COMPUTER SCIENCE TECHNICAL REPORT

USE OF CASE-BASED REASONING IN THE CONTEXT-SPECIFIC SELECTION OF NOTATIONS FOR REQUIREMENTS SPECIFICATION

Sara Jones, Venki Shankararaman and Carol Britton

April 1997 Report No 310

Use of Case-Based Reasoning in the Context-Specific Selection of Notations for Requirements Specification

S. Jones, V. Shankararaman and C. Britton
Department of Computer Science
University of Hertfordshire
College Lane, Hatfield, Herts. UK
AL10 9AB
Tel: 01707 284370 / 284351 / 284354
Fax: 01707 284303
Email: S.Jones, V.Shankararaman, C.Britton@herts.ac.uk

Abstract

Choosing an inappropriate notation for specifying software requirements is likely to compromise the effectiveness and efficiency with which the requirements process can be conducted. A proper choice of notation should ideally take into account both the characteristics of available notations and the many, and often conflicting requirements and constraints placed on notations by the circumstances of the project. This paper discusses the application of case-based reasoning technology to the problem of choosing notations. Our aim is to provide a decision-support tool for managers of the requirements process which will facilitate choices about notations by exploiting knowledge about both project development contexts and the notations themselves.

Keywords: requirements specification, case-based reasoning, project context, notations

1. Introduction

Fayad and Laitinen report that there is often a tendency, amongst software practitioners, to adopt a 'one size fits all' attitude to the choice of development methods (Fayad and Laitinen 97). However, it is increasingly acknowledged that methods for software system development need to be chosen or tailored according to the needs of the specific context in which they are to be used (see, for example, Fayad and Laitinen 97, Hidding 97, Birk 97, Henninger 97).

One example of an area in which the choice of techniques for particular projects often reflects the experience or preferences of the development team more than an objective consideration of possible alternatives is that of selecting methods and notations for requirements specification (McCluskey et al 95). The impact of the choice of representation on successful performance of many system development activities has

been recognised for some time (Green 89 and 91, McCluskey et al 95, Modugno et al 94), and it therefore seems likely that the use of an inappropriate notation for requirements specification may compromise the effectiveness and efficiency with which the requirements process can be conducted.

One reason why decisions about the choice of requirements notations are not made in a more objective way may simply be that such decisions can involve a number of extremely complex considerations. A proper choice of notation should ideally take into account factors relating to the nature of the project under consideration, the characteristics of available notations and the many, and often conflicting requirements and constraints placed on notations by the circumstances of the project.

In situations such as these, where decisions need to be made on the basis of incomplete, uncertain or apparently conflicting information and a simple, algorithmic solution is not possible, the best support may be provided by knowledge-based technology of some kind. This paper discusses the application of case-based reasoning technology to the problem of choosing notations for requirements specification in a way which takes into account a broad range of considerations about both the project of interest and the notations under consideration.

The paper is structured as follows. Section 2 discusses some related work in the fields of knowledge-based support for requirements engineering and method engineering. Section 3 describes our motivation for building a tool and briefly reviews the kinds of knowledge on which it is based. Our approach to the application of case-based reasoning and our prototype implementation are presented in sections 4 and 5. We end with some discussion and directions for further work.

2. Related Work

A review of tools providing knowledge-based support for the process of requirements engineering is presented in Bolton et al (92). Most work in this area has focused on exploiting knowledge about the application domain in which a system is to be developed. Some projects have investigated the provision of knowledge-based support for analysts employing particular requirements methods (or 'methodologies'): for example, Kramer et al (88) describe a knowledge-based tool supporting the use of the CORE method. Other tools such as KATE (Fickas 87), Analyst-Assist (Adhami et al 89) and the Requirements Apprentice (Rich et al 87) were influenced by studies investigating the performance of expert requirements engineers and attempted to support or replicate various forms of expert behaviour in applying techniques for

requirements elicitation or analysis. There has, however, been little emphasis on exploiting knowledge about the context in which a project is to take place, or on supporting the requirements engineer in choosing what methods or techniques might be most appropriate for a particular problem.

Questions relating to the choice or tailoring of methods and techniques for software development are addressed within the method engineering community. One example of a knowledge-based tool which aims to support software practitioners in making decisions about what software tools and techniques to use in a new project is BORE (Henninger 97). BORE (Building an Organisational Repository of Experiences) stores knowledge about previous software development projects as 'cases' which contain information about relevant project issues as well as resources used to address these issues. The repository of cases may be browsed or searched using case 'characteristics' which are specified in terms of a controlled vocabulary. Information about the context for a particular case is provided in the form of resource lists which enumerate relevant tools, people, projects and development methods.

Because it is aimed at supporting the software development process as a whole, the knowledge used in BORE is of a very general and loosely structured nature. In our work, we focus on the process of requirements specification. This allows us to consider in more detail questions about what knowledge can be used to support decisions regarding the choice of requirements specification notations, and how such knowledge might best be structured.

3. Requirements for the Tool

An investigation into the information needs of IT practitioners with respect to different methodologies is reported by Hidding (1997). Hidding describes practitioners' dissatisfaction with the way in which information about different methodologies is currently presented in voluminous binders and handbooks which means that it is often difficult to access relevant information when it is needed. Alternative ways of presenting information are therefore needed.

Hidding distinguishes between the roles of 'planners' and 'doers' (roughly equating to the roles of managers and analysts or programmers), saying that those with the greatest need for information about the suitability of different methods for use in different situations are the planners: if those planning a project make incorrect decisions about what methods should be used, they may expose the project to unnecessary risks, and are liable to take the blame if the project runs over time or budget, or if the system

delivered is of poor quality. We have therefore targeted our efforts at providing a decision-support tool for IT planners, or managers of the requirements process.

Focusing on the specific case of selecting methods and notations for requirements specification, we can identify two different types of knowledge which a requirements manager may use in selecting notations for requirements specification. The first of these is knowledge about the project context or situation in which a notation is to be used. The second relates directly to the notations themselves. Each of these types of knowledge is discussed in more detail below.

In the following paragraphs, we briefly discuss the origins of knowledge currently being implemented in our tool and describe our initial efforts at validating this knowledge. The process is discussed in more detail elsewhere (Jones and Britton 97).

3.1 Project Context Knowledge

Various authors have characterised particular development situations in different ways. Potts, for example, discusses the needs of software developers who build off-the-shelf application software (Potts 95), Sommerville and Sawyer characterise requirements engineering for critical systems (Sommerville and Sawyer 97), McCluskey et al consider various aspects of the development situation which affected their choice of notation for use in a project developing a prototype decision support system for air traffic controllers (McCluskey et al 95), and Jones and Britton have attempted to identify the distinguishing features of multimedia development projects which might have an impact on the choice of notations for use in this area (Jones and Britton 96, Britton et al 97, Britton and Jones 98).

Work such as that described by O'Neill et al (97) and Sutcliffe et al (97) is currently investigating which features of a project context might influence the choice of notations or representations at various stages of the software development process. Christel and Kang's report for the SEI identified a number of factors relating to the scope of a project, the need for understanding by various parties, and the volatility of requirements, which they suggested should influence the developer's choice of requirements techniques and notations (Christel and Kang 92). More recently, Sommerville and Sawyer have listed some generic guidelines for choosing models and methods (Sommerville and Sawyer 97) and the RESPECT project has made some general recommendations as to the stages of system development for which certain techniques and representations are most appropriate (Maguire 97). However with each of these authors taking a slightly different perspective on the problem, it is difficult for

the practising software developer to know on what basis the choice of modelling notation for a particular project should be made.

Following Kellogg (90) we suggest that the important features of a project context which might influence a developer's choice of modelling notation could be considered in terms of factors relating to:

- the intended users of the representations (stakeholders)
- the purposes for which the representations are intended to be used, and
- the environment in which the representations are to be produced and employed, and
- specific features of the system under development.

From a review of the literature, and from the experience of software developers within the University, we have identified a number of features in each of the above areas. We currently have 6 features relating to the intended users of the representations, 12 concerning the purposes for which the representations are intended to be used, 5 about the environment in which the representations are to be produced, and 4 relating to the system under development.
Each feature is effectively a dimension along which projects may vary. For each dimension, we have identified an initial set of possible values, for example

'Extent of stakeholder involvement'

may be categorised as 'high', 'medium' or 'low' for a particular project, and

'Feasibility of training stakeholders to understand a technique'

may be categorised as 'feasible' or 'not feasible'.

3.2 Knowledge about Notations

Work on criteria for choosing modelling notations has been carried out by authors from both the academic and industrial communities. Farbey (93) identifies criteria such as readability, modifiability and lack of ambiguity, as well as the ease with which a well-presented specification can be produced, and the amount of support available. Davis (88 and 93) suggests a list of criteria pertaining to the effectiveness of representations and the choice of notations which includes the ideas that the notation should permit annotation and traceability, facilitate modification, and provide a basis for automated checking and generation of prototypes and system tests. Among the most relevant publications from authors in industry is the STARTS guide (DTI 87), which identified criteria for modelling notations including qualities such as rigour, suitability for agreement with the end-user and assistance with structuring the requirements.

Once again, we have used a review of the literature, and the experience of local developers to identify 12 important features of notations. Any notation for specifying requirements can be rated against features such as these. For example, Davis (93) provides a table in which 10 notations for requirements specification are numerically rated against 11 criteria similar to those we have used, and Wieringa (96) also provides a structured comparison of 3 different notations in terms of their coverage.

We may describe what we would ideally like from a notation for a new project by filling out values for these features. For example, we may specify that:

'Ease of understanding a model'

should be 'high', or that

'Degree to which technique is based on maths and/or logic'

may be 'low' if we are not concerned with developing a mathematically rigorous model. We may also use the same knowledge structure to record our experiences with using a particular notation at the end of a project.

3.3 Knowledge Validation

The two types of knowledge described above have so far been validated in two ways. First, we worked through four local projects, which had already been completed, characterising them in terms of our proposed project features and using our understanding of that characterisation to help identify which features of notations should have been most relevant for selecting notations for use in each project. Two further researchers, who are not part of our team, have also worked through a project of their own in this way and have suggested amendments and additions to our initial lists of features.

Secondly, we have presented our ideas to academics and practitioners in a workshop held at STEP97, a recent conference on software engineering (Budgen et al 97). The aim of this workshop was to promote discussion of any factors which may significantly influence a developer's choice of notations for requirements representation. It was attended by 10 practitioners and 12 academics and lasted for two hours. Our ideas about project context and notation knowledge were presented at the beginning of the workshop, and there was then a general discussion regarding the way in which notations are currently chosen, and way in which decisions about choice of notation might be supported. At the end of the discussion, workshop participants were asked to fill out a rating sheet in which they rated each feature of a project context or notation as 'very important', 'quite important', 'not very important' or 'not at all important'.

Detailed information about the results of this exercise is presented in Jones et al (97). Of the features considered at the workshop, 36 (out of a total of 41) were considered to be either 'very important' or 'quite important' in determining what notation to use by at least half the respondents. We therefore feel a reasonable degree of confidence in the validity of project and notation features which have so far been identified, although we plan to extend and modify our lists as we gain further experience.

4 Application of Case-Based Reasoning (CBR)

4.1 Introduction to CBR

Case-Based Reasoning is modelled on the human approach to problem solving. Upon being presented with a new problem, a human will attempt to use their knowledge of the solutions to previously encountered similar problems (or parts thereof) to arrive at a proposed solution to the new problem: for example, lawyers use old cases as precedents to justify arguments in new cases - hence the well known phrase "Precedence is 9/10 of the law". In general, human competency at solving problems in a given domain increases as a function of the number of previously encountered (and remembered!) problems and their solutions, whether successful or otherwise (Kolodner 93).

CBR does not require a domain model, rather it requires a large number of previously solved cases. Thus CBR, while perhaps not removing the well known 'knowledge elicitation bottleneck' problem in the development of classic knowledge-based systems (KBS), certainly eases it. Elicitation simply becomes the task of gathering case histories which would normally be more effectively undertaken by the users (i.e. practitioners in the problem domain) rather than by the system implementers. Ideally, a CBR system should not only store information about successful cases, but also use failed solutions to avoid pitfalls. The large number of cases required may be managed by applying database management techniques. A CBR system is thus more easily maintained than a KBS and can also learn by acquiring new knowledge as cases.

The critical element in implementing a CBR system is identifying significant features that can be used to describe a case - a much easier task than creating a domain model. A case is a piece of knowledge representing an experience in a particular context. Typically a case comprises the problem statement, including contextual information (e.g. invariant and pre-conditions), the solution to the problem, and/or the outcome of applying the solution (e.g. post-conditions).These are usually represented as a set of

feature-value pairs. Cases can be implemented using frames, objects, predicates, semantic nets and rules. Frame and object representations (which are very similar anyway) are most often used in current CBR systems.

4.2 Rationale for using CBR

The following points form our rationale for using CBR for the problem of selecting notations for requirements specification:

- The domain of notation selection is currently ill defined: it is difficult to develop theories or rules which can accurately identify appropriate notations for requirements specification based on project features.
- On completion of every project it is possible to record the experience, both good and bad, of using particular notations for the project as a case. Over a period of time it is possible to accumulate a large number of cases, which can then be used to suggest appropriate notations based on previous project experiences.
- In most instances it is very difficult to identify precisely one optimum notation for a project. The CBR approach overcomes this problem by suggesting a set of possible notations and ranking them in the order of suitability to the particular project.

4.3 Application of CBR

A tool for requirements managers wanting advice on what notations to use on a new project would ideally allow users to simply specify details of their new project in order to obtain a recommendation as to what notation or notations might be most useful. Knowledge about requirements notations and their applicability under different circumstances is not yet sufficiently developed that such a system could be implemented. However, with case-based reasoning, we are able to provide users with information about projects similar to their own as well as the notations which were used on those projects and the advantages and disadvantages to those notations which developers observed during the course of the project.

In our system, a case consists of a set of feature-value pairs defining a project context, a further set of feature-value pairs defining a notation used in that context, the name of the notation and a free text description of the experience of developers using the notation in the project described. An example of a completed case relating to a project called MAISIE is shown in figure 1.

PROJECT FEATURES	
Stakeholders:	
Extent of stakeholder involvement in the requirements process	high
Stakeholder's understanding of software systems	medium
:	:
:	:
Intended purpose of models:	
As a vehicle for communication and negotiation between the developer and other stakeholders	yes
As a vehicle for communication between members of the development team	yes
:	:
:	:
Environment:	
Stability of the requirements	medium
Likelihood of conflicts / inconsistencies	high
:	:
:	:
System:	
Extent to which the system is safety-critical	low
Extent to which the system is security-critical	low
:	:
:	:
NOTATION FEATURES	
Ease of producing a model	medium
Ease of understanding a model	high
:	:
:	:
NOTATION USED	
Name: Storyboards	
Description: Storyboards were used to script interactions in the four main sections of the system which included an on-line quiz, an interactive story and a multimedia educational component. They were extremely useful in discussing possible scenarios with teachers and school nurses who were not familiar with the full capabilities of multimedia systems. However, aspects of the system such as the dialogue structure for the interactive story had to be specified separately as the storyboards did not provide enough detail to be able to implement these directly.	

Figure 1: Extracts from one of the cases for the MAISIE project

At the beginning of a new project, a user can fill out the 'project features' part of the template, by assigning relevant values to project features of interest, and can then ask the system to find the best match (or matches) to the new project from within the case base. The user can also differentially weight features if the match is to be influenced more by the values of some features than others. By examining the cases identified by the system, the user can gain a good understanding of what notations have been used in similar projects in the past, and what are likely to be the benefits and problems associated with a range of options. An experienced requirements manager, who, at the

beginning of a new project, already has a feel for the kind of notations which might be appropriate, can also fill out the 'notation features' section of the template. The system can then use both project and notation features in finding a match from the case base, and can present the user with a smaller, and better matched sample of information about past projects.

In order that the system can 'learn' with increasing experience, users are asked to enter new cases into the case base at the end of every project. Separate cases are currently used for every notation used: for example, as the MAISIE project used the two separate notations of storyboards and state transition diagrams, two separate cases were entered into the case base. The 'project features' components of each case were the same, but the 'notation features' components recorded the ease with which models were produced, understood etc within the MAISIE project, as well as any informal comments from the developers about the pros and cons of each.

The two separate processes of using the case base to retrieve relevant information, and updating the case base at the end of a project, are shown in figure 2.

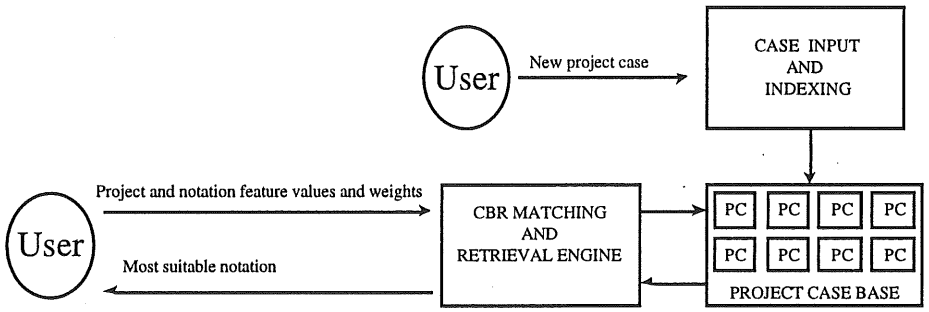

Figure 2: Creating and using the case base

Our approach has the advantage that it provides support for both experienced and less experienced project managers. It also provides an easy route to learning from experience. As more projects are completed, cases relating to those projects can quite easily be added to the system, and we may, over the course of time, build up an increasingly accurate picture of:

- overall ratings for notational features (are data flow diagrams, in general, easier to modify than state transition diagrams?)
- links between particular project features and the need for notations with particular properties (are relationships such as those identified in section 4.2 correct?), and

- effective approaches to matching new projects with those in the case base (are some project features of greater significance than others in determining what notations will be most appropriate?)

The following section briefly discusses the way in which we are implementing a prototype based on these ideas.

5. Prototype Implementation

A prototype tool is currently being implemented. The prototype being developed is an extension of an existing CBR tool called CAROL (Case Assisted Reuse of Object Libraries). The following is a brief description of the main components of the system. For more details on CAROL, the reader may refer to Shankararaman et al (96) and Maguire et al (95).

5.1 Case-Base Creation

The Case-Base Creation module allows a user to input new cases in order to build up the case base on completion of a project. This is done by setting values for the various project and notation features which define a completed project. New cases are appropriately indexed and then stored in the case base.

5.2 Specification

The Specification function provides users of the system with an input template for specifying features relating to a new project. Users can define values for some or all of the project and/or notation features in order to find a suitable notation. The input template is basically an interface between users and the case representation structure.

5.3 Retrieval

Retrieval matches the user's specification of project features either against all, or user specified parts of, the case-base. A list of matched case names is returned, together with their overall similarity score and 'goodness' of fit with the new project details, according to a default, or user defined, 'goodness' criterion. A group of options and settings are available, including the 'retrieval settings' and 'attribute weightings' facilities, which allow the user to change the search strategy by pre-selecting parts of the case-base or emphasising the importance of one or more features. This allows the user to narrow or broaden the search space, depending on whether a quick search (requiring some user knowledge about the domain of the desired project) is wanted, or whether users want the system to find as many semantic relations between cases as possible.

6. Discussion

In this paper, we have identified the need for decision-support tools which can assist the requirements manager in identifying notations for requirements specification appropriate to the needs of a particular project. We have identified two different kinds of knowledge on which such a tool may draw, and have described our initial work on eliciting and validating knowledge of these kinds. Since the task of notation selection is still rather ill defined and our knowledge is still incomplete and relatively uncertain, the most suitable form of knowledge-based technology for developing a support tool was felt to be that of case based reasoning. We have described our approach to the application of case-based reasoning technology in the domain of notation selection, and have introduced a prototype tool which implements our approach.

As discussed by Mylopoulos et al (97), AI can make an important contribution to software engineering, simply by providing knowledge representation techniques as a framework for capturing and recording knowledge. This knowledge may then usefully be exploited by human practitioners, even if it is not incorporated into a full-blown knowledge-based system. Moreover, the act of eliciting and representing previously unrecorded knowledge is in itself useful.

Our experience to date has certainly been useful in these respects, as in considering the possible application of case-based technology and assessing alternative designs for our tool, we have increased our understanding of the task of notation selection. In addition, as discussed in section 4.3, one of the benefits of our proposed design is that it provides a basis for learning more about the domain of interest. As we accumulate greater numbers of project cases from a range of organisations, we will be able to build up an increasingly accurate picture of knowledge relating to the task of notation selection. This, in turn, will allow us to refine our tool and investigate the use of further knowledge-based techniques to increase its efficiency.

References

Adhami, E., Pyburn, R. and Champion, R., "A Knowledge-based approach to requirements engineering", in Software Engineering Environments: Research and Practice, K. Bennett (ed), chap.12, Ellis Horwood, 1989.

Birk, A. "Modelling the Application Domains of Software Engineering Technologies", Fraunhofer (IESE) Technical report no. 014.97/E, 1997.

Bolton, D. Jones, S. Till, D. Furber, D. and Green, S., "Knowledge-Based Support for Requirements Engineering", in International Journal of Software Engineering and Knowledge Engineering, 2(2), 1992.

Britton, C., Jones, S., Myers,M. and Sharif,M. "A survey of current practice in multimedia system development" Journal of Information and Software Technology, 39, 1997.

Britton, C. and Jones, S. "Using project characterisation in selecting notations for modelling requirements", to be presented at IEEE Engineering Computer-Based Systems, Jerusalem, 1998.

Budgen, D. , Hoffnagle, G. and Trienekens, J. (Eds.) Proc. of the Eighth International Workshop on Software Technology and Engineering Practice, IEEE C. S. Press 1997

Christel, N. and Kang, K. "Issues in Requirements Elicitation", Technical Report CMU/SEI-92-TR-12, Software Engineering Institute, 1992

Davis, A. "A Comparison of techniques for the specification of external system behaviour" Communications of the ACM, Volume 31, Number 9, September 1988

Davis, A.M. "Software Requirements. Objects, Functions and States" Prentice Hall International, 1993

Department for Trade and Industry and National Computing Centre "The STARTS Guide", Second edition, volume 1, NCC Publications, 1987

Farbey, B. "Software quality metrics: considerations about requirements and requirement specifications" in *Software Engineering: A European Perspective*, R. Thayer and A. McGetterick (Eds.), IEEE C.S.Press, 1993

Fayad, M. and Laitinen, M., "Process Assessment Considered Wasteful", Communications of the ACM, 40(11), 1997.

Fickas, S. "Automating the Analysis Process", in Proc. of the 4th International Workshop on Software Specification and Design, IEEE C. S. Press, 1987.

Green, T.R.G. "Cognitive Dimensions of Notations" in *People and Computers (HCI 89)* Sutcliffe and McCaulay (Eds.) C.U.P, 1989

Green, T.R.G. (1991) "Describing Information Artifacts with Cognitive Dimensions and Structure Maps" in *People and Computers VI, Proceedings of the HCI'91 Conference*, Diaper, D. and Hammond, N. (Eds.), August 1991

Henninger, S. "Case-Based Knowledge Management Tools for Software Development", in Automated Software Engineering, 4(3), July 1997.

Hidding, G., "Reinventing Methodology: Who Reads it and Why?", Communications of the ACM, 40(11), 1997.

Jones,S. and Britton,C. "Early Elicitation and Definition of Requirements for an Interactive Multimedia Information System" Proc. of ICRE96, the Second International Conference on Requirements Engineering, IEEE Computer Society Press, 1996.

Jones, S., Britton, C. and Lam, W. "Towards A Framework for Selecting Notations for Modelling Requirements" Department of Computer Science, University of Hertfordshire Technical Report, 1997.

Kellog, W. "Qualitative Artifact Analysis", in *Human Computer Interaction: INTERACT90* Diaper, D., Gilmore D., Cockton, G. and Shackell, B. (Eds.) Elsevier Science Publishers, BV (North Holland) 1990.

Kolodner, J. "Case-based Reasoning", Morgan Kaufman, 1993.

Kramer, J., Ng, K., Potts, C. and Whitehead, K., "Tool Support for Requirements Analysis", Software Engineering Journal, May 1988.

Macaulay, L.A. "Requirements for Requirements Engineering Techniques", Proceedings of ICRE96, the Second International Conference on Requirements Engineering, IEEE Computer Society Press, 1996

Maguire, M. "RESPECT User Requirements Framework Handbook" Version 2.2, HUSAT Research Institute, April 1997.

Maguire, P. Shankararaman, V., Szegfue, R. and Morss, L., "Application of Case-Based Reasoning to Software Reuse" in Lecture Notes in Artificial Intelligence: Progress in Case Based Reasoning, I. Watson (ed), Springer Verlag, 1995.

McCluskey, T.L., Porteous, J.M., Naik, Y, Taylor, C.N. and Jones, S. (1995) *A Requirements Capture Method and its use in an Air traffic Control Application.* Software practice and Experience, Vol 25 (1)

Modugno, F., Green T.R.G. and Myers B.A. (1994) "Visual programming in a Visual Domain: A Case Study of Cognitive Dimensions" in *People and Computers IX, Proceedings of HCI'94*, Glasgow, August 1994

Mylopoulos, J., Borgida, A. and Yu, E., "Representing Software Engineering Knowledge", Automated Software Engineering", 4 (3), 1997.

O'Neill, E., Johnson, P. and Johnson, H. "Representations in co-operative software development: an initial framework", Proc. of the Intl. Workshop on Representations in Software Development, July 1997, Queen Mary and Westfield College, University of London

Potts, C. "Invented Requirements and Imagined Customers: Requirements Engineering for Off-the-Shelf Software", in Proceedings of RE95, the Second IEEE International Symposium on Requirements Engineering, IEEE C. S. Press, 1995.

Rich, C., Waters, R. and Reubenstein, H., "Toward a Requirements Apprentice", in Proc. of the 4th Intl Workshop on Software Specification and Design, IEEE C. S. Press, 1987.

Shankararaman, V. and Maguire, P., "Case-Based Reasoning for Software Reuse", in Knowledge Based Computer Systems - Research and Applications, K. Anjaneylu, M. Sasikumar and S. Ramani (eds), Narosa Publishing House, London, 1996.

Sommerville, I. and Sawyer, P. "Requirements engineering: a good practice guide", Wiley, 1997

Sutcliffe, A.G., Maiden N.A.M. and Bright, B. "An Evaluation Framework for Representations in Requirements Engineering", Proc. Intl. Workshop on Representations in Software Development, July 1997, Queen Mary and Westfield College, University of London

Wieringa, R., "Requirements Engineering", Wiley, 1996.