The Formal and the Informal in Requirements Engineering

Jacob L. Cybulski
Department of Information Systems
The University of Melbourne
Parkville, Vic 3052
Phone: +613 9344 9244, Fax: +613 9349 4596
Email: j.cybulski@dis.unimelb.edu.au

Abstract

This paper discusses the issue of formality in the process of requirements engineering. First, the paper emphasises the significance of capturing, recording and maintaining user's original, sketchy, plain-English, statements of requirements. Such informal requirements are commonly used as a basis for stakeholders negotiation, they can be easily discussed and validated by the client community, and they can always be referred to in the subsequent development process. At the same time, we stress the importance of mapping informal requirements texts into their formal specification which could be regarded as a legal document between users and developers, and which could be used with confidence by developers as a complete, cohesive and unambiguous statement of client's needs. We argue the benefits and the pitfalls of integrating the two approaches to expressing software requirements within a single development paradigm. We report a number methods coping with this duality of software requirements and we offer our own reuse-based solution to requirements analysis, organisation and synthesis leading to the production of quality requirements specifications.

1. The Split

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 [29]. 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 [13].

File:	*Formal-Informal.doc*
Version:	*1*
Date:	*Sunday, September 15, 1996*
Word Count:	*2192*
Publication:	

	Project Management	**Requirements Engineering**	**Software Design**	**Software Implementation**
Goals	project plan	requirements specification	software designs	working programs
Tasks	project estimation project planning project monitoring resource management configuration management introduction of standards quality assurance	elaboration of needs and objectives requirements acquisition requirements modelling generation and evaluation of alternatives requirements validation	designing software architecture designing data model and flow designing user interfaces designing program logic and control designing tests	data manipulation program coding program testing installation
Sample Artefact Types	proposals and feasibility studies management reports project charts and tables	pictures, drawings and photographs informal requirements texts data dictionaries diagrams and charts formal specifications prototypes	diagrams and charts decision tables pseudo-code and PDL formal designs	source code files and databases
Domain	management/ problem	problem/ modelling	modelling/ implementation	implementation

Table 1 - Requirements engineering vs other development phases

As the elicitation process precedes all of requirements engineering phases, its effectiveness and efficiency is deemed to be of pivotal importance to the whole cycle. There are many factors that influence the efficacy of the requirements elicitation process, though, factors regarded as decisive to the requirements elicitation success include [14] :-

- *successful communication of requirements* - the main source of communication problems amongst project stakeholders are terminological differences, introduction of ambiguities through the use of natural language, lack of formality and rigour, and the inherent complexity of requirements [13];
- *ability to trace requirements* - traceability is considered difficult because of the complexity and frequency of software revisions, which also favour early formalisation of software requirements, the subsequent loss of their original informal form, which is one of the main problems in software maintenance [62, 26]; and,

- *well defined elicitation process* - elicitation processes are hard to control due to inadequately defined business processes, organisational procedures and the poorly understood relationships between people involved, their responsibilities and the tasks performed.

The above-mentioned problems and complexities, evidently, find their source in the human factors of the requirements engineering process and the informal nature of pre-specification form of requirements documents, i.e. the media used to record them, their loose structure and inappropriate notation.

- *Media.* Initial requirements come in variety of media types, to include free text, graphics, image, video and animation, speech, sound, and other sign systems [50]. They are rarely in a computer-readable form, and when such a computer representation is available, its structure and rigour is hardly ever yielding to further automation, formalisation and integration.
- *Organisation.* Documents produced at such early stages of software development frequently lack formal structure and logical organisation, they may include minutes of meetings, logs of email discussions, interview transcripts, statements of work, requests for proposals, operational concept documents, technical notes, manuals, and technological surveys [62].
- *Notation.* Early requirements are commonly recorded in natural language, the notation severely criticised as suffering from noise due to redundancy, 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, or wishful thinking due to the lack of realistic validation [48].

Requirements specification documents, being the basis of further development, are commonly called for to be in a precise, technical and symbolic notation. A host of commercially accepted diagramming techniques are currently in use for just this purpose [57]. Formal specification languages and methods, based on the sound mathematical foundation, have a further advantage over the informal approaches to capturing software requirements. Such formal tools have been successfully used in Ada development [43, e.g. Clear], software requirements specification [27, e.g. CML and Telos, CIM, GIST, Taxis/RML, ERAE and KAOS], and domain analysis [60, e.g. CIP, OBJ and Z]. Formal description of entire problem domains also allows for the creation of formal development frameworks, which may lead to additional cost and productivity gains [24].

Unfortunately, the issue of requirements capture is not as simple as the notational convenience, rigidity of the elicitation process, or the completeness and cohesion of the collected requirements. As some of the practicing requirements engineers and researchers note, informal notations are frequently the only form of requirements expression acceptable to the end users. Such informal methods may also bring a number of other developmental benefits to the entire process of requirements engineering.

- 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 [14].

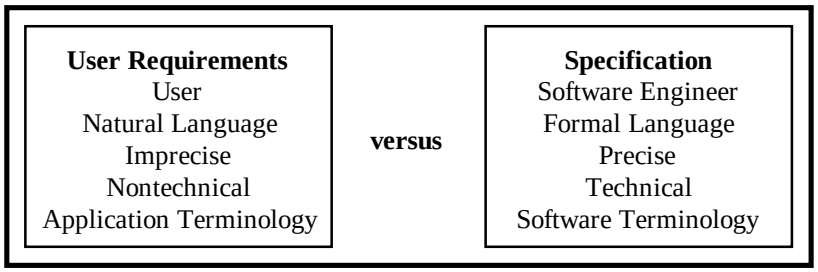

Figure 1 - Requirements vs specification dichotomy

- 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 [13].
- 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].[1]
- 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 [31].

2. The Dichotomy

The dichotomy between user requirements and their specifications is illustrated in Figure 1 [37]. This conflict of informality and rigour in the capture and specification of software requirements has been the main theme of a number of research projects which resulted in several proposed solutions to the problem.

- The most commonly practiced 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 [61], walkthroughs can be used to achieve a consensus on key requirements [30], or the formal specifications can be recast back into a form readily understood by the clients, e.g. into natural language [34, 21, 59].

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

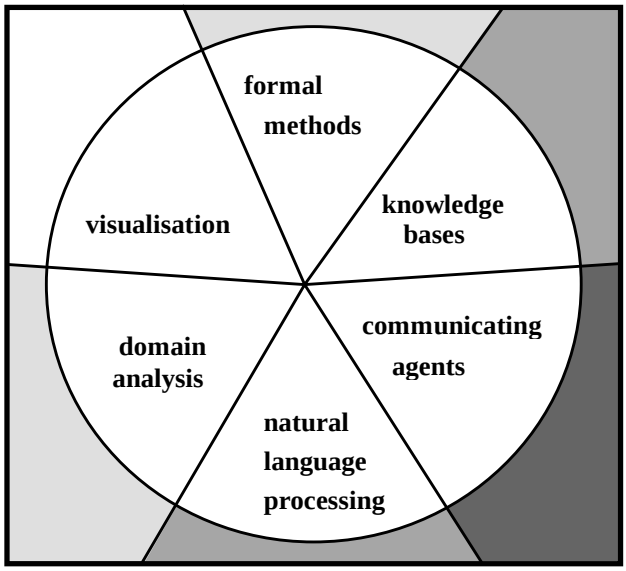

Figure 2 - Communication vs. formalisation approaches

- Alternatively, software stakeholders can be involved in a joint development of specifications starting from a very sketchy and informal project proposals down to their rigid and formalised specifications, e.g. as in JAD [63]. 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.
- Employment of communication agents monitoring end users working with prototype systems and reporting mismatches between developer's expectations and the system's actual usage [25].
- 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 [28, 8].
- Another approach is to record all informal requirements which can later be elaborated and formalised into a more rigid specification document [58]. 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 [20, 62, 39].
- 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 [56, 7, 35, 41, 55].
- Some researchers favour stating all requirements in a controlled natural language, so that all of the requirements statements could be readily analysed, structured, indexed and classified [5, 12, 2, 19], or translated into formal, possibly even executable, statements of software specification [52, 23, 64].

- Visualisation tools, which model and represent a customer problem in a graphical form [51, 3, 11, 38], 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 [45].
- Domain analysis [54] and enterprise modelling [32] aim 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.
- Finally, the systematic reuse of requirements and domain-related information in a wide-spectrum artefact reuse [44] 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;

3. The Conjunction

The work reported in this paper favours yet another approach to dealing with the duality of requirements expression. We believe that both informal and formal approaches to requirements engineering can bring certain benefits to the overall process, thus, it seems logical that they both should be integrated into one cohesive formula as also advocated by Balzer [5, 4], Abbott [1], Barstow [6], Johnson [33], Hsia [31] and Burns [10]. We admit the need to capture software requirements in the form and media most appropriate for the problem domain and convenient to the user community. We also advocate that such collections of mostly informal requirements should be analysed, refined and ultimately formalised, so that other conventional methods were possible in the ensuing development process. Where we may differ to many other researchers is our conviction that the best way of improving the cycle of requirements engineering, and elicitation in particular, is to :-

- process informal documents with a view to identify an opportunity to reuse previously formalised requirements specifications; and to this end,
- provide requirements engineers with such software reuse tools which could assist them in the process of analysis, refinement and formalisation of informal requirements while; while at the same time,
- utilising the skills, knowledge and experience of the people intimately involved in the process.

The concept of reusing system and software requirement documents, their specifications and associated development processes is not new. The recent studies clearly show that early reuse benefits the entire development process and the quality of the final product [15]. The importance of such early reuse can be best characterised by the following remarks.

- 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 [36].

- 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 [22, 47].
- 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 [46].
- CASE-assisted reuse of requirements and high level designs early in the design process results in the reuse of subsequent lifecycle products [53].
- 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 [40].

Others support these claims, voicing the need to reuse large-scale artefacts going beyond design components and including entire design frameworks and domain resources [42]. Bubenko et. al. [9] further propose to combine design and reuse libraries to accommodate development processes capable of reusing conceptual schemas to support the process of requirements engineering. Morel and Faget [49] aim at extending this approach even to the entire software life-cycle.

4. Summary and Conclusions

In this paper we elaborated the need for such a software requirements engineering process which could embrace the production of both informal and formal, open and constrained, free-format and structured, plain-English and symbolic, multi-perspective and singular, formats of software requirements documents. We looked at the issues of document media, organisation and notation. We considered pros and cons of notational formality. We have discussed a number of approaches to the emerging dichotomy of requirements communication versus their formalisation. Finally we outlined our own methods of requirements development based on the construction of informal documents and their analysis for references to formal reusable requirements specifications.

Several of the issues raised in this paper have already been partially addressed by a range of assisting technologies, such as formal specification languages, object-oriented modelling, full-text databases, information retrieval, natural language processing, hypertext and knowledge-based systems [15, 16]. All of these technologies are capable to contribute to the more effective analysis, representation and access to knowledge embedded in the informal requirements texts. We, therefore, decided to engage in the pursuit of a software development tool which could permit the freedom of informal expression, the power of computer-aided requirements analysis, and the comfort of a rigid specification at the end of the requirements engineering cycle. The tool, RARE IDIOM/SoDA [18], has been designed and prototyped, it is still in its experimental phase, however, it is planned that once it matures, it will become integrated into a cohesive and complete suite of software engineering products forming a powerful CASE environment - HyperCASE [17].

5. Acknowledgements

I would like to thank Assoc. Prof. Karl Reed for his valuable comments on the subject of this article. I would also like to acknowledge the support of Amdahl Australian Intelligent Tools Programme at La Trobe University which permitted development of tools and concepts leading to this publication.

6. References

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

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

[9] J. Bubenko, C. Rolland, P. Loucopoulos, and V. DeAntonellis, "Facilitating "Fuzzy to Formal" requirements modelling," presented at The First International Conference on Requirements Engineering, Colorado Springs, Colorado, 1994.

[10] 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.

[11] 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.

[12] 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.

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

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

[15] J. L. Cybulski, "Sharing and reuse in the development of information systems," The University of Melbourne, Department of Information Systems, Melbourne, Research Report 96/4, June 1996.

[16] J. L. Cybulski, "Reusing requirements specifications: review of methods and techniques," presented at 1st Australian Requirements Engineering Workshop, Melbourne, 1996 (To appear).

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

[18] 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.

[19] 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.

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

[21] 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.

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

[23] 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.

[24] D. Garlan, "The role of formal reusable frameworks," presented at ACM SIGSOFT, International Workshop on Formal Methods in Software Development, Napa, California, 1990.

[25] 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.

[26] 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.

[27] S. Greenspan, J. Mylopoulos, and A. Borgida, "On formal requirements modeling languages: RML revisited," presented at 16th International Conference on Software Engineering, Sorrento, Italy, 1994.

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

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

[30] 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.

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

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

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

[34] 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.

[35] 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.

[36] 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.

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

[38] 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.

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

[40] 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.

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

[42] H. Li, “Reuse-in-the-large: modeling, specification and management,” presented at Advances in Software Reuse: Selected Papers from the Second International Workshop on Software Reusability, Lucca, Italy, 1993.

[43] S. D. Litvintchouk and A. S. Matsumoto, “Design of ADA systems yielding reusable components: an approach using structured algebraic specification,” 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. 227-245.

[44] 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.

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

[46] 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.

[47] Y. Matsumoto, "Some experiences in promoting reusable software: presentation in higher abstract levels," in *Software Reusability: Concepts and Models*, vol. 2, T. J. Biggerstaff and A. J. Perlis, Eds. New York, New York: ACM Addison Wesley Publishing Company, 1989, pp. 157-185.

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

[49] J.-M. Morel and J. Faget, "The REBOOT environment," presented at Advances in Software Reuse: Selected Papers from the Second International Workshop on Software Reusability, Lucca, Italy, 1993.

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

[51] 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.

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

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

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

[55] 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.

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

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

[58] 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.

[59] 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.

[60] Y. V. Srinivas, “Algebraic specification for domains,” in *Domain Analysis and Software Systems Modeling*, R. Prieto-Diaz and G. Arango, Eds. Los Alamitos, California: IEEE Computer Society Press, 1991, pp. 90-124.

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

[62] 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.

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

[64] 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.