Managing Use Cases During Goal-Driven Requirements Engineering: Challenges Encountered and Lessons Learned

Annie I. Antón, John H. Dempster, Devon F. Siege
College of Engineering
North Carolina State University
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. Given the challenges inherent in managing large collections of scenarios, this shortcut is cause for concern and deserves focused attention. We discuss our experiences with a goal-driven analysis of a requirements specification for an electronic commerce application for a large international company. We describe scenario management within the context of this goal-driven requirements analysis effort. 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 during goal and scenario analysis.

Keywords

Scenario management, use cases, requirements engineering, goals, requirements traceability, electronic commerce.

I. 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, Fow97] for systems engineering. UML is based on the object-oriented approaches of Booch [Boo94], Jacobson [Jac92] and Rumbaugh [Rum91] and has engaged so much interest that it now boasts its own conference. 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 also widely used in other areas of system development, including Human Computer Interaction (HCI), software engineering and organizational process design [AP98b]. Jarke et. al. also discuss the lack of a unified research framework for scenario management despite the various disciplines in which scenarios have received widespread attention including HCI, strategic management and software engineering [JBC98]. Scenarios are used purposefully to "stimulate thinking" in all three disciplines [JBC98]. HCI scenarios support the specification of user interfaces and strategic management scenarios help with the exploration of alternative futures. In software engineering, scenarios are used to gather and validate requirements [AP98b, JBC98]. Scenarios are also a reasonable approach for managing change during the software process. For example, evolutionary scenarios are used to envisage how a system itself may change due to, for instance, political or technological discontinuities [AP98b]. While scenarios are useful for managing change and evolution, managing scenario traceability across multiple changes becomes increasingly difficult.

The state of scenario management in practice was reported in [WPJ98]. In this study, the use of scenarios was examined in 15 European projects to learn how scenarios were produced and utilized as well as to identify the benefits and problems associated with scenario usage in industrial settings. As documented in [WPJ98], practitioners using scenarios and/or use cases in industrial settings incur very specific challenges. Specifically, Weidenhaupt et.al. highlight several key areas needing support including: the need for appropriate process

Submitted to IEEE International Conference on Software Engineering (ICSE 2000) Limerick, Ireland

guidance as well as comprehensive support for managing both scenario traceability and evolution.

Researchers at North Carolina State University (NCSU) are developing scenario management strategies [AAB99] in an effort to address the challenges discussed in [WPJ98]. The NCSU scenario management strategies support evolution by employing shared scenario elements to identify and maintain common episodes among scenarios. Measures are used to quantify the similarity between scenarios, serving as heuristics that provide process guidance to practitioners in finding, for example, duplicate scenarios, scenarios needing further elaboration or those that may have been previously overlooked. Tool support is under development, but is not yet available. In the interim, we are engaging in requirements specification activities involving the analysis of large collections of scenarios and use cases, to deepen our understanding of the challenges our scenario management tool must address.

This case study was motivated by our desire to observe scenario management in an industrial setting while applying goal-driven requirements analysis methods. This paper reports on our experiences in managing a large collection of use cases during our requirements specification activities for an electronic commerce application. In Section II we discuss relevant work in goals and scenarios for requirements specification. Section III focuses on the case study as well as the challenges and associated risks incurred. The lessons learned are detailed in Section IV, followed by discussion of future work in Section V.

II. RELATED WORK

The terms use cases and scenarios mean different things to different people; we thus discuss these terms in the context of other relevant related work including the goal-based requirements engineering literature.

A. Scenarios and Use Cases

Scenarios aid analysts and stakeholders in developing an understanding of current or envisaged systems and business processes [AMP94, Ant96, Ant97, AP98a, AP98b, BL98, Jac92, Pot95, WPJ98]. They describe concrete system behaviors by summarizing behavior traces of existing or planned systems. Use cases, introduced by the object-oriented community [BRJ99, Fow97, 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.

A representational framework for scenarios and use cases appears in [AP98b]. In the human computer interaction (HCI) community scenarios have been used to improve communication between end-users and developers for designing user interfaces, task modeling and prototyping. Requirements engineering benefits from an initial emphasis on use cases, but they benefit in turn from a semantics that connects them to purposeful activities [Pot95, Pot99].

B. Goals and Scenarios

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. Since goals are evolutionary, they provide a common language for analysts and stakeholders. 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. Furthermore, since goals are typically more stable than requirements [Ant97], they are a beneficial source for requirements derivation. Goals are operationalized and refined into requirements and point to new, previously unconsidered scenarios. Similarly, scenarios also help in the discovery of goals [AMP94, AP98a, JBC98, Pot99, RSB98]. Although the merits and benefits of scenario-based and goal-based analysis in requirements engineering are well understood, researchers are now faced with the question of how to use scenarios and goals in a complimentary fashion. Several approaches, which we briefly discuss, do show promise [AAB99, MMM98, RSB98].

Organizing goals hierarchically provides a useful way to represent the relationships between goals and subgoals so that we can reason about those relationships [AMP94, DvL93]. The Goal-Based Requirements Analysis Method (GBRAM) [Ant96, Ant97, AP98a], uses a goal topography to structure and organize such requirements information as scenarios, goal obstacles, and constraints. These topographies support analysts in finding and sorting goals into functional requirements while scenarios help in documenting issues, surfacing new goals and elaborating requirements. Goal hierarchies offer a useful way to visualize goals and their related scenarios [Ant97, ALR96] as does the previously discussed scenario management strategy [AAB99]. CREWS-SAVRE also organizes scenarios hierarchically according to goals and goal obstacles; the goals serve as a grounded, shared understanding for stakeholders [MMM98]. Finally, goal-scenario coupling, as documented in [RSB98, RGK99], provides an integrative approach to goal and scenario-oriented requirements analysis. The CREWS-L'Ecritoire approach employs bi-directional coupling to facilitate navigation between goals and their associated scenarios.

III. ANALYSIS OF THE ELECTRONIC COMMERCE AND QUOTATION SYSTEM

The company with which we collaborated on this effort 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. The processes for these two activities were identified as areas for improvement. 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. The variety of methods used by sales people qualifies the process as a candidate for process improvements. Such desired improvements include producing consistent quotations and order pricing as well as more adequate means for tracking statistical information to reveal important market trends. A new, more tightly integrated system is needed to facilitate the provision of consistent lowest prices and to aid the company's distributed sales force by providing a more streamlined bidding process. An electronic commerce, company-wide intranet system was required to manage the quotation and ordering process for product bidding. We now provide an overview of our goal-driven analysis of the system and discuss the challenges encountered during the case study.

A. Goal Analysis Efforts

A goal-based approach [Ant97] was adopted in an effort to surface requirements for the redesign of the electronic commerce quotation and ordering system. During our inquiry-driven analysis [PTA94], 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. Since we lacked tool support to record analysis artifacts (e.g., goals, agents, constraints, etc.), we employed Microsoft Excel to produce a spreadsheet workbook similar to those produced in previous case studies [Ant97, AP98a]. The Excel sorting capability facilitated the identification, ordering as well as the reconciliation of synonymous and redundant goals. Since traceability was an objective throughout the study, we tracked and documented any changes to the goals and the associated rationale, such as when they were merged and/or removed, as well as the origin of the goal. Naturally, since this was a manual process, we also took measures to ensure that we did not lose any auxiliary notes associated with each of these 'modified' goals [Ant97].

Our analyst team was provided with a Software Requirements Specification (SRS) that contains 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. For the remainder of this paper, we use the term 'authors' to refer 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.

Figure 1 shows the basic structure for a typical SRS use case. Each of the 52 use cases in the SRS includes the following information:

- a unique identifier;
- a title;
- 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.

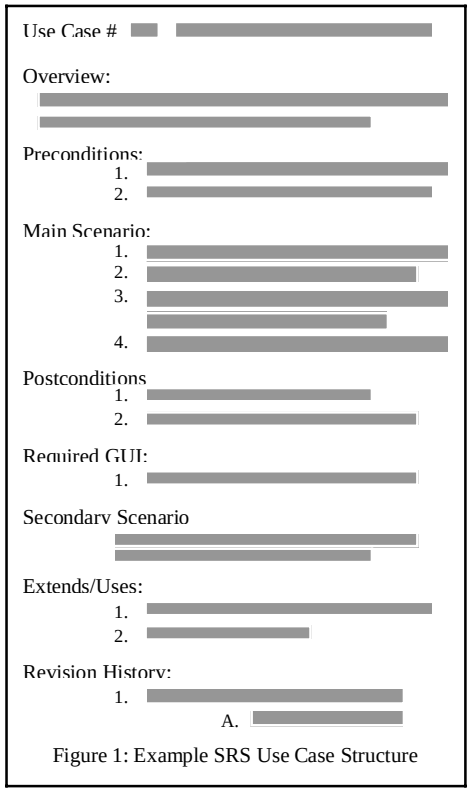

Figure 1: Example SRS Use Case Structure

A main scenario is an ordered list of user interactions with the system. The primary difference between a main and secondary scenario is that a main scenario is normative while secondary scenarios show possible branching and 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 analysts met for sessions ranging from one to three hours in duration, once a week for two months. Prior to each weekly meeting, individual analysts performed a goal analysis of agreed upon use cases that we then discussed

and revised while collaboratively recording all goals and auxiliary notes. Our meeting preparations allowed us to concentrate primarily on revising and extending the original analyses during our meeting sessions. During these sessions we applied the GBRAM [Ant96, Ant97, AP98a]. Our analysis efforts initially generated an initial total of 292 goals. While identifying these goals we struggled due to five specific challenges posed by the collection of use cases with which we were working. These challenges, and the associated risks incurred, are discussed in the following subsections.

B. Challenges Encountered and Associated Risks

During this case study we faced various challenges which required unexpected extra work on the part of the analysts and introduced additional risk into the requirements specification:

1) 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 the given use case was intended to describe; this is perhaps partially due to the way individual authors write use cases. On numerous occasions we, the analysts, had to refer to other use cases to deepen our understanding of a given use case. There are various possible reasons for this. First, while the analysts are all very familiar with the domain of electronic commerce, there were some use cases that dealt with company-specific practices and policies (e.g., underlying business rules which had not been clearly articulated; use case authors had the benefit of this tacit knowledge). Second, although each use case in the SRS included a brief overview, the overviews did not on average provide appropriate contextual information to describe the circumstances under which the use case is relevant, as advocated in [Lil99]. Context makes use cases more meaningful; thus, use case authors should seek to make use cases understandable to all stakeholders (including the authors themselves). We chose to provide this context by always attaching a goal to each use case as in [AMP94, Ant97, RSB98].

Lack of contextual information increases the risk that system requirements may be misinterpreted. Context makes use cases more meaningful and when adequate context is not provided, valuable information about the use cases may be lost. We, thus, 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 result later in the process from the misinterpretation.

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

The authors of the SRS are not the intended end-users, thus they lack implicit knowledge of the tasks that users expect to complete with the system. The authors did elicit information from a sample customer base; however, these customers were included only in the initial stages of the SRS production. Since, the actual use cases were not written in participatory fashion, intended users were later unable to contribute additional scenarios or examine the use cases for accuracy and validation. Instead, the SRS authors relied excessively on the available user interface designs to construct the use cases, producing a collection of use cases that focused more on the system's GUI rather than the intended functionality of the system.

Although the authors initially involved the users of the system in the collection of data for the SRS, a more customer oriented approach, such as contextual design, would have been very helpful. In contextual design the actual users are observed using the system with the intent of ensuring the focus of the analysis is on users' tasks and objectives [BH98]. Hence, one would expect more directed use cases to be produced given more stakeholder involvement.

Additionally, the authors constructed the use cases from the perspective of one actor, a salesperson, even though other actors were cited according to the role(s) they supposedly played in each use case. Unfortunately, these other viewpoints were never considered or integrated into the specification.

In light of the lack of user participation throughout the requirements specification process, we identified a number of potential project risks. Since the scenarios were developed by the authors and were not validated by system users, we suspect the scenarios provide some misinformation about typical and/or expected user and system interactions. This lack of validation is problematic during product design and development [Dav93]. The fact that the authors only investigated scenarios from one actor's viewpoint is also an area of potential risk, since exploring into multiple viewpoints yields more comprehensive coverage of the requirements.

3) Traceability is difficult to manage

Due to the lack of requirements management tool support, a significant amount of overhead was incurred due to our having to manually maintain pre-traceability information including the source and origin of each goal [DP98, Ram98]. When we removed duplicate goals, as in GBRAM [Ant97], we maintained a list of all the duplicate goal numbers as a minimal form of traceability. When additional traceability information was needed we had to refer to previous versions of the spreadsheet to determine the true source of a given goal or scenario. This was particularly time consuming and awkward. We maintained this source information manually using rudimentary manipulations (e.g., copy and move) to the new spreadsheet. While maintaining pre-traceability requires dutiful attention, 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 reduce or skimp on traceablility efforts in order to save time or money. Additionally, any manual processes (as were our traceability efforts throughout this study) are always subject to human error. Manual traceabilty techniques may also result in the production of static documents. These static documents often remain unmodified after their initial creation and as a result often become obsolete [Ram98]. Traceability must be implemented to reduce the risk of inconsistencies and ensure compliance with the requirements specification. Introducing additional traceability can affect cost and scheduling estimates. However, while traceablity may require an initial investment of time and money, the gains and benefits well outweigh the costs [Ram98].

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

The purpose of creating a use case dictates and effects its style and content. The analyzed SRS gives the appearance of having been written in retrospect, possibly as a process requirement dictated by management. Additionally, as previously mentioned, the SRS authors relied excessively on the available user interface to construct their use cases, producing a collection of use cases that emphasized the expected user interface rather than the intended functionality of the system. Perhaps not coincidentally, 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 production of use cases concentrated primarily on GUI specific features, and in this case, reflects the perspective of those individuals responsible for the GUI production, instead of the perspective of the individuals who use (or will use) the system.

Deriving use cases from GUIs rather than from the actual user goals and objectives yields too much implementation specific detail which should be addressed during software design, not requirements engineering [Lil99]. The use cases in the SRS were laden with details about specific design widgets, items that are inherent to, and should be included in a system design. The GBRAM includes heuristics for goal refinement, one of which indicates that any goals based on system-specific entities should be restated without including system-specific information [Ant97]. A misdirected GUI use case focus poses significant challenges since, according to the GBRAM goal refinement heuristics, all GUI references must be extrapolated from the goals that were derived from these GUI-specific use cases. In the following subsection we further discuss how this heuristic affected goal evolution in our study.

Inconsistency across available screen designs also results in corresponding inconsistencies in the derived goals. For example, 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 on the screen layout. 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 although they may be considered irrelevant since these details should not be included in a use case unless the purpose of the use case is specifically to illuminate design issues pertaining to the user interface.

The use cases in the SRS provide a system scope description of user and system interactions, not a description of interactions between subsystems. The inclusion of design information in use cases which describe subsystem interactions is acceptable but such information should not be recorded in system level use cases [Lil99]. Instead, as suggested in [Lil99], design information discovered during requirements analysis should be placed in a separate "Design Guidance" document.

According to [EJW95], design considerations may be equated to problem solutions, and introducing these solutions prior to the problem definition may corrupt the analysis process. The over-reliance of the use cases on a GUI, thus, introduces the risk of software design interfering with the analysis process. If design considerations are introduced prior to the completion of the requirements specification, the design of the product may suffer. Since the screens and the scenarios are so tightly bound, the risk that the scenarios (and the subsequently derived goals) and the screen design may be inconsistent increases as design modifications are introduced [Lil99]. Such coupling between design and scenarios can increase the effects of traditional software engineering risks associated with cost and scheduling.

Consider a baselined GUI design which serves as the basis for use case construction during requirements engineering activities. If a major requirement is discovered after the design is baselined it may require significant modifications to the GUI. It is then likely that both the GUI and the use cases would have to perhaps be reinvented; alternatively, the designer may need to incorporate changes into the design in less than optimal ways to avoid having to redesign the GUI. Obviously, it is best to avoid such a situation entirely by not allowing implementation-specific details to creep into the use cases.

5) 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. Additionally, some use cases that were referenced by other use cases were never defined. In order to identify the missing use case names, we created an "Includes Tree" to show the relationship between all the use cases and discovered 15 references to missing use cases. Figure 2 shows a portion of a use case tree in which all referenced cases are defined. In contrast, Figure 3 shows a use case tree in which use cases referenced do not exist. The use cases referenced could actually be missing use cases, or may be using inconsistent names when referencing other use cases.

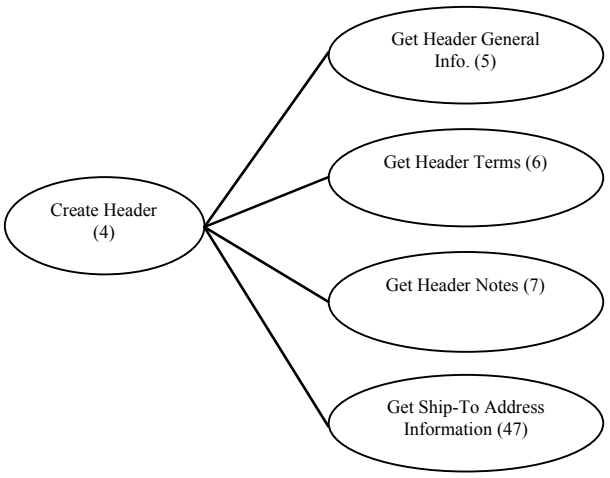

FIGURE 2: SRS Use Case "Includes Tree"

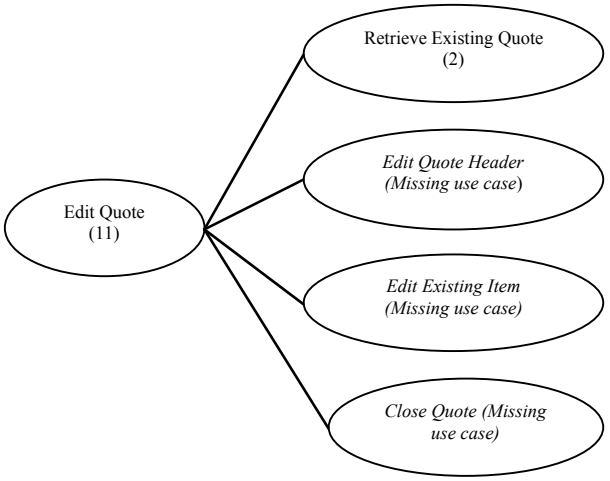

FIGURE 3: SRS Use Case "Includes Tree" with Missing Use Cases

The risks associated with missing use cases are similar to those of use cases lacking context. A missing use case, while it provides a use case name, obviously represents missing interaction information that in turn suggest missing requirements. A simple use case name is not a sufficient description of all the details of a use case which affect the associated derived requirements.

Discrepancies between use case names also introduce an element of risk since use cases provide a way to reference a potentially large body of information using just a few words (the use case name) or a unique identifier, such as a number. When use cases are referenced using the incorrect name it increases the likelihood for conflicts.

C. Goal Evolution

The goal analysis process began by identifying goals in the main and secondary scenarios. Goals and their associated information were identified, numbered, classified and stored. The information tracked included the goal number, responsible agent, the use case from which the goal was derived, any constraints identified for the goal, pre- and post-conditions, as well as any issues, rationale or questions related to the goal. Documenting this information provides both pre-traceability and the preservation of information that will be used later to discover new goals and requirements.

As preparation for each meeting session, goals were identified from predetermined scenarios by each of the analysts. During the sessions we reviewed the goals while documenting any newly identified goals. This process improved the accuracy of the goals by inserting a level of review and inspection prior to the recording of each goal. It was in these reviews that the problems associated with management of the goals and scenarios began to take shape.

Figure 4 shows the evolution of the goal set. The shaded boxes represent those goals which were 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.

Goals were refined by applying the GBRAM heuristics [Ant97], resulting in the elimination of 100 goals. Not surprisingly, 70 GUI-specific goals were identified. Although these goals contain important information for usability purposes, we ignored them for purposes of defining the system requirements since they are implementation-specific [Ant97]. The 70 goals were deleted, however 12 of them were suggestive of underlying processes or activities so we reformulated those goals by restating them. These restated goals convey implementation-independent ideas taken from the extracted goals. Here the categorization enriched the data set by identifying implementation-specific goals and either replacing or deleting them from the goal set.

Goals were named using meaningful keywords selected from a predefined set of goal categories [Ant97]. The boxes in the ellipse in Figure 4 show the number of goals named with each of these keywords. In Section IV we provide further discussion of ACHIEVE, MAKE, NOTIFY, INFORM, PROVIDE and ALLOW goals.

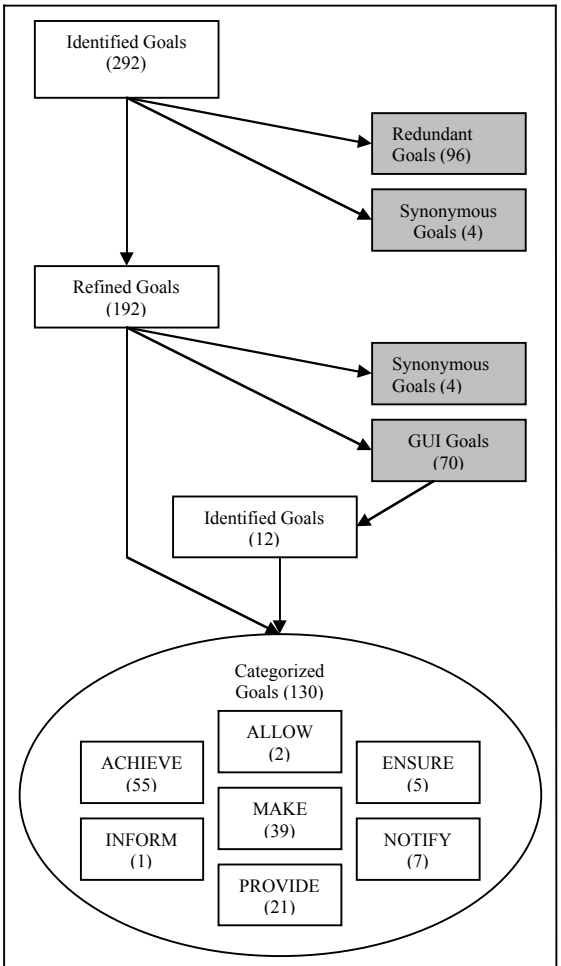

FIGURE 4: Goal Evolution Diagram

IV. LESSONS LEARNED

Despite the challenges encountered, our analysis yielded several lessons learned which we believe to be of interest to practitioners and researchers alike.

1) Achievement goal categories lead to the derivation of a more complete set of goals

We found it valuable to separate user goals from system goals by expressing achievement goals accordingly. Previously, when employing the GBRAM [Ant97], we made no real or clear distinction between MAKE and ACHIEVE goals. During this case study, however, we chose to express these goals as follows: 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 a very valuable technique. This correspondence between the user and system goals helped us 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 helped us identify nearly twice as many functional requirements as we had during 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.

2) Domain specific goal classes can 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 case study are the goals for an electronic commerce application, classified according to subject matter, as reported in [AP98a]. These electronic commerce goal classes are:

- process support goals;
- electronic commerce goals;
- information display and organization goals; and
- security and access control.

Process support goals describe goals that 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 describe those goals involved in limiting access to authorized users [AP98a]. It should be noted that these categories are not mutually exclusive. The same goal can be, and often is, classified according to more than of these goal classes. For example, in the CommerceNet case study,

the goal <KNOW what user has looked at previously> was classified as both an information display and an organization goal as well as a security and access control goal.

One should expect to have all four of these goal classes represented in the goal set for any electronic commerce application. In this study, 120 process support goals were identified. 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. Additionally, each system response from a user action translated into a different process goal. We also identified 34 information and organization goals which were, typically, buried in the pre- and post-conditions of certain scenarios. More in depth analysis is still required to determine how the customers intend to logically organize information within this intranet application. As expected, we identified a fair number of electronic commerce goals; the 73 electronic commerce goals emphasized such items as quotes, shopping carts, product ordering, notifications and confirmations. The non-electronic commerce goals described generic system functions that could be expected in any system.

Interestingly, the SRS does not clarify which types of users have what kinds of access to the system. Given that this system is an electronic commerce application, we found this particularly alarming. There were only 8 security and access goals identified and these solely considered the mechanism for user login. This observation 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.

3) Naming goals according to constraints aids in maintaining goal context

At times we found it 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 in the goal name 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.

4) 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. It was clear that we needed a better way to distinguish between these two types of objectives and we did so 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. Consider the goal <NOTIFY user that no matching item exists>; in this goal the primary actor (the system) provides the secondary actor (the user) with the information (that a matching item does indeed exist). Likewise, the goal <INFORM user of similar quotes> describes the system goal for providing the user with information concerning the existence of matching items.

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. For example, the goal <PROVIDE saving capability> provides a secondary actor (potentially the user) with the capability to save some unidentified piece of relevant information to the electronic commerce system. Likewise, the goal <ALLOW user logged on> permits the secondary actor (e.g., user) to utilize the service which monitors access control.

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

5) A collection of use cases is not a suitable substitute for a requirements specification

A collection of use cases comprised the greater part of the electronic commerce and quotation system SRS. As previously mentioned, the SRS appeared to have been produced perhaps under some duress as a management directive. Although such circumstances are unfortunate, it

is not an all together unusual or unheard of occurrence in many companies today due to increased pressure for shortened product delivery windows or, for example, tightened budgets due to moratoriums on spending influenced by pressure to meet earnings estimates. Nonetheless, a requirements specification is a necessity and must comply with standards such as MIL-STD-498 for government contracts. A software requirements specification 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 which simply summarizes behavior traces; a use case 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 as they develop a concrete understanding of the customer's needs and as they identify hidden requirements. The various available use case representations [AP98b, BRJ99] are expressive to both practitioners and stakeholders, allowing stakeholders to clearly relate their experiences to validate the system requirements and specifications.

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

V. DISCUSSION AND FUTURE WORK

Scenarios are proven to be valuable for eliciting information about system requirements, communicating with stakeholders and providing context for requirements [AMP94, Pot95, PTA94, RSB98, WPJ98]. Although valuable, scenarios together with their associated use cases, can be difficult to manage. This explains why scenario management is receiving increased attention among researchers in the software engineering community [AAB99, JBC98, 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 employing a collection of use cases as a replacement for a more formal requirements specification.

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. These observations led us to analyze the risks associated with various challenges encountered during our analysis. We believe 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]. We are currently collaborating with a telecommunications company, to evaluate requirements management technologies specifically with respect to how they support requirements traceability. This work will naturally serve to ground our findings so that we can appropriately define guidelines for scenario and episode traceability in our scenario management tool [AAB99].

Our immediate plans also involve further investigation of another issue that we have not adequately addressed herein. It became apparent during our study that the GBRAM would benefit from the addition of specific heuristics to guide analysts in identifying, resolving and managing inconsistencies. To this end, we plan to apply some of the findings in [vLD98] to further extend and refine the GBRAM [Ant97] in this way.

Although not discussed in this paper, episodes play a significant role in effective scenario management [PTA94]. Episodes are sequences of events shared by two or more scenarios [AAB99]. We can manage episodes by distinguishing events and recognizing those that are identical; two events are said to be identical when they share the same actor and action. Identical event sequences, or episodes, across multiple use cases was a common occurrence in this investigation. The SRS authors frequently reused event sequences applicable to various situations. For example, one sequence, classified as a secondary scenario in the SRS, was repeated 17 times throughout the SRS, causing the analysts to repeatedly review the same information. Our scenario management tool, currently under development, will automatically identify such shared sequences, while providing traceability across episodes and scenarios.

VI. ACKNOWLEDGEMENTS

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

the sponsor during the Summer of 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.

[ALR96] Antón, A.I., E. Liang and R. Rodenstein. A Web-Based Requirements Analysis Tool, in *5th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pp. 238-243, June 1996.

[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.

[BI96] Boehm, B. and H. In. Identifying Quality Requirement Conflicts, *IEEE Software*, pages 25-35, March 1996.

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

[BL98] Breitman, K.K. and J.C. Leite. A Framework for Scenario Evolution, *International Conference on Requirements Engineering (ICRE'98)*, Colorado Springs, CO, pp. 214-221, 6-10 April 1998.

[Boo94] Booch, G. *Object-Oriented Analysis with Applications*. 2nd edition, Benjamin/Cummings, Redwood City, CA, 1994.

[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.

[DvL93] Dardenne, A., A. van Lamsweerde and S. Fickas. Goal-Directed Requirements Acquisition, *Science of Computer Programming*, 20(1-2), pp. 3-50, April 1993.

[EJW95] Embley, D.W., R.B. Jackson and S.N. Woodfield. OO Systems Analysis: Is it or isn't it?, *IEEE Software* 12(4), pp. 19-33, July 1995.

[Fow97] Fowler, Martin. *UML Distilled: Applying the Standard Object Modeling Notation*, Addison-Wesley, 1997.

[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.

[Kar96] Karlson, J. Software Requirements Prioritizing, In *Proc. 2nd International Conference on Requirements Engineering (ICRE '96)*, pages 110-116, Colorado Springs, Colorado, USA, April 1996.

[KR96] Karlson, J. and K. Ryan. Prioritizing Software Requirements in an Industrial Setting, In *IEEE International Conference on Systems, Man and Cybernetics*, Vol. 2, pages 953-958, 1996.

[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.

[Rum91] Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy and W. Lorenzen. *Object-Oriented Modeling and design*. Prentice-Hall, Englewood Cliffs, NJ, 1991.

[vLD98] van Lamsweerde, A., Darimont, R., Letier, E. Managing Conflicts in Goal-driven Requirements Engineering, *IEEE Transactions on Software Engineering*, 24(11), pp. 908-926, November 1998.

[vLDM95] van Lamsweerde, A. and Darimont, R. and Massonet, P. Goal-Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt, *Proceedings 2nd International Symposium on Requirements Engineering (RE`95)*, York, UK, pp. 194-203, March 1995.

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