Deriving Goals from a Use-Case Based Requirements Specification

Article *in* Requirements Engineering · February 2001

DOI: 10.1007/PL00010356 · Source: DBLP

5 authors, including:

Annie I. Antón
Georgia Institute of Technology
150 PUBLICATIONS **5,004** CITATIONS

SEE PROFILE

Aldo Dagnino
North Carolina State University
43 PUBLICATIONS **541** CITATIONS

SEE PROFILE

Requirements Eng (2001) 6:63–73

Requirements Engineering

Deriving Goals from a Use-Case Based Requirements Specification

Annie I. Antón[a], Ryan A. Carter[a], Aldo Dagnino[b], John H. Dempster[a] and Devon F. Siege[a]
[a]College of Engineering, North Carolina State University, Raleigh, North Carolina, USA; [b]Asea Brown Boveri Inc., Power T&D Company, Raleigh, North Carolina, USA

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 at 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 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 during goal and scenario analysis

Keywords: Goal-based requirements engineering; Requirements specification; Use cases

1. Introduction

Scenarios have increased in popularity among software engineers due, in part, to Jacobson's use case approach [1] and the more recent introduction of the Unified Modeling Language (UML) [2,3] for systems engineering. UML is based on the object-oriented approaches of Booch [4], Jacobson [1] and Rumbaugh [5] and has engaged so much interest that it now boasts its own conference. The availability of UML and the associated tool support have made scenario and use case analysis even more accessible to requirements analysts and practitioners.

Scenarios are used purposefully to 'stimulate thinking' in various disciplines, including human–computer interaction (HCI), strategic management and software engineering [6]. In HCI, scenarios are used to improve communication between end-users and developers for designing user interfaces, task modelling and prototyping, supporting the specification of user interfaces. In strategic management, scenarios help in exploring alternative futures. In software engineering, scenarios are used to gather and validate requirements [6,7]. Scenarios are also a reasonable approach for managing change and evolution during the software process; however, managing scenario traceability across multiple changes becomes increasingly difficult. For example, evolutionary scenarios are used to envisage how a system itself may change due to, for instance, political or technological discontinuities [7].

The state of scenario management in practice was reported in Weidenhaupt et al. [8]. In this study, the use of scenarios was examined in 15 European projects to learn how scenarios were produced and utilised as well as to identify the benefits and problems associated with scenario usage in industrial settings. Practitioners using scenarios and/or use cases in industrial settings incur very specific challenges. Specifically, as reported in Weidenhaupt [8], several key areas needing support include 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. For this particular study, we employed the goal and

Correspondence and offprint requests to: Annie I. Antón, North Carolina State University, College of Engineering, 1010 Main Campus Drive, Raleigh, NC 27695-7534 USA. Email: anton@csc.ncsu.edu

scenario identification and elaboration heuristics available in the GBRAM (Goal-Based Requirements Analysis Method) [9,10]. This paper reports on our experiences in managing a large collection of use cases during the requirements specification activities for an electronic commerce application. In Section 2, we discuss relevant work in goals and scenarios for requirements specification. Section 3 describes the case study by focusing on goal analysis and evolution throughout our investigation. Section 4 discusses challenges encountered and provides a risk analysis. The lessons learned are detailed in Section 5, followed by discussion of future work in Section 6.

2. 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.

Scenarios aid analysts and stakeholders in developing an understanding of current or envisaged systems and business processes [1,7–14]. They describe concrete system behaviours by summarising behaviour traces of existing or planned systems. Use cases, introduced by the object-oriented community [1–3], 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 Antón and Potts [7]. In the HCI community, scenarios have been used to improve communication between end-users and developers for designing user interfaces, task modelling and prototyping. Requirements engineering benefits from an initial emphasis on use cases, but use cases benefit in turn from a semantics that connects them to purposeful activities [14,15]. 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 systems. 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 [10], they are a beneficial source for requirements derivation. Goals are operationalised and refined into requirements and point to new, previously unconsidered scenarios. Similarly, scenarios also help in the discovery of goals [6,11,12,15,16]. 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 complementary fashion. Several approaches, which we briefly discuss, do show promise [16–18].

Organising goals hierarchically provides a useful way to represent the relationships between goals and subgoals so that we can reason about those relationships [11,19]. The GBRAM [9,10,12] uses a goal topography to structure and organise 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 visualise goals and their related scenarios [10,20] as does the previously discussed scenario management strategy [17]. CREWS-SAVRE also organises scenarios hierarchically according to goals and goal obstacles; the goals serve as a grounded, shared understanding for stakeholders [18]. Finally, goal-scenario coupling, as documented in [16,21], provides an integrative approach to goal and scenario-oriented requirements analysis. The CREWS-L'Ecritoire approach employs bidirectional coupling to facilitate navigation between goals and their associated scenarios.

For this requirements specification effort, we employed the GBRAM in which goals are operationalised and refined into requirements and point to new, previously unconsidered scenarios (for a detailed explanation of the GBRAM and its heuristics see Antón [10]).

3. Electronic Commerce and Quotation System Analysis

The GBRAM [10] was employed to analyse an existing requirements specification for an electronic commerce, company-wide intranet system to manage the quotation and ordering process for product bidding. This section provides an overview of the Euronet system and our goal analysis efforts and summarises the evolution of goals throughout our study.

The company with which we collaborated has numerous plants throughout the Americas, Asia and Europe that manufacture a variety of engineering 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 salesperson and/or each plant having a different process, using various computer systems, printed catalogues or direct salespersons 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 was expected to produce consistent quotations and order prices while tracking statistical information, such as market trends. Another advantage identified in the development of this system was to provide better service capabilities to current and potential customers.

3.1. Euronet

During 1996 and 1997, a small information technologies team in a Distribution Transformers Business Unit of ABB began exploring internet technology with the goal of building a catalogue-based intranet system for overhead distribution transformers. This work resulted in a two advanced prototype software applications that had the potential of replacing an existing sales order handling system, being used within the US field sales organisation, and a quotation system for the Underground Distribution Transformer Division. These two prototypes led to the establishment of the Euronet project. The initial objective of the Euronet project was to deliver a system with the field sales functionality and the catalogue system functionality to be used by salespersons for 'standard-catalogue' transformers in every European plant. This basic functionality already existed in the sales order handling and quotation prototype systems. However, the Euronet system also needed additional functionality; for example, to handle multiple languages, consider multiple currencies and allow the exchange of quotations and orders in each country in the local language and currency. A verbal agreement on the total functionality of the Euronet system was reached and a two-page document was generated to formalise the contract with the project sponsor. The project was to be completed within three to four months of initiation. It is necessary to point out that the developer team was the primary team responsible to identify the required functionality for the Euronet system. The envisioned users had only a vague notion of the actual system functionality.

A development team, comprised of software developers, IT specialists and people with intimate knowledge about the business activities, was assembled to deliver this system. The development team began working in several parallel tracks. First, members with intimate knowledge about the business defined the business implications of combining and/or uniting sales and technical support people across Europe with a single system. Second, an initial Software Requirements Specification (SRS) document was created combining knowledge about the current functionality of the prototypes and the identified business implications. Third, based on the functionality of the prototypes developed, software developers began the Euronet system implementation. Although use cases were employed to develop the SRS document, according to the software development team the use cases were not sufficient in and of themselves to develop the initial SRS document. Once the SRS was completed, it actually served as the basis to develop the users' manual. Similarly, the initial SRS document was used by another team member as the basis for the unit and system test plans. The initial SRS document was not regularly maintained; as functionality was added to the system, the software development team did not have the time or resources to update and maintain the SRS. The SRS was, thus, incomplete and inconsistent and the fact that it was never intended to be the sole contract for the project partially contributed to this.

3.2. Goal Analysis Efforts

Our analyst team was provided with an SRS containing typical descriptive information such as system scope, boundaries and a feature summary, 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 on an attempt to create an implementation-independent requirements specification. Developers had created use cases based on the screens. These use cases essentially described activities that the users were supposed to perform 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 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 other use cases [2]. 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.

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 [9,10,12], initially generating a total of 292 goals as discussed below.

3.3. Goal Evolution

The goal analysis began by identifying goals in main and secondary scenarios. Goals and their associated information were identified, numbered, classified and stored. The information tracked (partially to ensure traceability) included the goal number, responsible agent(s), 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. As preparation for each meeting session, analysts identified goals from predetermined scenarios. 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, 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.

Goals were named using meaningful keywords selected from a predefined set of goal categories [10]. The boxes in the ellipse in Fig. 2 show the number of goals named with each of these keywords. In Section 4 we provide further discussion of `NOTIFY`, `INFORM`, `PROVIDE` and `ALLOW` goals. Figure 2 shows the evolution of the goal set. Shaded boxes represent goals removed from the goal set for various reasons, which 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.

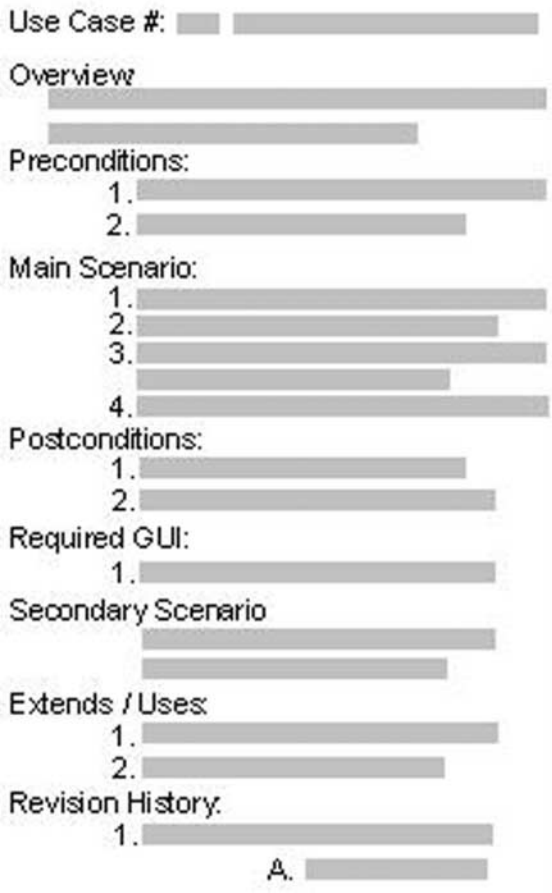

Fig. 1. Example SRS (use case structure).

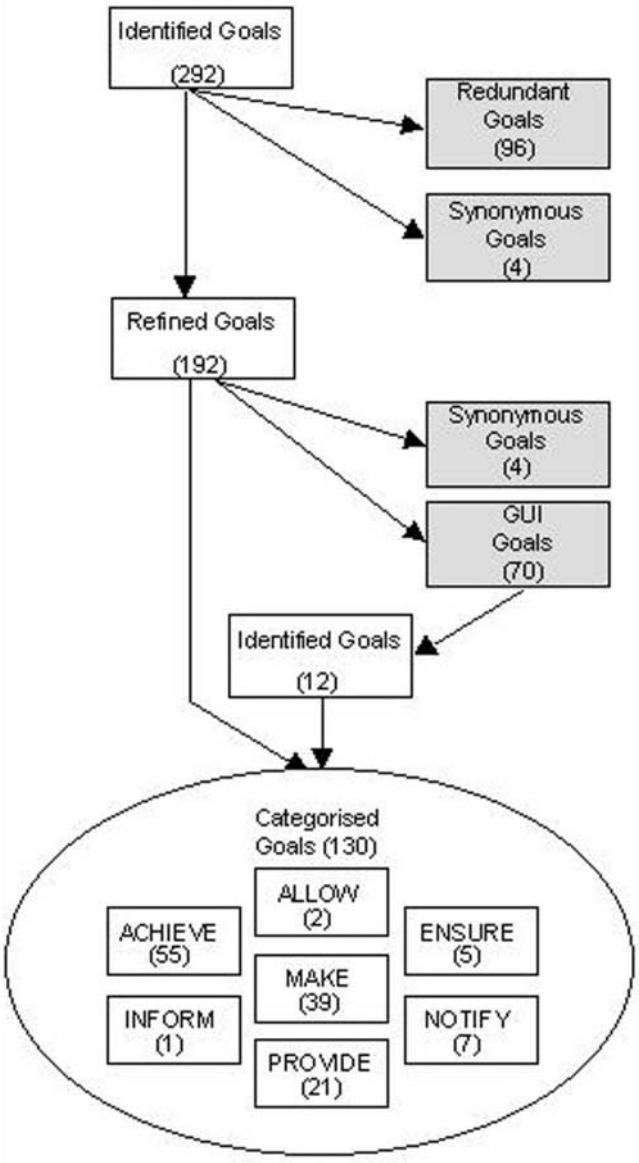

Fig. 2. Goal evolution diagram.

Goals were refined by applying the GBRAM heuristics [10], resulting in the elimination of 100 goals. Seventy 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 categorisation enriched the goal set by replacing implementation-specific goals. While identifying these goals, we struggled due to five specific challenges.

4. Challenges and Associated Risks

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

Context is 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. On numerous occasions we had to refer to other use cases to deepen our contextual understanding. There were various reasons for this. First, while the analysts are very familiar with the domain of electronic commerce, there were some use cases which 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 contextual information to describe the circumstances under which the use case is relevant as advocated in Lilly [22]. 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 provided this context by always attaching a goal to each use case, as in [10,11,16]; every use case title was thus expressed as a goal.

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 incur the obvious risk of producing an incomplete or erroneous specification. Use cases are typically utilised 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.

The use case authors were not the intended users of the system
Since the SRS authors were 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. Since the actual use cases were not written in a participatory fashion, the actual 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 [23]. Hence, one would expect more directed use cases to be produced given more stakeholder involvement. Furthermore, 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 SRS.

In light of the lack of user participation throughout the requirements specification process, we identified a number of potential project risks. The scenarios were not validated by system users; thus, we 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 [24]. The fact that the authors only investigated scenarios from one viewpoint also introduces risk, since exploring multiple viewpoints yields more comprehensive requirements coverage.

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 [25,26]. Each time a duplicate goal was resolved/removed, we maintained a list of 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 goal 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 a new spreadsheet. Whereas 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 life cycle artefacts. Maintaining traceability information can be quite time consuming; thus, it may be tempting to skimp on traceability to save time or money. Additionally, any manual processes (as were our traceability efforts throughout this study) are always subject to human error. Manual traceability may also result in the production of static documents, which often remain unmodified after their initial creation and as a result often become obsolete [26]. Traceability reduces the risk of inconsistencies and ensures compliance with the SRS. Introducing additional traceability can affect cost and scheduling estimates. However, the gains and benefits well outweigh the costs [26].

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 inherently affects its style and content. The SRS appears to have been written in retrospect, possibly as a process requirement dictated by management. Additionally, the SRS authors relied excessively on the available user interface to construct their use cases. 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 that should be addressed during software design, not requirements engineering [22]. 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 [10]. An excessive focus on GUI in the use cases posed significant challenges since all GUI references had to be extrapolated from the derived goals [10].

Inconsistency across available screen designs also results in corresponding inconsistencies in the derived goals. 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 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 that describe subsystem interactions is acceptable but such information should not be recorded in system-level use cases. Instead, as suggested in Lilly,[22], design information discovered during requirements analysis should be placed in a separate 'design guidance' document.

Introducing design considerations prior to problem definition may corrupt the analysis process [27]. The over-reliance of use cases on a GUI therefore introduces the risk of software design interfering with initial analysis. If design considerations are introduced prior to the completion of 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 [22]. 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. 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 perhaps have to 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.

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 non-existent use cases. Additionally, some referenced use cases were never defined. To identify missing use cases, we created an 'includes tree' to track relationships between use cases and discovered 15 missing use cases. Figure 3 shows a portion of a use case tree in which all referenced cases are defined. In contrast, Fig. 4 shows a use case tree in which use cases referenced do not exist. Referenced use cases may be missing or 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. A missing use case, while it provides a use case name, obviously represents missing interaction information, suggesting missing requirements. A simple use case name does not sufficiently describe the use case details which affect the derived requirements. Discrepancies between use case names also introduces 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. Use cases referenced using the incorrect name also increases the likelihood of conflicts.

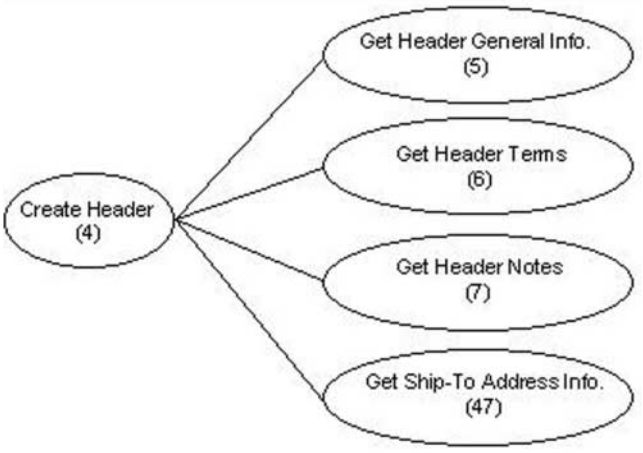

Fig. 3. SRS use case 'includes tree'.

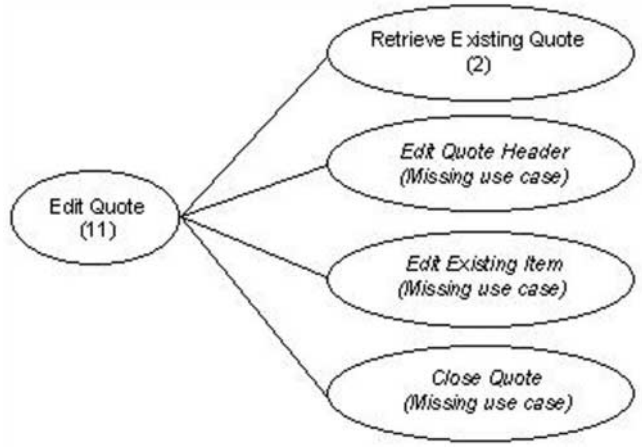

Fig. 4. SRS use case 'includes tree' with missing use cases.

5. Lessons Learned

Despite the challenges encountered, our analysis yielded several lessons learned that we believe to be of interest to 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 by expressing achievement goals accordingly. Previously, when employing the GBRAM [10], we made no real or clear distinction between `MAKE` and

ACHIEVE goals. During this case 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 [9,10,12]. 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 Antón [10]. Of particular relevance to this study are the goal classes for an electronic commerce application, classified according to subject matter, as reported in Antón and Potts [12]. These electronic commerce goal classes are:

- process support;
- electronic commerce;
- information display and organisation; 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 organisation goals describe the organisation and presentation of information by the system. Finally, security and access control goals describe those goals involved in limiting access to authorised users [12]. These categories are not mutually exclusive. The same goal can be, and often is, classified according to more than one 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 organisation 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 organisation 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; the 73 electronic commerce goals emphasised 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.

Interestingly, 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 particularly alarming. There were only eight security and access goals 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 behaviours have been considered for the given system. Upon realising that we had not derived a sufficient number of security and access control goals, we were able to further analyse 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 [28]. Many goals in this study resembled the kinds of goals reported in Antón and Potts [12]; they involve provision of certain capabilities and functions or provision of information to and from 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 delivery or provision of information to a given actor. Notification goals can involve an email message that is sent or an error message that is displayed on the screen. In this study, the primary actor delivering 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 system. Likewise, the goal `<ALLOW user logged on>` permits the secondary actor (e.g., user) to utilise 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.

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. As previously mentioned, the SRS appeared to be produced under some duress 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 summarises behaviour 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. The various available use case representations [2,7] 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.

6. Discussion and Future Work

Scenarios are valuable for eliciting information about system requirements, communicating with stakeholders and providing context for requirements [8,11,14,16,29]. 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 [6,8,17]. 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. The results presented are applicable to other domains beyond electronic commerce such as transaction-based systems and messaging systems.

Our study enabled us to identify a number of helpful techniques for extending the current heuristics found in the GBRAM [10]. The study also yielded some rather interesting observations, leading us to analyse the risks associated with various challenges encountered. 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 Antón [10] 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 [17]. 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 [17].

Given the over-reliance on use cases, which were not altogether complete or correct, during the early stages of this development effort, it is not surprising that the initial Euronet system lacked much of the corporation's desired functionality, resulting in an unfortunate series of necessary upgrades and additions. The system was determined to be successful only after three versions were completed. Furthermore, when members of the development team were asked to reflect upon the applicability and efficiency of the use case-based approach for preparing the SRS, their responses were less than favourable: team members indicated that, in future projects, they would prefer to forgo the documentation of use cases as the primary expression of requirements if possible. Thus, another approach – one that would provide a more complete set of requirements and reward and minimise the practitioners' efforts – may prove more beneficial to software development than a solid use-cases-as-requirements approach for some software systems. Through the use of the GBRAM methodology, for example, a more complete and consistent set of requirements could be created, leading to a better evolution of systems such as Euronet.

Even though our primary focus throughout this study was on the role of goals and use cases during requirements specification, we also observed some interesting organisational challenges which we believe are not endemic to this entity. For example, in retrospect, our sponsor has chosen to view this effort as somewhat evolutionary since there was no rigid adherence to software process guidelines (such as maintenance of the requirements and design specifications). Although the effort was unintentionally evolutionary in nature, the use case-based specification did serve as the initial basis for subsequent prototyping cycles and, as previously discussed, new functionality and component upgrades were introduced during each phase. The final system was considered complete only after three prototyping cycles. It was eventually deemed successful, but the lessons learned suggest both the need for further research and advances in the role of use cases and goals during requirements specification as well as the need to investigate and develop more appropriate evolutionary requirements and software process models for rapid development environments.

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 van Lamsweerde et al. [30] to further extend and refine the GBRAM [10] in this way. Episodes also play a significant role in effective scenario management [29]. Episodes are sequences of events shared by two or more scenarios [17]. We manage episodes by distinguishing events and recognising 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.

Researchers at North Carolina State University (NCSU) are developing scenario management strategies [17] in an effort to address the challenges discussed in Weidenhaupt et al. [8]. 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.

Finally, just as use cases and scenarios are beneficial for uncovering requirements [29], evolutionary prototyping also aids in identifying missing or previously unknown requirements [24]. To this end, we are continuing our collaboration with our project sponsor as we focus on defining appropriate evolutionary prototyping models that provide analysts with feedback and additional insights into the evolving set of requirements needed to build a given system.

Acknowledgements. The authors wish to thank David Daugherty of Asea Brown Boveri for sponsoring this project, as well as Janet Valashinas and Randy Avery for their participation in goal elicitation and follow-up interviews; Thomas Alspaugh for his 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 1999–2000.

References

1. Jacobson I et al. Object-oriented software engineering: a use case driven approach. Addison-Wesley, Reading, MA, 1992
2. Booch G, Rumbaugh J, Jacobson I. The Unified Modeling Language user guide. Addison-Wesley, Reading, MA, 1999
3. Fowler M. UML distilled: applying the standard object modeling notation. Addison-Wesley, Reading, MA, 1997
4. Booch G. Object-oriented analysis with applications (2nd edn). Benjamin/Cummings, Redwood City, CA, 1994
5. Rumbaugh J et al. Object-oriented modeling and design. Prentice-Hall, Englewood Cliffs, NJ, 1991
6. Jarke M, Bui XT, Carroll JM. Scenario management: an interdisciplinary approach. Requirements Eng 1998;3(3–4):154–173
7. Antón AI, Potts C. A representational framework for scenarios of systems use. Requirements Eng 1998;3(3–4):219–241
8. Weidenhaupt K, Pohl K, Jarke M, Haumer P. Scenarios in system development: current practice. IEEE Software 1998;15(2)
9. Antón AI. Goal-based requirements analysis. In: International conference on requirements engineering (ICRE'96), Colorado Springs, CO, April 1996, pp 136–144
10. Antón AI. Goal identification and refinement in the specification of software-based information systems. PhD thesis, Georgia Institute of Technology, Atlanta, GA, June 1997
11. Antón AI, McCracken WM, Potts C. Goal decomposition and scenario analysis in business process reengineering. In: Advanced information systems engineering, 6th international conference proceedings (CaiSE'94), Utrecht, The Netherlands, 6–10 June 1994, pp 94–104
12. Antón AI, Potts C. The use of goals to surface requirements for evolving systems. In: International conference on software engineering (ICSE'98), Kyoto, Japan, 19–25 April 1998, pp 157–166
13. Breitman KK, Leite JC. A framework for scenario evolution. In: International conference on requirements engineering (ICRE'98), Colorado Springs, CO, 6–10 April 1998, pp 214–221
14. Potts C. 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, MI, August 1995, pp 247256
15. Potts C. A ScenIC: a strategy for inquiry-driven requirements determination. In: Proceedings of the IEEE 4th international symposium on requirements engineering (RE'99), Limerick, Ireland, 7–11 June 1999
16. Rolland C, Souveyet C, Ben Achour C. Guiding goal modeling using scenarios. IEEE Trans Software Eng 1998;24(12):1055–1071
17. Alspaugh TA, Antón AI, Barnes T, Mott B. An integrated scenario management strategy. In: International symposium on requirements engineering (RE'99), Limerick, Ireland, June 1999, pp 142–149
18. Maiden N, Minocha S, Manning K, Ryan M. CREWS-SAVRE: systematic scenario generation and use. In: International conference on requirements engineering (ICRE'98), April 1998, pp 148–155
19. Dardenne A, van Lamsweerde A, Fickas S. Goal-directed requirements acquisition. Sci Comput Program 1993;20(1–2):3–50
20. Antón AI, Liang E, Rodenstein R. A web-based requirements analysis tool. In: 5th workshop on enabling technologies: infrastructure for collaborative enterprises, June 1996, pp 238–243
21. Rolland C, Grosz G, Kla R. Experience with goal–scenario coupling in requirements engineering. In: 4th IEEE international symposium on requirements engineering, 7–11 June 1999, pp 74–81
22. Lilly S. Use case pitfalls: top 10 problems from real projects using use cases. In: Proceedings, technology of object-oriented languages and systems, 1–5 August 1999, pp 174–183
23. Hotzblat K, Beyer H. Contextual design. Morgan Kaufmann, San Francisco, CA, 1998
24. Davis AM. Software requirements: objects, functions, and states. Prentice-Hall, Englewood Cliffs, 1993
25. Dömges R, Pohl K. Adapting traceability environments to project-specific needs. Commun ACM 1998;41(12):54–62
26. Ramesh B. Factors influencing requirements traceability practice. Commun ACM 1998;41(12):37–44
27. Embley DW, Jackson RB, Woodfield SN. OO systems analysis: is it or isn't it? IEEE Software 1995;12(4):19–33
28. Jackson M. Software requirements and specifications. Addison-Wesley, Reading, MA, 1995
29. Potts C, Takahashi K, Antón A. Inquiry-based requirements analysis. IEEE Software 1994; 11(2):21–32
30. van Lamsweerde A, Darimont R, Letier E. Managing conflicts in goal-driven requirements engineering. IEEE Trans Software Eng 1998;24(11):908–926