easiness of management of requirements (Buhne et al., 2004). However, their method cannot support to make an abstracted requirement specification, while our method enables to generate an abstracted functional requirement.

Justo proposes a repository for reusing requirements specification (Justo, 1996). His method enables to reuse functional requirements each of which consists of an action and objects of the action. Since these actions and their objects are fully depend on a specific system, quite similar specification can be constructed with the proposed method, while our method improves the reusability by decomposing requirements specification and replacing concrete nouns with more abstract nouns.

Morisaki proposes a software metrics of abstraction level of software document using a thesaurus (Morisaki, 2011). He calculates abstraction level of software document by abstraction level and cardinality of words in the documents. His method focuses on calculating abstraction level of software documents, but does not focus on the reusability of software documents.

Wilson et al. propose a software metrics of several characteristics of requirements specifications and identify statements that need to be improved (Wilson et al., 1997). Their method enables to detect the weakness of an SRS, but does not support to reuse of the SRS.

Peiriyasamy et al. propose a method for structural compatibility of formal specifications written with Z (Periyasamy and Chidambaram, 1997). Here specifications may be different abstraction levels each other. By understanding the behaviors of a specification of high abstraction level software developers can reuse a compatible specification of detailed abstraction level. Their method enables to detect compatible specifications, but does not support to make specifications of different abstraction levels.

There exist several researches on ontology based requirements elicitation (Kaiya and Saeki, 2005), (Li et al., 2007). They reuse ontology for requirements elicitation, but do not reuse software requirements specification.

Our method can be applied to make similar products in the product line engineering (Pohl et al., 2005).

6 CONCLUSIONS

We have established a construction method of software functional requirements parts by decomposing requirements specifications and replacing concrete nouns with abstract ones. In abstraction, we can avoid use of synonyms and replace with adequate nouns by using thesaurus. We have developed a supporting tool of replacing concrete nouns with abstract nouns using thesaurus with C# based on the proposed method. We also evaluate the usefulness and the correctness of the abstraction method and the supporting tool through an experiment.

We do not touch upon construction of SRSs using SRS parts. In constructing a SRS using SRS parts, we have to replace abstract nouns with concrete nouns. As future works, we have to develop a SRS construction method using functional requirement parts and evaluate the method.

ACKNOWLEDGEMENTS

We would like to thank to Mr. Masato Satonaka and Mr. Takahiro Yokoyama, members of our laboratory for their contributions to the research. This research is partly supported by the Grant-in Aid for Scientific Research (c), Japan Society for the Promotion of Science.

REFERENCES

Buhne, S., Halmans, G., Pohl, K., Weber, M., Kleinwechter, H., Wierczoch, T., 2004. "Defining requirements at different levels of abstraction," In proc. 12th IEEE International Requirements Engineering Conference (RE2004), pp.346-347, IEEE Computer Society.

Fillmore C.J., 1968. The Case for Case, Universals in Linguistic Theory, ed. Bach & Harrms, Holy, Richard and Winston Publishing, Chicago.

IEEE std830, 1998. IEEE Recommended Practice for Software Requirements Specification, IEEE std 830-1998, IEEE Computer Society.

Justo, J.L.B., 1996. "A repository to support requirement specifications reuse," In proc. IEEE Information Systems Conference of New Zealand, pp.53-62, IEEE Computer Society.

Kaiya, H., Saeki, M. 2005. "Ontology Based Requirements Analysis: Lightweight Semantic Processing Approach," In proc. Fifth International Conference on Quality Software (QSIC 2005), pp. 19-20.

Li, Z., Wang Z., Yang, Y., Wu, Y., Liu, Y. 2007. "Towards a Multiple Ontology Framework for Requirements Elicitation and Reuse," In proc. 31th Annual International Computer Software and Applications Conference (COMPSAC 2007), pp.189-195, IEEE Computer Society.

Morisaki, S., 2011. "Metrics of the abstraction of software documents using thesaurus (in Japanese)," In proc. 18th Workshop of the Foundation of Software Engineering, pp.213-218, Kindai-Kagaku publishing Co., Japan.

Ohnishi A. and Agusa, K. 1991. "Japanese Software Requirements Definition Based on Requirements Frame Model," Distributed Environments (Ohno, Y. ed.), Springer-Verlag, pp.7-19.

Ohnishi, A., 1996. Software Requirements Specification Database based on Requirements Frame Model, In proc. IEEE 2nd International Conference on Requirements Engineering (ICRE96), pp.221-228, IEEE Computer Society.

Periyasamy, K., Chidambaram, J., 1997. "A method for structural compatibility in software reuse using requirements specification," In proc. IEEE 21th Annual International Computer Software and Applications Conference (COMPSAC'97), pp.426-433, IEEE Computer Society.

Pohl K., Boeckle G., Linden F., 2005. Software Product Line Engineering, Foundations, Principles and Techniques, Springer.

Shank R., 1997. Representation and Understanding of Text, *Machine Intelligence 8,* Ellis Honrood Ltd., Cambridge, pp.575-607.

Wilson, W.M., Rosenberg, L.H., Hyatt, L.E., 1997. "Automated Analysis of Requirements Specifications," In proc. 19th IEEE International Conference on Software Engineering (ICSE 1997), pp.161-171, IEEE Computer Society.

Yamaguchi, T. ed., 2006, "Japanese Thesaurus, CD-ROM version (in Japanese)," Daishukan-shoten, Japan.