Deriving Goals from a Use-Case Based Requirements Specification for an Electronic Commerce System

Annie I. Antón, John H. Dempster, Devon F. Siege

North Carolina State University
College of Engineering
1010 Main Campus Drive
Raleigh, NC 27695-7534 USA
+1 919.515.5764
{aianton,jhdempst,dfsiege}@eos.ncsu.edu

Abstract. Use cases and scenarios have emerged as prominent analysis tools during requirements engineering activities due to both their richness and informality. In some instances, for example when a project's budget or schedule time is reduced on short notice, practitioners have been known to adopt a collection of use cases as a suitable substitute for a requirements specification. This shortcut is cause for concern and deserves focused attention. We describe our experiences during a goal-driven requirements analysis effort for an electronic commerce application. In particular, we identify the specific risks incurred, focusing more on the challenges imposed due to traceability, inconsistent use of terminology, incompleteness and consistency, rather than on traditional software project management risks. We conclude by discussing the impact of the lessons learned for requirements engineering in the context of building quality systems.

1 Introduction

Scenarios have increased in popularity among software engineers due, in part, to Jacobson's use case approach [Jac92] and the more recent introduction of the Unified Modeling Language (UML) [BRJ99] for systems engineering. The availability of UML and the associated tool support has made scenario and use case analysis even more accessible to requirements analysts and practitioners.

Scenarios are used purposefully to "stimulate thinking" in all various disciplines, including human computer interaction (HCI), strategic management and software engineering [JBC98]. Scenarios are also a reasonable approach for managing change during the software process; however, managing scenario traceability across multiple changes becomes increasingly difficult. Practitioners using scenarios and/or use cases in industry incur very specific challenges. Specifically, as reported in [WPJ98], several key areas need support including: the need for appropriate process guidance as well as comprehensive support for managing both scenario traceability and evolution.

This case study was motivated by our desire to observe scenario management in an industrial setting. We employed the goal and scenario identification and elaboration heuristics available in the GBRAM (Goal-Based Requirements Analysis Method) [Ant96, Ant97]. This paper reports on our experiences in deriving goals from a collection of use cases during requirements analysis activities for an electronic commerce application. In Section II we discuss relevant work in goals and scenarios for require-

ments specification. Section III describes the case study by focusing on goal analysis and evolution throughout our investigation. Section IV discusses challenges and associated risks. The lessons learned are detailed in Section VI, followed by discussion of future work in Section VII.

2 Related Work

Scenarios aid analysts and stakeholders in developing an understanding of current or envisaged systems and business processes [AMP94, Jac92, PTA94, WPJ98]. They describe concrete system behaviors by summarizing behavior traces of existing or planned systems. Use cases [BRJ99, Jac92], describe the possible system interactions that external agents may have with a system. In UML, scenarios are comprised of sets of actions and interactions that involve specific objects [BRJ99]. A representational framework for scenarios and use cases appears in [AP98b].

Scenario analysis is a very effective and proven technique for surfacing goals during requirement engineering. Goals are the objectives and targets of achievement for a system. Goal-driven approaches focus on why systems are constructed, expressing the rationale and justification for the proposed system. Focusing on goals, instead of specific requirements, allows analysts to communicate with stakeholders using a language based on concepts with which they are both comfortable and familiar.

For this requirements specification effort, we employed the GBRAM in which goals are operationalized and refined into requirements and point to new, previously unconsidered scenarios (for a detailed explanation of the GBRAM and its heuristics see: [Ant97]). Similarly, scenarios also help in the discovery of goals [AMP94, AP98a, JBC98, Pot99, RSB98].

3 Electronic Commerce and Quotation System Analysis

The GBRAM [Ant97] was employed to analyze an existing requirements specifcation for an electronic commerce, company-wide Intranet system to manage the quotation and ordering process for product bidding. This section provides an overview of our goal analysis efforts and summarizes the evolution of goals throughout our study.

The company with which we collaborated has numerous plants throughout Europe that produce a variety of electrical products. The parent company maintains its own sales force, whose members provide quotations for product pricing and place orders for customers. Existing processes for providing quotations or ordering products were numerous and ad hoc, with each sales person and/or each plant having a different process, using various computer programs, printed catalogs or direct sales persons as plant contacts. A new, more tightly integrated system was needed to facilitate the provision of consistent lowest prices with a streamlined bidding process to aid the company's distributed sales force. Specifically, the new system must produce consistent quotations and order prices while tracking statistical information, such as market trends.

3.1 Goal Analysis Efforts

Our analysis team was provided with a Software Requirements Specification (SRS) containing typical descriptive information such as system scope, boundaries, a feature summary, etc. as well as a total of 52 use cases and 26 screen designs. The existing requirements specification was based on outlines of web-screens, and in attempt to create an implementation-independent RS, developers had created use cases based on the screens. These use cases essentially described what the users were supposed to do with the screens. Our efforts entailed deriving goals from these use

cases. Each goal was annotated with relevant auxiliary notes including agents, constraints, pre- and post-conditions, scenarios and questions as well as answers provided by various stakeholders during follow-up interviews. For traceability we tracked and documented any changes to the goals and the associated rationale.

Each of the 52 use cases in the SRS includes the following information: a unique identifier (integer), a title (concise and representative textual name), an overview, pre- and post-conditions, a main scenario, zero or more secondary scenarios, the required Graphical User Interfaces (GUIs), a list of use cases included or extended by each use case, and a revision history.

The overview provides a brief summary of the use case, capturing a synopsis of the interactions of the interactions described in the use case. Pre- and post-conditions describe those states that must be true prior to the initiation or after the completion of a use case, respectively. The main scenario is an ordered list of user interactions with the system. Whereas the main scenario is normative, secondary scenarios show possible branching and possible alternative courses (such as when exceptional conditions arise). The steps in the main scenario are numbered sequentially while the secondary scenarios are essentially textual descriptions in the form of a paragraph with no numbered steps. The required GUIs are enumerated as a list of system interfaces that an actor uses to complete interactions described by the main and secondary scenarios. An enumerated list of use cases that the use case 'uses' or 'extends' describes the dependency relationship between the use case described and the use case mass [BRJ99]. The main or secondary scenarios must reference each use case in this list to provide the proper context. Finally, authors document changes in the revision history. The term 'authors' refers to the six individuals who authored the SRS, whereas the term 'analysts' refers to those who actively participated in this case study (the co-authors of this paper). 'Stakeholders' include the authors of the SRS as well as a number of intended users of the electronic commerce and quotation system.

Analysts met for sessions ranging from one to three hours in duration, once a week for two months. Prior to each weekly meeting, each analyst performed a goal analysis of agreed upon use cases that were then discussed and revised while collaboratively recording all goals and auxiliary notes. Meeting preparations allowed us to concentrate primarily on revising and extending the original analyses during our meeting sessions. During these sessions initially generating a total of 292 goals as discussed below.

3.2 Goal Evolution

Goal analysis began by identifying goals in main and secondary scenarios. The information tracked (partially to ensure traceability) included the goal number, responsible agent(s), the use case from which the goal was derived, constraints, pre- and post-conditions, as well as issues, rationale or questions related to each goal. Analysts identified goals from predetermined scenarios prior to meeting sessions. During the sessions we reviewed the goals, documenting any newly identified goals.

Goals were named using meaningful keywords selected from a predefined set of goal categories [Ant97]. The boxes in the ellipse in Figure 1 show the number of goals named with each of these keywords. In Section IV we provide further discussion of `NOTIFY`, `INFORM`, `PROVIDE` and `ALLOW` goals. Figure 1 shows the evolution of the goal set. Shaded boxes represent goals removed from the goal set for various reasons that we now discuss. Our initial analysis of the use cases produced 292 goals. The top two rectangles on the right side of the figure show the number of goals that were merged to eliminate synonymous and redundant goals.

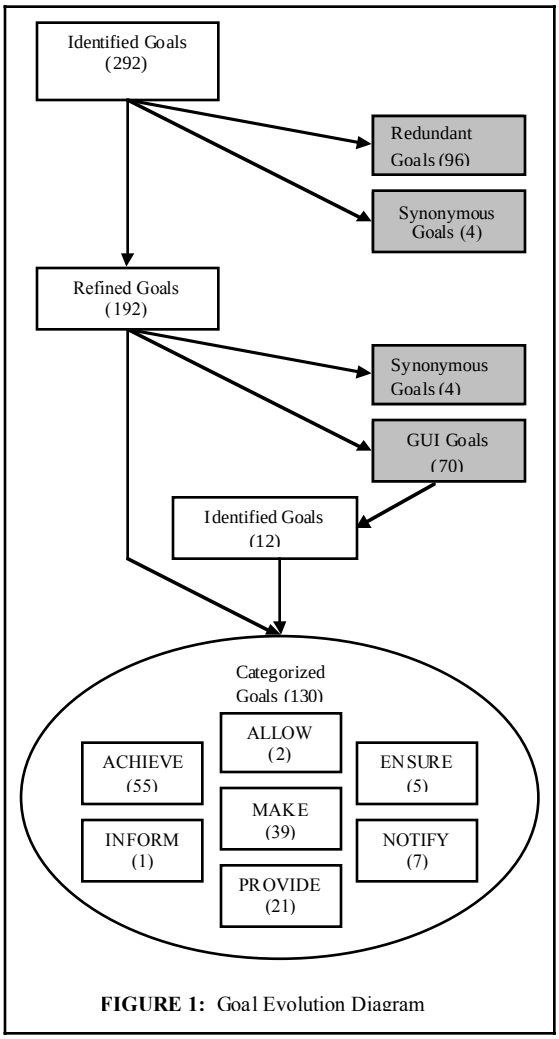

FIGURE 1: Goal Evolution Diagram

Goals were refined by applying the GBRAM heuristics [Ant97], resulting in the elimination of 100 goals. 70 GUI-specific goals were deleted, however 12 of them were suggestive of underlying processes or activities/ these goals were then reformulated and included in the goal set. For example, the original goal `<MAKE a match message displayed>` was restated as: `<NOTIFY user of no match>`. These restated goals convey important implementation-independent information; the categorization enriched the goal set by replacing implementation-specific goals. While identifying these goals we struggled due to the five specific challenges discussed in Section 4.

4 Challenges Encountered and Associated Risks

We faced various challenges requiring unexpected extra work on the part of the analysts and which introduced additional risk into the requirements specification.

Context was not always obvious for each use case

Use cases in and of themselves do not always provide an adequate understanding of the interaction that they are intended to describe; this is partially due to the way individual authors write use cases. Several factors required us to repeatedly refer to other use cases to deepen our contextual understanding. First, while we are all experienced with the electronic commerce domain, some use cases dealt with company-specific practices and policies (e.g., underlying business rules which had not been clearly articulated; the use case authors had the benefit of this tacit knowledge). Second, the SRS use case overviews did not provide enough information to describe the circumstances under which a use case is relevant. We introduced meaningful context by always attaching a goal to each use case as in [AMP94, Ant97, RSB98]; every use case title was thus expressed as a goal.

Lack of contextual information increases the risk that system requirements may be misinterpreted. When adequate context is not provided in a use case, valuable information may be lost. We incur the obvious risk of producing an incomplete or erroneous specification. Use cases are typically utilized to surface requirements early on and yet without the appropriate context the very benefits we expect are reduced, since it is likely that errors will surface later in the process from the misinterpretation.

The use case authors were not the intended users of the system

Since the SRS authors are not the intended end-users, they lacked implicit knowledge of the tasks that users expect to complete with the system. While the authors did elicit information from a sample customer base, these customers were included only in the initial stages of the SRS production. The actual intended users were later unable to contribute additional scenarios or examine the use cases for accuracy and validation. A more customer oriented approach, such as contextual design, would have been very helpful. In contextual design users are observed employing a system with the intent of ensuring the focus of the analysis is on users' tasks and objectives [BH98]. One would expect such an approach to result in the production of more directed due to increased stakeholder participation. Although several actors were cited according to the roles they fulfill in each use case, the SRS authors constructed the use cases from the perspective of one actor, a salesperson. Unfortunately, the other (actor) viewpoints were never considered or integrated into the SRS.

The lack of stakeholder involvement introduced specific risks throughout the requirements specification process. Alarmingly, the scenarios were never validated by system users; and we, thus, suspect the scenarios may be plagued due to assumptions made about typical and/or expected user and system interactions. This lack of validation is problematic [Dav93]. Since we know that exploring multiple viewpoints yields more comprehensive requirements coverage, the focus on a single viewpoint also introduces requirements coverage risks.

Traceability is difficult to manage

We incurred a significant amount of overhead due to our manually maintaining pre-traceability information [DP98, Ram98]. Each time a duplicate goal was reconciled/removed, we updated a list of the duplicate goal numbers as a minimal form of traceability. All goal data was captured in MS Excel spreadsheets which was particularly time consuming and awkward. We supported goal evolution using, for example, rudimentary manipulations (e.g., copy and move) to ensure retention of goal source information. Maintaining pre-traceability required dutiful attention during our analysis, but it can be greatly simplified with appropriate tool support.

Traceability is a measure of quality that reduces the risk of, for example, not propagating changes across lifecycle artifacts. Maintaining traceability information can be quite time consuming, thus it may be tempting to skimp in an effort to save time or money. However, manual traceability may result in the production of static documents, which often remain unmodified after their initial creation and as a result often become obsolete [Ram98]. Traceability reduces the risk of inconsistencies and ensures compliance with the SRS. As we discuss below inconsistencies in the SRS itself also made it more difficult to maintain such traceability.

Deriving use cases primarily from the current system's GUI focuses too much attention on design and implementation

The SRS was written under duress and a very tight delivery deadline. This perhaps caused the SRS authors to rely excessively on the available user interface as the basis for their use cases. Interestingly, the SRS authors were also responsible for designing the GUI. Unfortunately, one task seemed to taint the other as the use cases became a product of the screen design, not of implementation-independent functionality. The purpose of creating a use case inherently effects its style and content. Deriving use cases from GUIs rather than from the actual user goals and objectives yields too much implementation-specific detail that should be addressed during software design, not requirements engineering.

The SRS use cases were laden with details about specific design widgets, items should be included in a system's design. An excessive focus on a system's user interface in the corresponding use cases poses significant challenges during analysis since all GUI references must be extrapolated from the derived goals [Ant97]. Inconsistency across available screen designs also resulted in corresponding inconsistencies in the derived goals (these were later resolved, but could have been prevented they had been resolved during SRS production). Multiple use cases included statements pertaining to specific menus and screen navigation (e.g., "return to the main screen"). However, the return to main screen option was not always provided**. It is not clear if this option was erroneously included in the use cases, or erroneously omitted from the screen design. Either way, an inconsistency exists.**

The SRS use cases provide a system scope description of user and system interactions, not a description of interactions between subsystems. The inclusion of design information that describes subsystem interactions is acceptable but such information should not be recorded in system level use cases. Introducing design considerations prior to problem definition may corrupt the analysis process. The over-reliance existing screen designs, thus, introduces the risk of software design interfering with initial problem analysis. If design considerations are introduced prior to completing the SRS, the design may suffer. Since the screens and the scenarios are so tightly bound, the risk that the scenarios, the subsequently derived goals, and the screen design may be inconsistent increases as the design evolves [Lil99]. Such coupling between design and scenarios magnifies the effects associated with cost and scheduling. Consider a baselined GUI design which serves as the basis for use case construction; major requirements discovered after design baselining will presumably require significant modifications to the GUI. It is then likely that both the GUI and use cases would have to be reinvented; alternatively, the designer may incorporate changes into the design in less than optimal ways to avoid having to redesign the GUI. Or, as is often the case in feature driven systems, the requirements may unfortunately be changed to accommodate the design. Obviously, it is best to avoid such a situation entirely by not allowing implementation-specific details to creep into the use cases.

Missing and inconsistent naming of use cases is indicative of an incomplete and flawed specification

The list of included and extended use cases often pointed to undefined or nonexistent use cases. Some referenced use cases were never defined. Thus, to identify missing use cases, we created an "Includes Tree" to track relationships between use cases. This approach led us to discover 15 missing use cases. Referenced use cases were either completely missing or were simply named inconsistently. The use case tree was a simple, yet tedious, mechanism for surfacing these inconsistencies.

The risks associated with missing use cases are similar to those of use cases lacking context. Missing use cases obviously represent missing interaction information and is indicative of missing requirements. A simple use case name does not sufficiently describe the use case details that affect the derived requirements. Discrepancies between use case names introduces additional risk since use cases provide a way to reference a potentially large body of information using just a few words (its name) or a unique identifier, such as a number. Use cases referenced using the incorrect name also increases the likelihood for conflicts in both the goals and requirements.

5 Lessons Learned

Our analysis yielded several valuable lessons for practitioners and researchers alike.

Achievement goal categories lead to the derivation of a more complete set of goals and viewpoint analysis

It is valuable to separate user goals from system goals. Previously, when employing the GBRAM [Ant97], we made no real or clear distinction between `MAKE` and `ACHIEVE` goals. During this study, however, we expressed these goals much more explicitly. All user goals (those that express the users' objectives) were named with the keyword `ACHIEVE` while all system goals (those that express the system's response to the users' goals) were named with the keyword `MAKE`. For example, `<ACHIEVE quote requested>` is representative of a user task while `<MAKE quote retrieved>` represents the system's response to the user's goal. Although this distinction may seem unremarkably simple, it proved to be very valuable. This correspondence between the user and system goals enabled us to more clearly define the system boundaries and appropriately assign goal responsibilities. Each time we identified an `ACHIEVE` goal, we also systematically considered any possible, related `MAKE`, `PROVIDE` or `ALLOW` goals. Likewise, for each `MAKE` goal, we formally considered all the possible `ACHIEVE` goals.

This simple distinction between user and system goals allowed us to identify nearly twice as many functional requirements as we had during our previous goal-driven analyses [Ant96, Ant97, AP98a]. We attribute this to the more strict and methodical consideration of multiple viewpoints associated with the users' objectives and the system responses to enable those objectives. The resulting user goals afford the ability to construct a more formal use case diagram, while the system goals facilitate the construction of more complete and realistic interaction diagrams.

Domain specific goal classes help ensure better requirements coverage

The notion of reusable goal classes that occur in various types of software systems is discussed in [Ant97]. Of particular relevance to this study are the goal classes for an electronic commerce application: process support, electronic commerce, information

display and organization, and security and access control. Process support goals pertain to system processes enacted by the user or the system. Electronic commerce goals deal with the base functionality of the system. Information display and organization goals describe the organization and presentation of information by the system. Finally, security and access control goals involve limiting access to authorized users [AP98a].

One should expect to have all four of these goal classes represented in the goal set for any electronic commerce application. In this study, we identified 120 process support goals; the large number of process goals is not surprising given that the authors relied so heavily on the GUI. The process goals are natural products of this type of problem decomposition. Each goal from a user action translated into a process goal and each system response from a user action translated into a different process goal. We also identified 34 information and organization goals that were, typically, buried in the pre- and post-conditions of certain scenarios. As expected, we identified a fair number of electronic commerce goals (73); these goals emphasized such items as quotes, shopping carts, product ordering, notifications and confirmations. The non-electronic commerce goals described generic system functions that would be expected in any system.

The SRS does not clarify which types of users have what kinds of access to the system. Given that this is an electronic commerce application, this was the source of much concern. Only 8 security and access goals were identified and these solely considered the mechanism for user login. This suggests that the availability of goal classes can indeed be very beneficial when developing the requirements for systems since the goal classes can help ensure that all expected behaviors have been considered for the given system. Upon realizing that we had not derived a sufficient number of security and access control goals, we were able to further analyze the various kinds of system users so that the access levels could be defined.

Naming goals according to constraints aids in maintaining context

It is beneficial to include constraint information in a goal name. For example, the goal `<ACHIEVE quote selected>` had different constraints depending on the context in which the goal was situated. We refined this general goal by using its constraints to create the following two goals: `<ACHIEVE quote selected for editing>` and `<ACHIEVE quote selected for searching>`. The inclusion of this constraint information ensured that we did not lose or forget the constraint information during goal and scenario analysis when the auxiliary notes (e.g., constraints, obstacles, pre- and post-conditions) were not immediately visible. The system's response to each of these user goals will be different since selecting to edit will invoke a response such as displaying editable fields whereas selecting for searching will require the system to display the results of the search, with a list of selectable options.

Distinguishing between provision of capability and information goals yields a logical separation of concerns

According to Jackson, separating concerns is simply a matter of structuring a complex topic as various more simple topics that can then be considered separately [Jac95]. Many goals in this case study resembled the kinds of goals observed in the study reported in [AP98a]; they involve the provision of certain capabilities and functions or the provision of information to and from and various actors. The word `PROVIDE` is often used indiscriminately in use cases to express objectives involving furnishing capability and/or sharing information. We distinguished between these two

types of objectives by identifying the following types of goals during the course of this case study:

NOTIFY and INFORM goals involve the delivery or provision of some information to a given actor. Notification goals can, for example, involve an e-mail message that is sent or an error message that is displayed on the screen. In the electronic commerce and quotation system, the primary actor delivering the information is typically the system and the secondary actor, a human user, is the information recipient.

PROVIDE and *ALLOW goals* encompass some service, capability or function. Typical services or capabilities provided by an electronic commerce system include manipulation of ordering or quotation information, security services and searching capabilities.

This distinction between provision of capability and provision of information offers a clear separation of concerns for allocation of goal responsibilities.

A use case collection is not a suitable substitute for an SRS

A use case collection comprised the greater part of the electronic commerce and quotation system SRS. The SRS was produced under some duress, perhaps as a management directive. Although such circumstances are unfortunate, it is not an unusual or unheard of occurrence due to increased pressure for shortened product delivery windows or tightened budgets due to moratoriums on spending influenced by pressure to meet earnings estimates. Nonetheless, an SRS is a necessity and must comply with standards such as MIL-STD-498 for government contracts. An SRS must specify specific requirements as the conditions for its acceptance, stated with a unique identifier (for purposes of traceability), in such a way that each requirement's objective may be later tested. A use case is not a specific or clear statement of such objectives. Instead, a use case is a representation that simply summarizes behavior traces; it augments and clarifies our understanding of the requirements, but a use case does not a requirement make. What a use case is, is a valuable tool that aids stakeholders in imagining how a proposed system will support their work and aids analysts in developing a concrete understanding of the customer's needs as they identify hidden requirements.

Our study gives evidence of software practitioners adopting a use case collection as a suitable substitute for a software requirements specification. We find this practice concerning and worthy of focused attention and further investigation for the mentioned reasons and given the specific risks and challenges discussed throughout this paper.

7 Discussion and Future Work

Scenarios are valuable for eliciting information about system requirements, communicating with stakeholders and providing context for requirements [AMP94, Pot95, PTA94, RSB98, WPJ98]. In this paper, we present our experiences in the form of challenges faced and lessons learned while managing a large set of use cases during a goal-driven requirements specification effort. Most importantly, we provide evidence of the pitfalls associated with replacing a formal requirements specification with a collection of use cases.

Our study enabled us to identify a number of helpful techniques for extending the current heuristics found in the GBRAM [Ant97]. The study also yielded some rather interesting observations, leading us to analyze the risks associated with the various challenges we encountered. These risks, although not typical software project management risks, deserve our attention since they can potentially impact project schedules, costs and ultimately product quality. Our investigation provided us with the

opportunity to further validate the heuristics provided in [Ant97] and we have provided several examples of their successful application throughout this paper (e.g., the consideration of system-specific goals to determine their underlying intent as well as the identification and resolution of simple inconsistencies). Traceability was both a priority and a challenge throughout this investigation. We expect to assuage some of these traceability issues by providing appropriate tool support for scenario management [AAB99].

Acknowledgements

The authors wish to thank the sponsors of this project, the individual stakeholders who participated in goal elicitation interviews; Thomas Alspaugh and Aldo Dagnino for their comments on drafts of this paper; and Michael Rappa for partial support of this work via the NCSU electronic commerce initiative. This research was made possible due to the co-authors' collaboration with the sponsor in 1999.

References

[AAB99] Alspaugh, T.A., A.I. Antón, T. Barnes, and B. Mott. An Integrated Scenario Management Strategy, *International Syposium on Requirements Engineering (RE'99),* Limerick, Ireland, pp. 142-149, June 1999.

[AMP94] Antón, A.I., W.M. McCracken, and C. Potts. Goal Decomposition and Scenario Analysis in Business Process Reengineering, *Advanced Information Systems Engineering, 6th International Conference Proceedings (CaiSE'94),* Utrecht, The Netherlands, pp. 94-104, 6-10 June 1994.

[Ant96] Antón, A.I. Goal-Based Requirements Analysis, *International Conference on Requirements Engineering (ICRE `96),* Colorado Springs, Colorado, USA, pp. 136-144, April 1996.

[Ant97] Antón, A.I. Goal Identification and Refinement in the Specification of Software-Based Information Systems, Ph.D. Thesis, Georgia Institute of Technology, Atlanta, GA, June 1997.

[AP98a] Antón, A.I. and C. Potts. The Use of Goals to Surface Requirements for Evolving Systems, in *International Conference on Software Engineering (ICSE '98),* Kyoto, Japan, pp. 157-166, 19-25 April 1998.

[AP98b] Antón, A.I. and C. Potts. A Representational Framework for Scenarios of Systems Use, *Requirements Engineering Journal*, Springer Verlag, 3(3-4), pp. 219-241, 1998.

[BH98] Hotzblat K. and H. Beyer, *Contextual Design*., Morgan Kaufmann, San Francisco, CA, 1998.

[BRJ99] Booch, G., J. Rumbaugh and I. Jacobson, *The Unified Modeling Language User Guide*, Addison Wesley, 1999.

[Dav93] Davis, A.M. *Software Requirements: Objects, Functions, & States,*. Prentice-Hall, 1993.

[DP98] Dömges, R., Pohl, K., Adapting Traceability Environments to Project-Specific Needs, *Communications of the ACM,* 41(12), pp. 54-62, December 1998.

[Jac92] Jacobson, I. et. al. *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, 1992.

[Jac95] Michael Jackson. *Software Requirements and Specifications*. Addison-Wesley, 1995.

[JBC98] Jarke, M., X.T. Bui and J.M. Carroll. Scenario Management: An Interdisciplinary Approach *Requirements Engineering Journal*, Springer Verlag, 3(3-4), pp. 154-173, 1998.

[Lil99] Lilly, Susan. Use Case Pitfalls: Top 10 Problems from Real Projects Using Use Cases, *Proceedings Technology of Object-Oriented Languages and Systems*, pp.174-183, 1-5 August 1999.

[MMM98] Maiden, N., S. Minocha, K. Manning and M. Ryan. CREWS-SAVRE: Systematic Scenario Generation and Use, *International Conference on Requirements Engineering (ICRE'98),* pp. 148-155, April 1998.

[Pot95] Colin Potts. Using Schematic Scenarios to Understand User Needs, in *Symposium on Designing Interactive Systems: Processes, Practices, Methods and Techniques*, pages 247-256, University of Michigan, Ann Arbor, Michigan, USA, August 1995.

[Pot99] Potts, C. A ScenIC: A Strategy for Inquiry-Driven Requirements Determination, *Proceedings IEEE 4th International Syposium on Requirements Engineering (RE'99*), Limerick, Ireland, 7-11 June 1999.

[PTA94] Potts, C., K. Takahashi, and A. Antón. Inquiry-Based Requirements Analysis, *IEEE Software*, 11(2), pp. 21-32, March 1994.

[Ram98] Ramesh, B. Factors Influencing Requirements Traceability Practice, *Communications of the ACM*, 41(12), pp. 37-44, December 1998.

[RSB98] Rolland, C., C. Souveyet, and C. Ben Achour, Guiding Goal Modeling Using Scenarios, *IEEE Transactions on Software Engineering*, 24(12), pp. 1055-1071, December 1998.

[WPJ98] Weidenhaupt, K., K. Pohl, M. Jarke and P. Haumer. Scenarios in System Development: Current Practice, *IEEE Software*, 15(2), March/April 1998.