Computer-Assisted Analysis and Refinement of Informal Software Requirements Documents

Jacob L. Cybulski
Dept of Information Systems
The University of Melbourne
Parkville, Vic 3052, Australia
Email: j.cybulski@dis.unimelb.edu.au

Karl Reed
School of Comp Science and Comp Engineering
La Trobe University
Bundoora, Vic 3083, Australia
Email: kreed@latcs1.cs.latrobe.edu.au

Abstract

This paper describes RARE (Reuse-Assisted Requirements Elicitation), a method enabling software requirements engineers to process informal software requirements effectively. RARE's object is to assist analysts in transforming requirements expressed in natural language into a comprehensive collection of rigorous specifications that can be used as a starting point in software development. However, unlike other approaches to managing requirements documents, RARE focuses on the application of reuse-intensive methods of dealing with requirements documents, their contents and structure, and the processes involved in the analysis and refinement of requirements texts. The RARE method circumscribes an iterative process of planning, gathering and elaboration, analysis, refinement, adaptation, integration and validation of requirements texts. The paper also describes the operation of IDIOM (Informal Document Interpreter, Organiser and Manager), a requirements management tool that supports the RARE method.

1: Introduction

One of the main problems in software development is the absence of adequate methods and tools for the computer-assisted processing of early software requirements, i.e. their acquisition, analysis, modelling and validation. This situation is commonly blamed on the widespread practice of encoding requirements in natural language [as implied by some early standards, e.g. IEEE-830-1984 - 15]. Such informal methods of requirements specification are frequently criticised by developers and researchers for the potential ambiguity and incompleteness of the resulting requirement specifications [8]. In our view, the methods of dealing with the deficiencies of informal requirements documents are still in the early stage of research [14, pp 76-77]. Considering the difficulties associated with the informal requirements documents, the majority of CASE providers, turned to expressing system and software requirements with the use of diagrams [as recommended by the new SRS standards, e.g. MIL-STD-498 - 20] and mathematical notations [e.g. Z, VDM, Spec or Larch - 27]. Formal specifications are claimed to be more accurate, precise and unambiguous, and are capable of rigorous, and in some cases also automated, analysis of specifications for consistency, completeness, realisability and correctness [14].

Regardless of the benefits offered by the formal specification methods, natural language is still the preferred form of expression by non-technically oriented customers [3]. In view of the need for the effective communication between the customers and developers, it is recognised that writing software requirements in natural language will remain in use for the foreseeable future. During the 1989 Workshop on Requirements Engineering and Rapid Prototyping (held by the U.S. Army Communications-Electronics Command Center), the

participants debated this very issue. In their conclusions [3, p 41], they stated that the specification language should serve the requirements engineering process rather than driving it. They noted that whilst it is desirable to use formal expression of requirements in the later stages of their processing, it is also critical that the actual users define the requirements. As it is not realistic to expect the users to learn complex formalisms, hence, the workshop participants pointed out, customers should use the notation they are most familiar with, i.e. natural language.

In our view, the problem of requirements formality can be stated succinctly as follows :-

> *The specification vehicle should be informal enough to allow an untrained customer to understand what system functions will be delivered upon the system completion, and sufficiently formal to allow a system designer to have unambiguous statement of customer requirements that can be implemented and validated in its widest sense.*

This conflict of informality and rigour in the manipulation of written software requirements specifications attracted and resulted in developmental, methodological and research efforts leading to a number of proposed solutions to the problem, e.g.

- Using executable prototypes to validate user requirements [30];
- Relying on JAD sessions to enhance communication between project stakeholders [32];
- Training clients in the use of formal methods [13];
- Gradual refinement of informal documents into their formal versions [25];
- Generation of formal specifications from documents written in natural language [10, 34];
- Generation of natural language narratives from formal specifications [26];
- Use of "readable" formal specifications [21];
- Synthesis of socio-organisational and formal techniques [29];
- Application of knowledge-based techniques to requirements analysis [e.g. 1, 24]; etc.

There are also many commercially-available requirements management system that can be used to effectively manage a large collection of software requirements, e.g. CASSETS, CORE, Cradle/SEE, RDD, RTM, RequisitePRO, SLATE or Xtie RT [17]. Typically, such systems automate many of the labour-intensive tasks commonly required of the requirements engineers, e.g. scanning text documents, requirements classification, storage and allocation, managing requirements traceability, configuration management, group support, etc. Few of the requirements management tools deal with the problem of processing informal software requirements or free form text documents.

RARE (Reuse-Assisted Requirements Elicitation)[1] offers yet another method of handling informal software requirements, i.e. by active reuse of requirements specifications, their components, analysis and refinement techniques and processes, and other development work-products in the process of requirements elicitation [4]. Due to the complexity of requirements elicitation tasks, RARE also requires the utilisation of a special-purpose requirements management tool - IDIOM (Informal Document Interpreter, Organiser and Manager). IDIOM provides the functions of a typical word-processing system [5], but it also assists developers in the production of quality requirements documents, and the identification, analysis, refinement and reuse of requirements contained in such documents.

Our approach is motivated by the beliefs that (some are now widely accepted) :-

- Software developers can be made more productive by applying reuse techniques at the earliest stages of software development, e.g. requirements engineering.
- Requirements engineering can be made more effective by reusing specification and design components during analysis and refinement of informal requirements.

And of particular significance is the view that :-

- Whenever reusable components are not directly available, then reuse of development processes, which lead to the analysis of similar requirements documents, could also be used to better facilitate the requirements elicitation tasks.

2: Integrating Requirements and Reuse

The reuse of software products and their components in the earliest phases of software development life-cycle (SDLC) has been argued as having a very positive impact on software projects via :-

- better utilization of available resources [18];
- encouragement of systematic reuse across the entire software life-cycle [9];
- improvement in the quality of specifications [16];
- developmental assistance early in the life-cycle [19];

1 Also known as SoDA (Software Designer's Aide), an integral part of a hypertext-based CASE environment developed by Amdahl Australian Intelligent Tools Programme at La Trobe University [HyperCASE - 7].

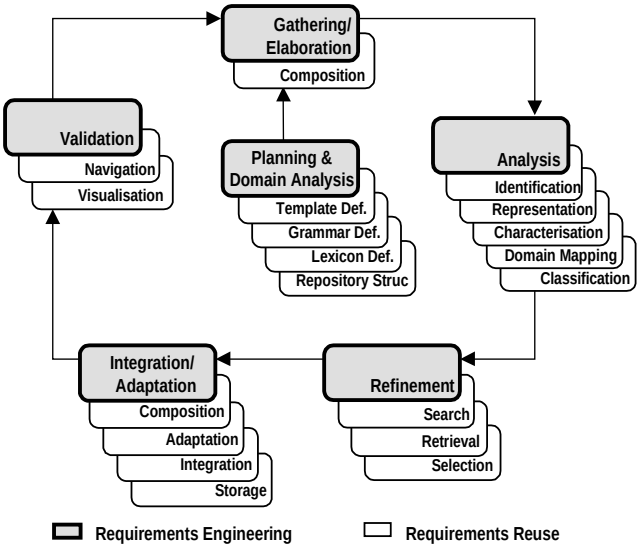


Figure 1: Processing stages in the RARE method

- easier reuse of the other life-cycle products [22], etc.

The potential benefits of early reuse have attracted many research projects that look in depth at the issues of requirements reuse [e.g. 2, 11, 28]. Similarly, RARE IDIOM also places strong emphasis on development of methods and techniques to support reuse of software requirements, specifications and high-level designs. The main difference between our approach and those by others, is in our belief that the processes of requirements engineering and those of software reuse can converge to form a complete and cohesive framework of early developmental activities. Furthermore, this approach offers venue for addressing the issue of cross-domain isomorphism. This fundamental assumption is based on our observation that in the process of requirements engineering, and especially requirements elicitation phase, analysts engage in behaviour that is also typical for software reuse (Cf. Figure 1).

RARE addresses the goals of both requirements engineering (planning, gathering, analysis, refinement, validation, integration and elaboration of requirements) and those of software reuse (planning, analysis, organisation and synthesis of artefacts), relying upon the similarities between these two.

IDIOM actively supports all of the RARE activities with tools that facilitate the tasks associated with software reuse and requirements processing. This is achieved by applying techniques drawn from the areas of natural language processing (restricted natural language, lexicons, grammars and parsers), knowledge representation (semantic networks and domain mapping), information retrieval (faceted classification and affinity analysis), databases (relational DBMS and SQL), hypertext (linking and navigation) and text editing and formatting (templates, forms and style sheets).

3: The RARE IDIOM Process

Our goal is to provide an integrated approach to requirements reuse consisting of reuse-assisted methods (RARE) and tools (IDIOM) provides an effective technological bridge between requirements engineering and software reuse. Let us briefly discuss the main phases of RARE IDIOM processing.

1.1: Planning, Gathering and Elaboration

Creation and the subsequent elaboration of a RARE IDIOM document involves filling in a requirements document template, so that the resulting document conforms to a pre-planned structure, style, form and contents. Text composition in such template-based editing is simplified as the system automatically positions and composes entire document sections, uses default text where appropriate, styles document paragraphs according to pre-defined text attributes, and guides the user as to the expected contents of edited portions of text.

A sample document template conforms largely to the IEEE Standard 830-1984 [15] and defines the following aspects of the documents produced with RARE IDIOM :-

- document structure and parts of its contents,
- text styles for headings, contents and annotations,
- the syntax and semantics of the text still to be inserted into a template.

A fragment of a document produced with the use of this template, the Commodity Paging System requirements specification [31] is shown in Table 1. The fragment shows the selected sections and paragraphs related to the "Observe Schedule Function". The selected text fragment provides an excellent example of the complexity in detecting and tracking requirements similarity and relationships that may exist even within a single requirements document. The standard requirements outline leads to replication of some of the information across the document, first, in a very cursory manner, in the overview and product requirements functions sections and later, in much greater depth, in the functional requirements section. Different aspects of the same requirement are introduced across the entire document (see the second column of the table), at different level of abstraction and formality. Neither the standard the document follows, the requirements structure, or in fact its terminology and syntax can help the reader of the document to identify the relationships between different parts of this document.

R#	Selected requirements	Aspect Introduced
1	**1 INTRODUCTION**	
2	**1.2 CPS SCOPE**	
3	• The schedule set by TSMN for updating commodity information can change at anytime, so the software must accommodate these changes.	**schedule variability**
4	• This software must also limit the number of messages sent during peak hours of the day.	**schedule optimisation**
5	• The software must be flexible enough to handle schedule changes while performing its functions in a cost-effective manner.	**schedule variability** **schedule optimisation**
6	**2 GENERAL DESCRIPTION**	
7	**2.2 PRODUCT PERSPECTIVE**	
8	• The software will define an access schedule for both information services, TSMN and the MicroPage.	**scheduling services**
9	• The schedule will determine when TSMN has made an update […].	**schedule objectives**
10	• [The software will] define a schedule for sending and receiving commodity information.	**schedule contents** **schedule events**
11	**2.3 PRODUCT REQUIREMENTS FUNCTIONS**	
12	**2.3.1 SET SCHEDULE FUNCTION**	
13	• [...] this software shall reconfigure the system in the event of a schedule change by either of the two services.	**schedule variability**
14	**2.3.2 OBSERVE SCHEDULE FUNCTION**	
15	• The observe schedule function will control when the system sends or receives any information.	**schedule events** **following schedule**
16	• This software shall use the schedule set up by the set schedule function and the system's internal clock to determine the appropriate time to obtain commodity information from TSMN or to send updates to MicroPage.	**following schedule** **schedule use**
17	• This function controls the real-time activities of the system.	**following schedule**
18	**3. SPECIFIC REQUIREMENTS**	
19	**3.2 FUNCTIONAL REQUIREMENTS**	
21	**3.2.1 SET SCHEDULE FUNCTION**	
22	• [Set schedule function sends the schedule to] observe schedule function.	**following schedule**
23	**3.2.2 OBSERVE SCHEDULE FUNCTION**	**function name**
24	• The observe schedule function uses the schedule and inputs from the system clock to determine the best time to send and receive information.	**scheduling events**
25	• The TSMN update schedule will be strictly followed.	**following schedule**
26	• As soon as a commodity update is available, this function will activate the receive commodity information function to obtain the update.	**receiving events**
27	• This function will also determine the appropriate time to send the update to the MicroPage by using their peak and off-peak hours as a reference.	**following schedule** **sending events**
28	• The goal is to limit the number of unnecessary updates sent to MicroPage to reduce costs.	**schedule optimisation** **schedule objectives**
29	**3.2.5 FORMAT PAGER MESSAGES FUNCTION**	
30	• [Format pager messages function sends Number_of_Messages to] observe schedule function.	**following schedule**
31	**3.4 PERFORMANCE REQUIREMENTS**	
32	• Observe schedule function: This function will detect a commodity update by TSMN within 250 milliseconds.	**following schedule** **scheduling constraints**
33	**APPENDIX C.**	
34	• Schedule =	**schedule data**
35	*the access schedule for both TSMN and MicroPage, the schedule defines when commodity updates are available and the best time to access MicroPage*	**schedule events** **receiving information** **sending information**
36	1 { @Schedule_Type + [Commodity_Update_Time | MicroPage_Access_Definition] } 999	

Table 1: Tracking software requirements (note that the highlighted area denotes the main focus of this example, i.e. all remaining paragraphs are related to this section)

Traditionally, the task of requirements document analysis is left in its entirety up to the skill of a requirements engineer, his or her experience with the elicitation techniques, knowledge of the problem domain, and of course the years of practical experience in dealing with problems of this kind. Nevertheless, even the most experienced analysts may have insurmountable problems handling very large and complex requirements

Paragraph Id			Paragraph Text	Domain	Key 1	Key 2	Key 3	Key 4	Key 5
128	3.2.1	0	SET SCHEDULE FUNCTION	cps	define	function	set	schedule	
129	3.2.1	1	The set schedule function defines the access times for both TSMN and MicroPage.	cps	define	access	time	TSMN	MicroPage
130	3.2.1	2	The access schedule for TSMN and MicroPage will be read from an operator and retained by CPS.	cps	read	schedule	access	operator	retain
131	3.2.1	3	The access schedule will contain times in which TSMN updates commodity information and MicroPage's peak/off-peak hours.	cps	contain	schedule	access	TSMN	update
132	3.2.1.1	0	INPUTS	cps	meta	input			
133	3.2.1.1	1	Schedule - Source: Operator	cps	input	schedule	operator		
134	3.2.1.2	0	PROCESSING REQUIREMENTS	cps	meta	process			
135	3.2.1.1	1	WHILE not end of schedule input DO	cps	repeat	input	schedule	end	
136	3.2.1.1	2	READ schedule record	cps	read	schedule	type		
137	3.2.1.1	3	INSERT schedule_type into SCHEDULE	cps	insert	schedule	type		
138	3.2.1.1	4	CASE schedule_type of "Stock Market Schedule"	cps	select	schedule	type	stock market	
139	3.2.1.1	5	INSERT commodity_update_time into SCHEDULE "MicroPage Schedule"	cps	insert	commodity	update	time	MicroPage
140	3.2.1.1	6	INSERT peak_hours into SCHEDULE	cps	insert	peak	hour	schedule	
141	3.2.1.1	7	INSERT off_peak-hours into SCHEDULE	cps	insert	off-peak	hour	schedule	
142	3.2.1.1	8	END CASE	cps	meta	end			
143	3.2.1.1	9	CLOSE the schedule file.	cps	close	schedule	file		
144	3.2.1.3	0	OUTPUTS	cps	meta	output			
145	3.2.1.3	1	Schedule - Destination: Observe Schedule Function	cps	output	schedule	observe	schedule	function

Table 2: Requirement characterisation

specifications. RARE IDIOM, however, can assist the requirements engineers by providing them with the structure to the analysis task and by guiding them through the analysis process. In addition, the analytic capabilities of conventional compiler-writing and information retrieval are brought to bare to automate parts of this process.

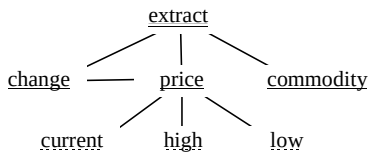
1.2: Analysis and Refinement of Requirements

RARE IDIOM interprets requirements texts with a simple DCG parser [12]. The parser verifies the document syntax and semantics, and searches for textual references to reusable specifications described in some other, referent, documents. In the process, the system highlights ill-structured phrases and allows their correction. At this early point in the analysis process, requirements engineers may be forced to rephrase certain requirements to unify the terminology across the entire document, to split composite sentences into simpler expression of individual requirements, or to move certain requirements to different sections. Reformulation of the document may be carried out until such a time that the system is satisfied that all requirements have been entered in appropriate sections of the document and have been expressed in a simple language. As the result of this analysis RARE IDIOM completes the following tasks:

1. It identifies in the document references to all known domain concepts, individual requirements and relationships between them. E.g.

   > This ~~software~~ will extract the current price, high, low, and change in price for each of the commodities.

2. The identified textual references are subsequently represented in a semantic network [33]. E.g.



3. The semantic network is then processed to nominate a number of keywords (in the order of their importance to the requirement) that best characterise the requirements text. E.g.

Keyword	Weights
extract	50%
price	20%
change	10%
commodity	10%

A collection of characteristic keywords provides an index searchable by domain terms (Cf. Table 2).

no	artefact	description	function	data	domain	method	environ	location	time	type
1	array	Defines an array indexed by numbers	aggregate	array	system	step	machine	memory	very fast	data
9	collection length	Returns the number of collection items	count	collection	system	traverse	machine	device	any	function
10	delete from collection	Deletes an item in a collection	delete	collection	system	sequence	machine	device	any	function
22	copy	Copies an object	copy	data	system	traverse	machine	device	any	function
23	display	Displays any information to the user	display	data	system	structure	user	monitor	fast	function
24	window	Window for reading or writing info	interact	data	system	structure	user	monitor	fast	module
95	message	Information displayed to the user	display	text	system	structure	user	monitor	fast	module
126	get from collection	Retrieves an item from a collection	get	collection	system	step	machine	device	fast	function
128	req11: obtain commodity information TSMN wire	Throughout the day, the software will obtain commodity information from TSMN wire service.	receive	record	cps	any	server	network	any	requirement
129	req13: transform price change message pager	It will transform price changes into messages understandable to the pagers.	format	number	cps	transform	change	monitor	fast	requirement

Table 3: Artefacts and requirements classification

4. The nominated problem domain keywords are mapped with the use of a domain-mapping thesaurus into a number of terms drawn from the solution domain, e.g.

Word	Facet	Facet Value
extract	function	get
price	data	number
change	function	change
change	function	exchange
change	environment	change
commodity	data	record

5. The faceted classification scheme [23] is then used to generate an affinity query [6] against a repository of reusable software artefacts (Cf. Table 3) to retrieve and prioritise matching design artefacts and requirements (Cf. Table 4). In the process, the analyst may expand or correct the generated query, e.g.

Facet	Value
function	get
data	number
domain	cps
method	any
environment	change
location	hard disk
time	acceptable
type	function

6. Multiple interpretations of concepts appearing in requirements documents must be resolved and refined before their complete formalisation. The user is allowed to browse through the text of requirements, analysing the presented lists of possible requirements interpretations, and selecting the system suggestions which best reflect his or her needs (Cf. Table 4). Refinement candidates are then selected from amongst reusable design and specification artefacts, that had been previously entered, analysed and refined, hence classified and indexed in the process as well. E.g.

Artefact	Artefact	Affinity
126	get from collection	1.0000
13	next in collection	0.8500
14	previous in collection	0.8500
15	first in collection	0.8500
16	last in collection	0.8500
40	file path	0.8302
17	add to collection	0.8260

7. Note also that classified and refined requirements statements are subsequently entered into an artefact repository for future reference, retrieval and navigation.

1.3: Integration/Adaptation and Validation

The process of requirements refinement results in a collection of hypertext links from informal requirements into components of reusable specifications and designs to facilitate traceability of the analysed results back to the source requirements. To complete the process of requirements formalisation, the analyst must derive a more struc-

tured and rigorous document from its informal version. The process takes advantage of reuse information contained in the links, to aid the transformation of informal requirements into fragments of more formal specifications that are the basis of reuse. Specifications could further be edited, corrected and completed by the analyst to reflect the intended meaning of the informal requirements. Finally, the specification fragments are integrated into a continuos specification document.

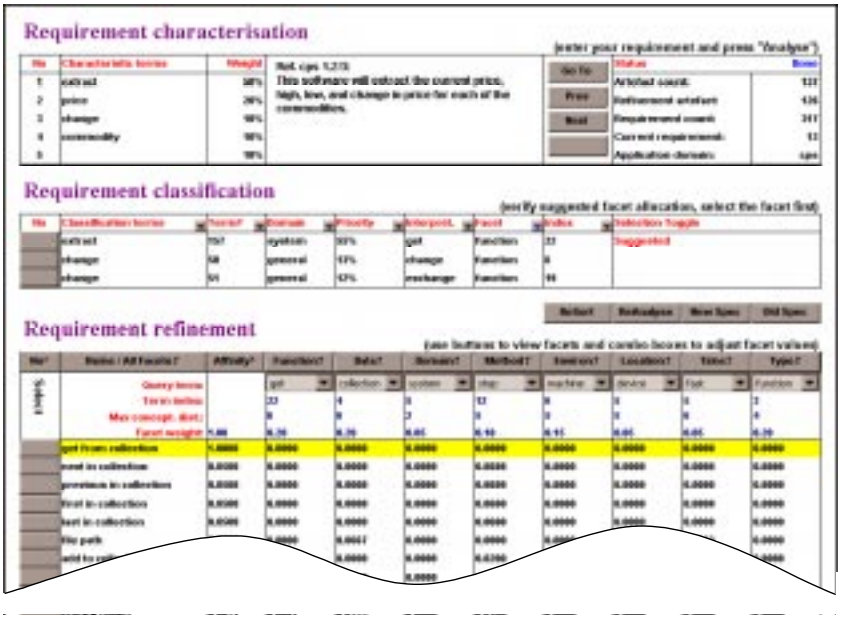


Table 4: Affinity analysis and requirements refinement (actual screen) - the highlighted line represents the selected refinement

Once the inter- and intra-document links have been established, RARE IDIOM allows easy navigation between templates, documents, and their components, and under HyperCASE [7], to other project artefacts. Apart from simple document reading and browsing, navigation may also assist the user in validating current interpretation and formalisation of requirements texts. This is achieved by justifying RARE IDIOM decisions leading to certain text interpretations which show the user all of the lexical, grammatical and semantic cues that lead to the formulation of requirements specification.

4: Summary and Conclusions

This paper investigated the possibility of improving the process of requirements elicitation from informal texts by identifying and elaborating references to reusable specifications. The paper also described the reuse-assisted requirements elicitation process (RARE) and a tool that is capable of facilitating this process (IDIOM).

When adopted, RARE IDIOM can help requirements engineers in the laborious task of analysing and modelling knowledge contained in volumes of plain-English software requirements documents. The RARE IDIOM methods are based on a conjecture that early identification of reuse potential in software development could significantly enhance the effectiveness and efficiency of the development tasks. In the pursuit of this goal, we proposed and implemented a scheme that combines natural language processing with the faceted classification of software requirements. Consequently, RARE is able to interpret, classify, analyse and refine informal requirements texts.

We are currently conducting a series of experiments aimed at determining the effectiveness of professional and novice developers equipped with RARE IDIOM. However, as the group under study is quite large (over 100 subjects) the comprehensive results are not yet available. The preliminary (and at this stage informal) results obtained so far are promising.

REFERENCES

1. Borgida, A., S. Greenspan, and J. Mylopoulos (1985): "Knowledge representation as the basis for requirements specifications." *IEEE Computer*: p. 82-90.

2. Castano, S. and V. De Antonellis (1994): "The F3 Reuse Environment for Requirements Engineering." *ACM SIGSOFT Software Engineering Notes.* ***19***(3): p. 62-65.

3. Cmu/Sei (1991): *Requirement Engineering and Analysis*, Technical Report CMU/SEI-91-TR-30, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania.

4. Cybulski, J.L. (1996): *Introduction to software reuse*, Research Report 96/4, The University of Melbourne, Department of Information Systems: Melbourne.

5. Cybulski, J.L. (1997): "Reuse of early life-cycle artefacts: reusing requirements with a word processor?" in *WISR-8*. Ohio State University, Columbus, Ohio, USA, p. Cybulski-1-7.

6. Cybulski, J.L., R.D.B. Neal, A. Kram, and J.C. Allen (1998): "Reuse of Early Life-Cycle Artifacts: Workproducts, Methods and Tools." *Annals of Software Engineering.* ***5***, To appear.

7. Cybulski, J.L. and K. Reed (1992): "A hypertext-based software engineering environment." *IEEE Software.* ***9***(2): p. 62-68.

8. Davis, A.M. (1990): *Software Requirements: Analysis and Specification*. Englewood Cliffs, New Jersey: Prentice Hall.

9. Frakes, W. and S. Isoda (1994): "Sucess factors of systematic reuse." *IEEE Software*. ***11***(5): p. 15-19.

10. Fuchs, N.E., H.F. Hofmann, and R. Schwitter (1994): *Specifying Logic Programs in Controlled Natural Language*, 94.17, Department of Computer Science, University of Zurich.

11. Fugini, M.G., O. Niestrasz, and B. Pernici (1992): "Application development through reuse: the Ithaca tools environment." *ACM SIGOIS Bulletin*. ***13***(2): p. 38-47.

12. Gazdar, G. and C. Mellish (1989): *Natural Language Processing in PROLOG*. Wokingham, England: Addison-Wesley Pub. Co.

13. Hall, A. (1990): "Seven Myths of Formal Methods." *IEEE Software*: p. 11-19.

14. Hsia, P., A. Davis, and D. Kung (1993): "Status Report: Requirements Engineering." *IEEE Software*: p. 75-79.

15. IEEE (1984): *Guide to Software Requirement Specifications*, Std 830-1984, The Institute of Electrical and Electronics Engineers, Inc.

16. Johnson, W.L. and D.R. Harris (1991): "Sharing and reuse of requirements knowledge." in *6th Annual Knowledge-Based Software Engineering Conference*. Syracuse, New York, USA: IEEE Computer Society Press, p. 57-66.

17. Jones, D.A., D.M. York, J.F. Nallon, J. Simpson, and I.R.W. Group (1995): "Factors Influencing Requirement Management Toolset Selection." in *Fifth Annual Symposium of the National Council on Systems Engineering*: International Council on Systems Engineering, url: http://www.incose.org/lib/rmtools.html.

18. Kang, K.C., S. Cohen, R. Holibaugh, J. Perry, and A.S. Peterson (1992): *A Reuse-Based Software Development Methodology*, Technical Report CMU/SEI-92-SR-4, Software Engineering Institute.

19. Maiden, N. and A. Sutcliffe (1989): "The abuse or re-use: why cognitive aspects of software re-usability are important," in *Software Re-use, Ultecht 1989*, L. Dusink and P. Hall, Editors. Springer-Verlag: London, U.K. p. 109-113.

20. MIL-STD-498 (1996): *MIL-STD-498 Overview and Tailoring Guidebook*, Report MIL-STD-498, Joint Logistics Commanders, Joint Policy Coordinating Group on Computer Resources Management: Washington, DC.

21. Parnas, D.L. and J. Madey (1993): "Functional documentation for computer systeme engineering." in *15th Int. Conf. on Software Engineering*. Baltimore, Tutorial Notes.

22. Poulin, J. (1993): "Integrated support for software reuse in computer-aided software engineering (CASE)." *ACM SIGSOFT Software Engineering Notes.* ***18***(4): p. 75-82.

23. Prieto-Diaz, R. (1991): "Implementing faceted classification for software reuse." *Communications of ACM*. ***34***(5): p. 88-97.

24. Reubenstein, H.B. and R.C. Waters (1989): "The requirements apprentice: An initial scenario." in *5th International Workshop on Software Specifications and Design*: IEEE Computer Society Press, p. 211-218.

25. Saeki, M., H. Horai, and H. Enomoto (1989): "Software development process from natural language specifications." in *11th International Conference on Software Egineering*. Pittsburgh, Pennsylvania: IEEE Computer Press, p. 64-73.

26. Salek, A., P.G. Sorenson, J.P. Tremblay, and J.M. Punshon (1994): "The REVIEW system: From formal specifications to natural language." in *The First International Conference on Requirements Engineering*. Colorado Springs, Colorado: IEEE Computer Society Press, p. 220-229.

27. Sannella, D. (1993): "A survey of formal software development methods," in *Software Engineering: A European Perspective*, R.H. Thayer and A.D. McGettrick, Editors. IEEE Computer Society Press: Los Alamitos, California. p. 281-297.

28. Sutcliffe, A. and N. Maiden (1990): "Assisting requirements analysis through specification reuse." in *Workshop on Next Generation CASE-Tools*. Noordwijkerhout, Netherlands.

29. Swatman, P.A., D.C. Fowler, and M.C.Y. Gan (1991): "Extending the useful application domain for formal methods," in *Z User Workshop*. Springer-Verlag: New York, NY. p. 125-144.

30. Vonk, R. (1989): *Prototyping: The Effective Use of CASE Technology*. Englewood Cliffs, N.J.: Prentice-Hall Int.

31. Wilkinson, R.T. (1990): "Software requirements specification for the Commodity Paging System," in *Standards, Guidelines, and Examples on System and Software Requirements Engineering*, M. Dorfman and R.H. Thayer, Editors. IEEE Computer Society Press: Los Alamitos, California. p. 457-477.

32. Wood, J. and D. Silver (1989): *Joint Application Design*. New York: John Wiley & Sons.

33. Woods, W.A. (1991): "Understanding subsumption and taxonomy: A framework for progress," in *Principles of Semantic Networks: Explorations in the Representation of Knowledge*, J.F. Sowa, Editor. Morgan Kaufmann Pub., Inc.: San Mateo, California. p. 45-94.

34. Yonezaki, N. (1989): "Natural language interface for requirements specification," in *Japanese Perspectives in Software Engineering*, Y. Matsumoto and Y. Ohno, Editors. Addison-Wesley Publishing Company: Singapore. p. 41-76.