RARE IDIOM: An Overview

Jacob L. Cybulski

Department of Computer Science and Computer Engineering
La Trobe University, Bundoora, Vic. 3083, Australia
Phone: +613 479 1270, Fax: +613 470 4915
Email: j.cybulski@dis.unimelb.edu.au

Abstract

One of the main problems in software development is insufficiency of methods and techniques currently employed in the process of requirements engineering, i.e. their acquisition, modeling and validation. The early approaches to requirements capture utilised natural language description of clients needs. Such techniques, although preferred by non-technically oriented customers, are widely criticised by developers for the potential ambiguity and incompleteness of the resulting requirement documents. Thus, majority of recently proposed schemes of requirement acquisition favour formal (i.e. mathematical) or semi-formal (i.e. diagrammatic) methods of requirements specification, both of which greatly simplify requirements verification in terms of their internal completeness and consistency but which at the same time may reduce the validity of captured requirements due to the customers inability to fully comprehend the mathematical or diagrammatical notation.

This paper addresses the problem of disparate needs of customer and developer communities and offers a computer-assisted method of arriving at a complete and cohesive formal specification of user requirements from the informal, potentially ambiguous and incomplete requirements texts. The proposed method, RARE (Reuse-Assisted Requirements Elicitation), relies on the iterative process of requirements gathering into a hypertext template, linguistic analysis of the collected natural-language requirements, followed by the knowledge-based refinement of the informal statements into a set of reusable specification artefacts, the subsequent integration and adaptation of reuse structures into partial specifications, and further elaboration of requirements to the complete satisfaction of all the software stakeholders. The paper describes IDIOM (Informal Document Interpreter, Organizer and Manager), the system aiding the RARE method of requirements elicitation.

Background

Development of large software products often starts with requirements engineering, a phase in the process of software construction which attempts to obtain a complete, consistent and unambiguous specification of requirements shared by different stakeholder groups. The main objective of this activity is to collect, formalise, analyse and validate customer requirements. A documented collection of such requirements, the requirements specification document, is a statement of consensus between the suppliers and customers of a software system, it constitutes the basis for the subsequent software design, and provides a reference point for any future validation of the constructed software. As the quality of a specification document has a significant impact on the quality of the final

product, so it is important that all information contained in this specification is unambiguous, complete, verifiable, consistent, modifiable, traceable, useable, necessary, and prioritised. [40, 10, 18].

It is practically impossible to construct a good quality requirements specification document in a single, monolithic step, so the process of requirements engineering is normally conducted iteratively and can be structured into several phases, e.g. elaboration of needs and objectives, requirements acquisition and modelling, generation and evaluation of alternatives, and finally requirements validation [36]. Elicitation of information from customers, as manifested in the first two phases of the requirements engineering cycle, is in itself a complicated process, and may be further broken up into a number of distinct steps, i.e. identification of information sources, information gathering, rationalisation and prioritisation of collected information, and integration of requirements with the aim to combine different view points [18].

Customer requirements are normally communicated to developers via informal channels, e.g. technical notes, minutes of meetings, statements of work, requests for proposals, operational concept documents, interviews with customers and users, manuals, or technological surveys [77]. Requirements, thus, come in a variety of media types, to include free text, graphics, image, video and animation, speech, sound, or other sign systems [63]. They are rarely in a computer-readable form, and even when such a computer representation is available, its structure and rigour is hardly ever yielding to automatic analysis, formalisation and integration. At the same time, the final specification document is commonly called for to be in a precise, technical and symbolic notation. This dichotomy between user requirements and their specifications is illustrated in Figure 1 [48].

This conflict between informality and rigour in the specification of software requirements becomes the main theme of a number of research projects which resulted in several solutions to the problem.

- The most common solution is to rely on the skill of requirements engineers to produce a mapping from their own informal notes and observations into a more rigorous specification document. Customer validation of specifications can be subsequently facilitated by producing prototypes or concept demonstrators [76], walkthroughs can be used to achieve a consensus on key requirements [37], or the formal specifications can be recast back into a form readily understood by the clients, e.g. into natural language [45, 25, 73].
- Alternatively, all software stakeholders can be involved in a joint development of specifications starting from a very sketchy and informal project proposals down to

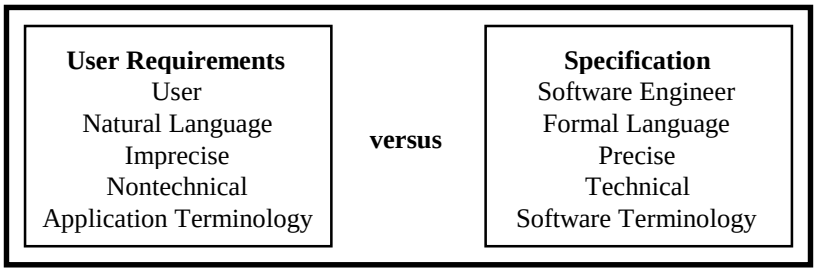

Figure 1 Requirements vs specification dichotomy

their rigid and formalised specifications, e.g. as in JAD [78]. This approach forces all participants of the requirements engineering process to find a common line of communication, it provides an opportunity of instantaneous feedback, query and correction.

- Another approach is to record all informal requirements which can later be elaborated and formalised into a more rigid specification document [72]. The refinement process may also preserve the links between informal and formal concepts for traceability sake, thus, creating a complete, cohesive and hyper-navigable requirements specification system [23, 77, 50].
- Yet another method is to state all requirements in a controlled natural language, so that all of the requirements statements could be readily translated into formal, possibly even executable, statements of software specification [65, 28, 79].

Regardless of the approach to the acquisition and formalisation of requirements, the human factor during information elicitation makes the whole process incredibly difficult and time consuming. The complexity and labour intensity can somewhat be reduced by specially devised methods and techniques (cf. Figure 2), for instance :-

- domain analysis [68] and enterprise modelling [39], aiming at the construction of a complete model of a domain or enterprise, allow subsequent application development to rely on the pre-classified and formalised domain or enterprise concepts and their relationship, thus, reducing the possibility of mis-communication, incompleteness and inconsistency;
- systematic reuse of requirements and domain-related information in a wide-spectrum artefact reuse [56] can accomplish several important upstream reuse objectives, e.g. selection of domain objects and associated software components during requirements specification, ablity to critique and analyse user requirements, detection of requirements incompleteness and filling in missing requirements details, and selection of an appropriate refinement of requirements into more detailed specifications and designs;

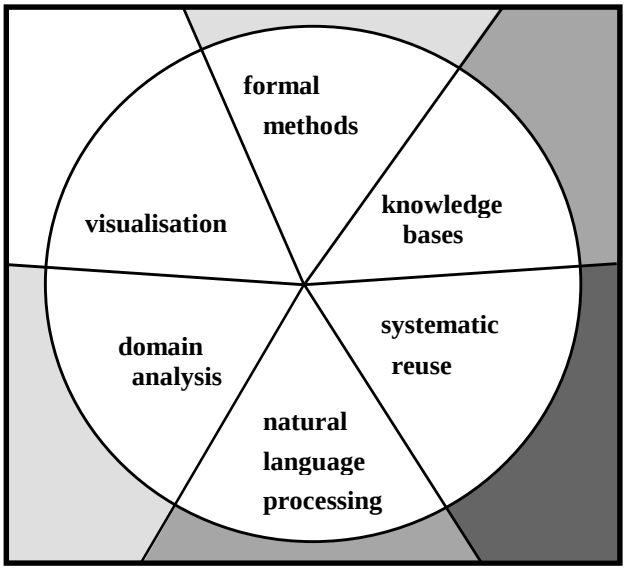

Figure 2 Reducing labour and complexity in requirements elicitation[1]

- employment of communication agents monitoring end users working with prototype systems and reporting mismatches between developer's expectations and the system's actual usage [31];
- visualisation tools, which model and represent a customer problem in a graphical form [64, 3, 15, 49], provide their users with the environment in which the process of identifying and describing the needs and concerns could be achieved by means of direct and intuitive manipulation of visually familiar concepts and objects [57];
- some of the software providing assistance in requirements and design employ knowledge acquisition, representation, and expert systems techniques to extract customer's knowledge into a sophisticated knowledge-base which could subsequently be reasoned about, processed, evaluated and formalised into a set of deliverable requirements specifications [70, 8, 46, 53, 69];
- the texts obtained in the process of requirements elicitation can be analysed, structured and interpreted with the use of natural language processing tools, thus, facilitating easier identification, indexing, retrieval and formalisation of requirements concepts [5, 17, 2, 79, 22, 28]; and finally,
- users can be trained in the use of a formal notation, so that they could be directly involved in the construction, validation and verification of formal specifications, predicting the behavioural consequences of the specification, or taking advantage of specification animation as available in a number of systems [34, 9].

The work reported in this paper combines several different approaches to effective and efficient bridging of the formal and informal approaches to expressing software requirements, i.e. it favours gradual elaboration and formalisation of informal requirements texts into reusable specifications with hypertext traceability links. This

1 Note that heavily shaded areas denote the focus of this research.

process is further assisted with knowledge-based reasoning and natural language processing techniques.

Rare Idiom Concepts and Motivation

The method and the system described in this paper, RARE IDIOM (Reuse-Assisted Requirements Elicitation with Informal Document Interpreter, Organiser and Manager), finds its roots in the SODA project [21], an integral part of HyperCASE [20] - a CASE environment developed by Amdahl Australian Intelligent Tools Programme. SODA's primary aim was to integrate the techniques of natural language processing, knowledge based systems and hypertext to support the process of interpreting plain English texts in search of textual references to reusable specification artefacts, the basis for the construction of navigable paths between related software documents and their components. RARE IDIOM extends this concept by providing a tool-assisted method of iterative elicitation of functional requirements and their subsequent formalisation into a set of LARCH LSL specifications [33].

The work instigated in SODA and continued in RARE IDIOM was motivated by the apparent problems in requirements elicitation, the benefits and pitfalls of formal requirements models, and the possibility of improving the process of requirements acquisition by reusing software requirements and specifications.

Requirements Elicitation. The effectiveness and efficiency of software requirements elicitation is of pivotal importance to the process of requirements engineering, which determines the success or failure of the entire development cycle. The most important issues in requirements elicitation are communication, traceability and process understanding [19].

- Communication is a major source of difficulty, because information elicitation is primarily a process of human communication. It was reported by Christel and Kang [18] that 56% of errors in installed systems were due to poor communication between various stakeholders and that these were the most expensive errors to correct using up to 82% of development time.
- Traceability is also riddled with problems because cost, schedule, and personnel factors force frequent revisions of systems in operation and those in development. It was also reported that majority of currently used techniques for requirements capture and their representation focus on information modelling using formal notations, thus, completely ignoring the form the requirements were initially communicated in. In the process the original requirements are obscured and the full traceability is sacrificed [77]. This inability to locate and access the source of original requirements is one of the most commonly cited problems of requirements engineering [32].
- Process understanding can cause elicitation problems when the process is not defined adequately and the interrelationships among processes are loose, informal, and not well understood. Streamlining and integrating elicitation steps can be accomplished by building tools to improve the capture and organisation of information in meetings, assisting in the negotiation process, and support in linking, traceability, and evolution of requirements. Few of the commercially available software tools listed in the comprehensive report by Rock-Evans [71]

Informal	Formal
Requirements elicitation from user's informal descriptions in natural language.	Specification of these requirement in an appropriate formal notation by developers.
Client's incomplete and inconsistent perception of his/her needs.	Developer's exigency to close the specification in a harmonious, rigid and comprehensive form.
Changing views of client's requirements (and changing requirements as well).	Elevated effort and the cost of maintaining highly cohesive and complete formal representation of these requirements.
Client's low-level of computer literacy and the resulting preference for the use of application domain concepts.	Developer's unfamiliarity with business terminology and his/her need to express the requirements in computer terms.
Multi-faceted and contradictory views of different software stakeholders.	Limitation of formal languages to express only one such view in an integral and complete fashion.
Software validation against client's needs expressed informally.	Verification against formal semantics of a specification notation.

Table 1 Formal vs informal approach to requirements engineering

address the issue of requirements elicitation at all, none is mentioned to be able to deal with aspects of notations capable of representing early, thus informal, software requirements.

This research was, therefore, undertaken to address the issue of requirements communication, traceability and process understanding by means of tools supporting informal requirements texts.

Requirements Formality. Some of the above-mentioned problems and complexities are commonly blamed on the use of "informal" notations to record software requirements. However, in spite of recent achievements in the area of formal specification methods and CASE, the communication barriers still separate developers using precise notation and users preferring the use of natural language. The two viewpoints clearly provide a bone of contention between a number of factors (cf. Table 1).

It becomes increasingly evident that formal specification languages are not performing well in the early stages of requirements acquisition and that they frequently do not scale up when applied to large, hard and real-life problems :-

- As the early stages of requirements acquisition are the subject of much negotiation between software stakeholders, thus, initial user requirements must necessarily be informal, i.e. vague, ambiguous and incomplete. Formal methods can be

successful when used after the concensus on the requirements has been reached [18].

- While it is desirable to specify requirements formally at some stage of requirements engineering, it is also critical that the actual users define these requirements, and they cannot be expected to learn and use complex formalisms effectively [19].
- More research should be conducted to provide requirements engineers with some tools and guiding principles of improving natural-language documents without the introduction of unnecessarily formal models [38].
- Natural language is not only the most familiar form of communication, but it is also the form which allows the implicit information to be omitted and the reader's attention to be directed to the explicitly expressed, important, components of the specification, thus, leading to software requirements specifications which are more concise, less complex and thus cheaper in maintenance than those written in a formal notation [5].[2]

At the same time, despite certain advantages in the use of natural language to capture early requirements, the farther requirements engineering phases evidently suffer from the use of natural language, due to the following factors [60] :-

- noise due to redundancy or author's remorse,
- silence due to omission,
- over-specification due to solution details,
- contradiction due to incompatibility,
- ambiguity due to multiple interpretations,
- forward reference to concepts introduced later,
- wishful thinking due to the lack of realistic validation, etc.

As both informal and formal approaches to requirements engineering can bring certain advantages to the overall process, thus, it seems logical that they both should be integrated into one cohesive method as advocated by Balzer [5, 4], Abbott [1], Barstow [6], Johnson [44], Hsia [38], PROTEUS project [12], and others.

Requirements Reuse. Effective reuse of system and software requirements, their specifications, and the processes which lead to their acquisition was seriously neglected in the conventional development methodologies. Recent studies, however, show that reuse in the early stages of software development results in the significant improvement of the overall process and the quality of the final product. Thus, it has been noted that :-

- Conventional requirements analysis techniques do not provide methods of reusing the results of previous analyses. Analysts are, thus, assumed to start with a blank slate for each new problem, and as a result, they are doomed to recreate previous mistakes [47].
- Many organisations aim at the introduction of systematic software reuse concerned primarily with the reuse of higher level life-cycle artefacts, such as requirements, designs, and subsystems [27].

---

[2] This approach to informal specification must be supported with appropriate natural language tools and reasoning mechanism, to assure specifications consistency and completeness.

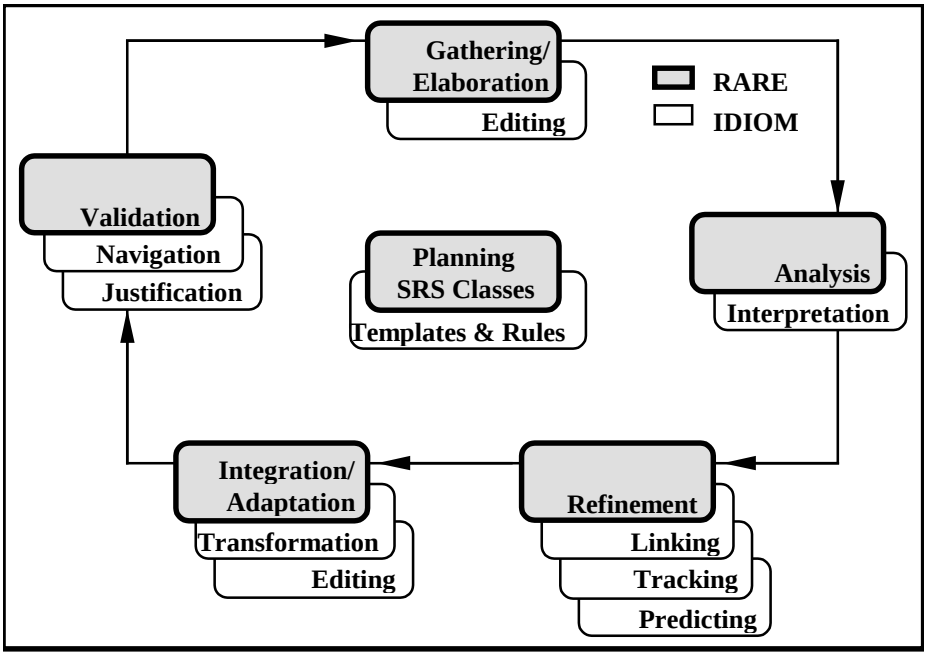

Figure 3 - Processing methods in RARE IDIOM

- Reusing requirements and specifications, rather then designs or code, provides cost and quality benefits, as well as, developmental assistance earlier in the software life-cycle [58].
- CASE-assisted reuse of requirements and high level designs early in the design process results in the reuse of subsequent lifecycle products [67].
- To effectively utilise available resources in the software development (i.e. software artefacts, techniques, methods, tools and human expertise), the resources that are applicable for the development of a target system must be identified early in the life-cycle, i.e. system requirements analysis phase [51].

RARE IDIOM embraces the need for requirements and specification reuse over the software lifecycle and provides the method and tool facilities to empower its users with effective identification and elaboration of reusable specifications, in the earliest possible stages of software development, i.e. when requirements are still in their vague and incomplete form, subject to further negotiations, reviews and changes.

RARE IDIOM Architecture and Operation

From the user's point of view, RARE IDIOM was designed to work in the simplest possible way, similar to that of a wordprocessor with spell and grammar checkers [61]. RARE IDIOM, however, clearly goes beyond text proofing, the system focuses instead on the creation of complete, correct, and formal requirements documents by gradual refinement and elaboration of their informal versions.

The methods utilised in RARE IDIOM, i.e. the RARE methodology and the IDIOM tool, demonstrate several novel approaches to the processing of requirements specification documents. We will describe these methods by giving a brief description of the system's requirements elicitation process which differs from a typical cycle of elicitation. The

RARE method views the elicitation process from the viewpoint of the IDIOM tool's capability, hence, it extends the elicitation process to include the elements of requirements modelling and analysis, thus, resulting in the following series of RARE IDIOM steps :- template planning, information gathering, followed by its analysis, refinement, adaptation and integration, validation, and cyclic elaboration of requirements (cf. Figure 3).

- **Planning.** RARE IDIOM *templates* are created to define classes of requirements documents sharing similar structure, visual text attributes, and comparable syntax and semantics. In the past, templates were used in a variety of information management projects, e.g. to improve program coding [3, 54, 75], to structure hypertext information [14], to organise software documents [29], or even process requirements specifications as in our case [21, 7]. RARE IDIOM templates, similar to those used by Garg and Scacchi, extend the notion of data forms and allow the composition and organisation of otherwise unstructured texts, thus, facilitating more effective processing and extraction of information contained in such textual documents.
- **Gathering/Elaboration.** Creation and the subsequent elaboration of a RARE IDIOM document involves *filling in* a template with text, so that the resulting document conforms to the pre-planned structure, style, form and contents. Text composition in such template-based editing is greatly simplified as the system automatically positions and composes entire document sections, inserts 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. Similar editing capabilities are available, in a more restricted way, in some commercial wordprocessing software, e.g. outlining and stationary facilities in Microsoft Word [61].
- **Analysis.** RARE IDIOM requirements texts are being *interpreted* with a bottom-up chart parser [52, 30] in search of textual references to reusable specifications described in some other, referent, documents. As a result of this analysis RARE IDIOM builds some knowledge structures describing those references and relationships between them, and stores them in a case-frame system [11] for later processing. In many respects our method of text processing is similar to those used in information retrieval [55, 43], knowledge acquisition [66, 74], hypertext structuring [35, 26], or automatic programming [62, 4]. What makes our approach different from others is our choice of inter-textual rather inter-conceptual semantics of analysed text. Such an approach gives us a seamless integration of our notion of text interpretation with that of hypertext linking and navigation [13].
- **Refinement.** Multiple interpretations of concepts appearing in requirements documents must be resolved and refined before their complete formalisation. Concept refinement is conducted by the user in the process of hypertext *linking*. The user is allowed to browse through the text of requirements, analysing the presented lists of possible requirements interpretations, and selecting those of the RARE IDIOM suggestions which best reflect his or her needs. Upon each selection, the system defines a hypertext link between the referencing and the referent texts, settles the semantics for a given passage of text, and finally, amends the RARE IDIOM's knowledge base with the chosen interpretation. While the automatic generation of hypertext links from natural language texts is certainly not new [59, 26], though, our approach of resolving the text semantics in the process of

hypertext link construction is not common. The refinement process is further assisted with RARE IDIOM's ability to limit the space of possible text interpretations by *predicting* the user preferences based on previously observed, by means of IDIOM activity *tracking*, patterns of interpretation, refinement and adaptation. This feature is similar to that of active browsing developed at University of Ottawa [24], or that used in HyperCAL to accelerate hyper-navigation [42].

- **Integration/Adaptation.** While the process of document refinement results in a collection of hypertext links from informal requirements into components of reusable specifications, this, by no means, constitutes the formalisation of software requirements. RARE IDIOM extends this refinement process. It takes advantage of reuse information contained in the links, to *transform* informal requirements into fragments of a formal specification. Such fragments are then integrated by RARE IDIOM into a continuos, though only partial, specification document. Specifications could later be *edited*, corrected and completed by the user to reflect the intended meaning of informal requirements. The method of software transformations have been commonly used to convert specifications or designs into programs [16, 25] and in some cases also applied to the earlier stages of software development [41, 45]. Few researchers indicate the possibility of utilising reuse information in this transformation process, as it is done in RARE IDIOM.
- **Validation.** Once the inter- and intra-document links have been established, RARE IDIOM allows easy navigation between templates, documents, and their components. Apart from simple document reading and browsing, navigation may also assist the user in validating current interpretation and formalisation of requirements texts. Additional validation checks may also be conducted with RARE IDIOM's justification of text interpretations by showing the user all of the lexical, grammatical and semantic cues that lead to the formulation of requirements specifications.

RARE IDIOM Example

Figure 4 and Figure 5 provide a simple example of RARE IDIOM capabilities. Figure 4 shows two templates and Figure 5 two specification documents linked into a hypertext system. One of the templates, "Module Spec", allows the creation of informal requirements for software modules, the other, "LSL Spec", permits refinement of these informal texts into the text of specifications written in Larch LSL [33].

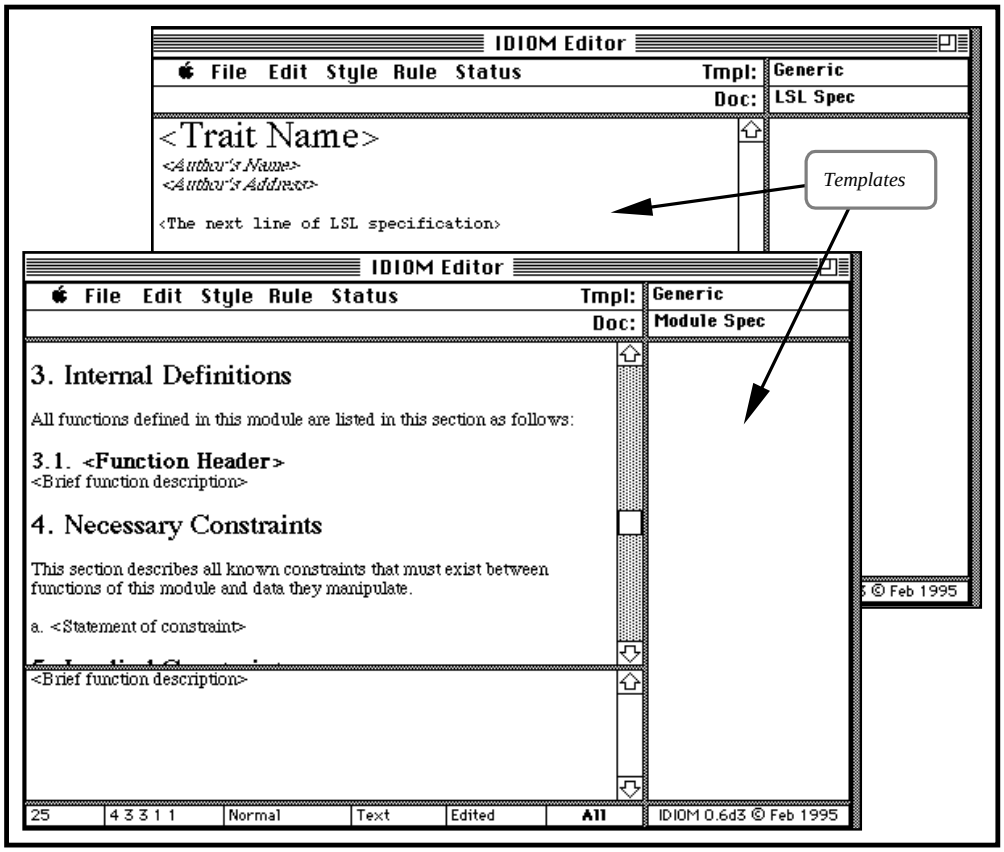

Figure 4 Example of RARE IDIOM templates

Both of the templates provide definitions of document outlines (in a form of nested paragraph structures), paragraph styles (in terms of a font face, size, protection, visibility, or header numbering), and grammatical rules (governing the syntax of text entered into the paragraphs of derived documents). For instance, in "Module Queue 1", all section headers numbered "3.x" will follow the syntax of "Function <name>" defined in paragraph "3.1" of "Module Spec", whereas the subsections of "3.x" will allow a much more flexible, though, still controlled syntax allowing entering the following variations: "This function <action>", "The function <action>", "Function <name> <action>", "<name> <action>", etc. The grammatical rules may also apply to the entire hierarchy of paragraphs, so that individual lines of text could be treated as composite of a larger parsing structure - e.g. the entire LSL specification is captured as a unit of text that has to conform to a single grammatical rule "<LSL Trait>", even though its individual paragraphs conform to a dummy rule "<LSL Cont>", indicating the continuation of the specification text.

Once a document has been derived from its template, e.g. "Module Queue 1" from "Module Spec", its contents can be freely entered, and subsequently analysed. During analysis, RARE IDIOM checks the text conformance to the pre-defined grammar, and flags all detected discrepancies. Having corrected all spelling and grammatical mistakes, the requirements engineer can inspect the document for any references to previously acquired reuse concepts (from analysed and refined documents). He or she scans the document paragraph by paragraph and RARE IDIOM provides hints as to the possible refinement of individual requiements, e.g. the paragraph: "The function calculates the length of a queue"

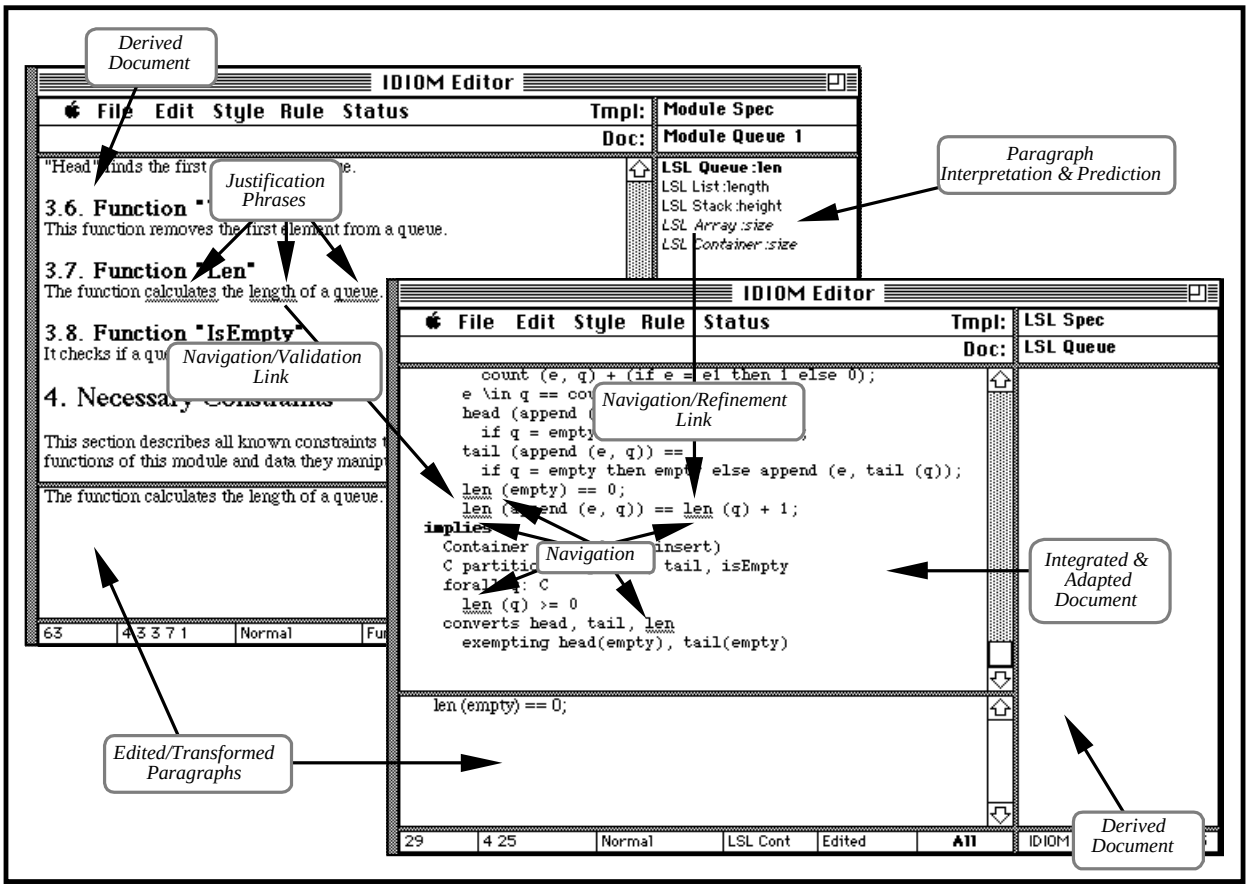

Figure 5 Example of RARE IDIOM documents

will match a number of formal concepts related to the abstract data type's size, e.g. that of a list, stack, array or a generic container. The selection of one of these interpretations, e.g. "LSL List: length" will highlite those of the paragraph phrases which lead to that interpretation in the first place, e.g. "calculates" and "length". After the entire document has been refined, the selected refinement concepts can now be gathered into a new formal document, automatically filling in the missing components as required, e.g. into "LSL Queue". This partial specification will have to be altered and adapted to better suit requirements engineer's objectives.

RARE IDIOM will observe all such changes and will attempt to repeat them in future, should the user refine another document in a similar manner. Once the process is complete, the refinement links can be navigated both ways, i.e. down the refinement path to assess the formalisation of requirements, up the traceability path in search of the original user's formulation of requirements, and laterally in search of all uses of and relationships between requirements concepts.

Summary

This chapter investigated the possibility of improving the process of requirements elicitation from informal texts by identifying and elaborating reuse information therein. The chapter also described the RARE IDIOM prototype assisting this elicitation process. Although RARE IDIOM is still under development, several objectives of the system can be demonstrated in practice.

- IDIOM prototype was implemented on a Macintosh Powerbook 170 computer running System 7.1. The system features a template-based editor, browser and linker all implemented in Apple HyperCard V2.1, they are complemented with an intelligent text interpreter, navigator and justifier in Open Prolog 1.0.4. The knowledge-base, grammar and lexicon were also implemented in Prolog. The implementation of specification transformations is still under investigation, the tracker and predictor were demonstrated in an experimental tutoring system HYPERCAL [42], but not integrated with RARE IDIOM as yet. Our choice of a prototype implementation platform resulted in its low speed-performance, making it only a concepts demonstrator rather than a practical requirements engineering tool. To improve the system efficiency, it is considered to port the system to C, X/Motif with DBQ database under UNIX.
- RARE method of eliciting requirements was defined and compared against other techniques in software requirements engineering. Since the method takes advantage of IDIOM as it primary tool, it, thus, required the customisation of IDIOM's knowledge-base to store reuse information. RARE also needed certain extensions to IDIOM's lexicon and grammar, so that they could accommodate the syntax and semantics of Larch LSL, our formal specification language. Further progress on the RARE and IDIOM integration is anticipated to occur later this year.
- RARE IDIOM's predecessor, SODA, was adopted by Amdahl Australian Intelligent Tools Programme as part of its HYPERCASE system [20, 21]. This work is conducted in parallel to RARE IDIOM developments, so the eventual convergence of concepts is likely.

Although, this paper demonstrated RARE IDIOM as a promising technique in the process of software requirements engineering, it is also planned, however, to extend RARE IDIOM tools and methods so that they are not limited to software engineering texts only, but, were also applicable to other engineering and business domains.

[1] R. J. Abbott, “Program design by informal English descriptions,” *Communications of the ACM*, vol. 26, pp. 882-894, 1983.

[2] C. Aguilera and D. M. Berry, “The use of a repeated phrase finder in requirements extraction,” *Journal of Systems and Software*, vol. 13, pp. 209-230, 1990.

[3] A. L. Ambler and M. M. Burnett, “Influence of visual technology on the evolution of language environments,” in *Visual Programming Environments: Paradigms and Systems*, E. P. Glinert, Ed. Los Alamitos, California: IEEE Computer Society Press, 1990, pp. 19-32.

[4] R. Balzer, “15 year perspective on automatic programming,” *IEEE Trans. on Soft. Eng.*, vol. SE, pp. 1257-1268, 1985.

[5] R. M. Balzer, N. Goldman, and D. Wile, “Informality in program specifications,” *IEEE Trans. on Software Eng.*, vol. SE, pp. 94-103, 1978.

[6] D. Barstow, “A perspective on automatic programming,” *The AI Magazine*, vol. 5, pp. 5-28, 1984.

[7] B. Biebow and S. Szulman, “Enrichment of semantic network for requirements expressed in natural language,” presented at Information Processing'89, San Francisco, California, 1989.

[8] A. Borgida, S. Greenspan, and J. Mylopoulos, “Knowledge representation as the basis for requirements specifications,” in *IEEE Computer*, 1985, pp. 82-90.

[9] J. P. Bowen and M. G. Hinchey, “Seven More Myths of Formal Methods,” Oxford University Computing Laboratory PRG-TR-7-94, 1994.

[10] J. W. Brackett, “Software Requirements,” Carnegie Mellon University, Software Engineering Institute, SEI Curriculum Module SEI-CM-19-1.2, 1990.

[11] B. Bruce, “Case systems for natural language,” *Artificial Intelligence*, vol. 6, pp. 327-360, 1975.

[12] A. Burns, D. Duffy, C. MacNish, J. McDeremid, and M. Osborne, “An Integrated Framework for Analysing Changing Requirements,” Department of Computer Science, University of York PROTEUS Deliverable 3.2, 1995.

[13] B. Campbell and J. M. Goodman, “HAM: a general purpose hypertext abstract machine,” *Communications of the ACM*, vol. 31, pp. 856-861, 1988.

[14] K. S. Catlin and L. N. Garett, “Hypermedia templates: an author's tool,” presented at Hypertext'91, San Antonio, Texas, 1991.

[15] S.-K. Chang, “Visual languages: a tutorial and survey,” in *Visual Programming Environments: Paradigms and Systems*, E. P. Glinert, Ed. Los Alamitos, California: IEEE Computer Society Press, 1990, pp. 7-17.

[16] T. E. Cheatham, Jr., “Reusability through program transformations,” in *Software Reusability: Concepts and Models*, vol. 1, T. J. Biggerstaff and A. J. Perlis, Eds. New York, New York: ACM Addison Wesley Publishing Company, 1989, pp. 321-336.

[17] D. N. Chin, K. Takea, and I. Miyamoto, “Using natural language and stereotypical knowledge for acquisition of software models,” presented at Proc. IEEE International Workshop on Tools for Artificial Intelligence, Fairfax, VA, USA, 1989.

[18] M. G. Christel and K. C. Kang, “Issues in Requirements Elicitation,” Software Engineering Institute, Carnegie Mellon University CMU/SEI-92-TR-12, 1992.

[19] Cmu/Sei, “Requirement Engineering and Analysis,” Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania CMU/SEI-91-TR-30, 1991.

[20] J. L. Cybulski and K. Reed, “A hypertext-based software engineering environment,” in *IEEE Software*, vol. 9, 1992, pp. 62-68.

[21] J. L. Cybulski and K. Reed, “The Use of Templates and Restricted English in Structuring and Analysis of Informal Requirements Specifications,” Amdahl Australian Intelligent Tools Programme, La Trobe University, Bundoora, Research Report TR024, 1993.

[22] D. D. Dankel, M. S. Schmalz, and K. S. Nielsen, “Understanding natural language software specifications,” presented at Fourteen International Avignon Conference, AI'94, Paris, France, 1994.

[23] M. DeBellis, “The Concept Demonstration Rapid Prototype System,” presented at Fifth Annual Knowedge-Based Software Assistant Conference, Syracuse, NY, 1990.

[24] C. Drummond, R. Holte, and D. Ionescu, “Accelerating browsing by automatically inferring a user's search goal,” presented at KBSE'93, The Eighth Knowledge-Based Software Engineering Conference, Chicago, Illinois, 1993.

[25] M. S. Feather, “Reuse in the context of a transformation-based methodology,” in *Software Reusability: Concepts and Models*, vol. 1, T. J. Biggerstaff and A. J. Perlis, Eds. New York, New York: ACM Addison Wesley Publishing Company, 1989, pp. 337-359.

[26] W. Fitzgerald and C. Wisdo, “Using natural language processing to construct large-scale hypertext systems,” presented at Eighth Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, 1994.

[27] W. Frakes and S. Isoda, “Sucess factors of systematic reuse,” in *IEEE Software*, 1994, pp. 15-19.

[28] N. E. Fuchs, H. F. Hofmann, and R. Schwitter, “Specifying Logic Programs in Controlled Natural Language,” Department of Computer Science, University of Zurich 94.17, 1994.

[29] P. K. Garg and W. Scacchi, “Hypertext system to manage software life-cycle documents,” *IEEE Software*, vol. 7, pp. 90-98, 1990.

[30] G. Gazdar and C. Mellish, *Natural Language Processing in PROLOG*. Wokingham, England: Addison-Wesley Pub. Co., 1989.

[31] A. Girgensohn, D. F. Redmiles, and F. M. Shipman, III, “Agent-based support for communication between developers and users in software design,” presented at KBSE'94, The Ninth Knowledge-Based Software Engineering Conference, Monterey, California, 1994.

[32] O. C. Z. Gotel and A. C. W. Finkelstein, “An analysis of the requirements traceability problem,” presented at The First International Conference on Requirements Engineering, Colorado Springs, Colorado, 1994.

[33] J. V. Guttag and J. J. Horning, *Larch: Languages and Tools for Formal Specifications*. New York: Springer-Verlag, 1993.

[34] A. Hall, “Seven Myths of Formal Methods,” in *IEEE Software*, 1990, pp. 11-19.

[35] R. Hammonwohner and U. Thiel, “Context oriented relations between text units - a structural model for hypertexts,” presented at Hypertext'87, Chapel Hill, North Carolina, 1987.

[36] H. F. Hofmann, “Requirement Engineering: A Survey of Methods and Tools,” Institut fŸr Informatik der Universitat ZŸrich 93.05, 1993.

[37] C. P. Hollocker, “A review process mix,” in *System and Software Requirements Engineering*, R. H. Thayer and M. Dorfman, Eds. Los Alamitos, California: IEEE Computer Society Press, 1990, pp. 485-491.

[38] P. Hsia, A. Davis, and D. Kung, “Status Report: Requirements Engineering,” in *IEEE Software*, 1993, pp. 75-79.

[39] Ibm, “System Application Architecture: AD/Cycle Concepts,” International Business Machines Corp. GC26-4531-01, 1991.

[40] Ieee, “Guide to Software Requirement Specifications,” The Institute of Electrical and Electronics Engineers, Inc. Std 830-1984, 1984.

[41] N. Iscoe, “Domain-specific reuse: an object-oriented and knowledge-based approach,” in *Software Reuse: Emerging Technology*, W. Tracz, Ed.: IEEE Computer Society Press, 1988, pp. 299-308.

[42] C. M. Jackway, “Extending Hypertext by Modeling User's Knowledge,” Department of Computer Science and Computer Engineering, La Trobe University, Honours Thesis 1994.

[43] M. Jarke, J. A. Turner, E. A. Stohr, Y. Vassiliou, N. H. White, and K. Michielsen, "A field evaluation of natural language for data retrieval," *IEEE Transactions on Software Engineering*, vol. SE, pp. 97-114, 1985.

[44] W. L. Johnson, K. M. Benner, and D. R. Harris, "Developing formal specifications from informal requirements," *IEEE Expert*, vol. 8, pp. 82-90, 1993.

[45] W. L. Johnson and M. S. Feather, "Using evolution transformation to construct specifications," in *Automatic Software Design*, M. R. Lowry and R. D. McCartney, Eds. Menlo Park, California: AAAI Press / The MIT Press, 1991, pp. 65-91.

[46] W. L. Johnson, M. S. Feather, and D. R. Harris, "The KBSA requirements/specification facet: ARIES," presented at 6th Annual Knowledge-Based Software Engineering Conference, Syracuse, New York, USA, 1991.

[47] W. L. Johnson and D. R. Harris, "Sharing and reuse of requirements knowledge," presented at 6th Annual Knowledge-Based Software Engineering Conference, Syracuse, New York, USA, 1991.

[48] G. W. Jones, *Software Engineering*. New York: John Wiley & Sons, 1990.

[49] M. Kado, M. Hirakawa, and T. Ichikawa, "HI-VISUAL for hierarchical development of large programs," presented at IEEE Workshop on Visual Languages, Seattle, Washington, 1992.

[50] H. Kaindl, "The missing link in requirements engineering," *ACM SIGSOFT Software Engineering Notes*, vol. 18, pp. 30-39, 1993.

[51] K. C. Kang, S. Cohen, R. Holibaugh, J. Perry, and A. S. Peterson, "A Reuse-Based Software Development Methodology," Software Engineering Institute CMU/SEI-92-SR-4, 1992.

[52] M. Kay, "Algorithm schemata and data structures in syntactic processing," Xerox Palo Alto Research Center CSL-80-12, 1980.

[53] T. Kinoshita, "A knowledge acquisition model with applications for requirements specification and definition," *SIGART Newsletter*, pp. 166-168, 1989.

[54] M. Lenz, H. A. Schmid, and P. F. Wolf, "Software reuse through building blocks," in *IEEE Software*, 1987, pp. 34-42.

[55] D. D. Lewis, W. B. Croft, and N. Bhandaru, "Language-oriented information retrieval," *International Journal of Intelligent Systems*, vol. 4, pp. 285-318, 1989.

[56] M. D. Lubars, "Wide-spectrum support for software reusability," in *Software Reuse: Emerging Technology*, W. Tracz, Ed. Washington, D.C.: Computer Society Press, 1988, pp. 275-281.

[57] K. H. Madsen, "A guide to metaphorical design," *Communications of the ACM*, vol. 37, pp. 57-62, 1994.

[58] N. Maiden and A. Sutcliffe, "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, Eds. London, U.K.: Springer-Verlag, 1989, pp. 109-113.

[59] K. L. McGraw, "HyperKAT: a tool to manage and document knowledge acquisition," in *Readings in Knowledge Acquisition: Current Practices and Trends*, K. L. McGraw and C. R. Westphal, Eds. West Sussex, England: Ellis Horwood Ltd, 1990, pp. 164-181.

[60] B. Meyer, "On formalism in specifications," in *IEEE Software*, 1985, pp. 6-26.

[61] Microsoft, *Microsoft Word User's Guide: Word Processing Program for the Macintosh, Version 5.0*. Usa: Microsoft Corporation, 1991.

[62] Y. Morimoto, S. Oyanagi, and Y. Nakayama, "Japanese language programming with accumulating a vocabulary," presented at 11th Annual International Computer Software & Applications Conference, Tokyo, Japan, 1987.

[63] S. J. Morris and A. C. W. Finkelstein, "Development of Multiple Media Documents," Imperial College, Department of Computing 1994.

[64] B. A. Myer, “Visual programming, programming by example, and program visualization: a taxonomy,” in *Visual Programming Environments: Paradigms and Systems*, E. P. Glinert, Ed. Los Alamitos, California: IEEE Computer Society Press, 1990, pp. 33-40.

[65] T. Naka, “Pseudo Japanese specification tool,” in *Faset*, vol. 1, 1987, pp. 29-32.

[66] F. Nishida, S. Takamatsu, T. Tani, and H. Kusaka, “Text analysis and knowledge extraction,” presented at 11th International Conference on CL, COLING, Bonn, Germany, 1986.

[67] J. Poulin, “Integrated support for software reuse in computer-aided software engineering (CASE),” *ACM SIGSOFT Software Engineering Notes*, vol. 18, pp. 75-82, 1993.

[68] R. Prieto-Diaz and G. Arango, “Domain Analysis and Software Systems Modeling,” . Los Alamitos, California: IEEE Computer Society Press, 1991.

[69] P. P. Puncello, P. Torrigiani, F. Pietri, R. Burlon, B. Cardile, and M. Conti, “ASPIS: a knowledge-based CASE environment,” in *IEEE Software*, 1988, pp. 58-65.

[70] H. B. Reubenstein and R. C. Waters, “The requirements apprentice: An initial scenario,” presented at 5th International Workshop on Software Specifications and Design, 1989.

[71] R. Rock-Evans, *CASE Analyst Workbenches: A Detailed Product Evaluation*. London, England: Ovum Ltd., 1989.

[72] M. Saeki, H. Horai, and H. Enomoto, “Software development process from natural language specifications,” presented at 11th International Conference on Software Egineering, Pittsburgh, Pennsylvania, 1989.

[73] A. Salek, P. G. Sorenson, J. P. Tremblay, and J. M. Punshon, “The REVIEW system: From formal specifications to natural language,” presented at The First International Conference on Requirements Engineering, Colorado Springs, Colorado, 1994.

[74] S. Szpakowicz, “Semi-automatic acquisition of conceptual structure from technical texts,” *International Journal Man-Machine Studies*, vol. 33, pp. 385-397, 1990.

[75] D. M. Volpano and R. B. Kieburtz, “The template approach to software reuse,” in *Software Reusability: Concepts and Models*, vol. 1, T. J. Biggerstaff and A. J. Perlis, Eds. New York, New York: ACM Addison Wesley Publishing Company, 1989, pp. 247-256.

[76] R. Vonk, *Prototyping: The Effective Use of CASE Technology*. Englewood Cliffs, N.J.: Prentice-Hall Int., 1989.

[77] D. P. Wood, M. G. Christel, and S. M. Stevens, “A multimedia approach to requirements capture and modeling,” presented at The First International Conference on Requirements Engineering, Colorado Springs, Colorado, 1994.

[78] J. Wood and D. Silver, *Joint Application Design*. New York: John Wiley & Sons, 1989.

[79] N. Yonezaki, “Natural language interface for requirements specification,” in *Japanese Perspectives in Software Engineering*, Y. Matsumoto and Y. Ohno, Eds. Singapore: Addison-Wesley Publishing Company, 1989, pp. 41-76.