CHAPTER 1

Introduction

If you don't understand the user's requirements, it doesn't matter how you code it.

Ed Yourdon

The core of this research is aimed at improving the early stages of the software development process in complex environments. The early planning and analysis stages have been recognized by many observers as critical, yet there have been few substantial advances in this arena. The primary focus of this work is the transformation of enterprise and systems goals into requirements, with the objective of formulating a method that is useful for identifying and refining the goals that systems must achieve and subsequently converting them into operational requirements. Specifically, the method detailed in this thesis provides procedural support for the early identification and elaboration of goals in the specification of software-based information systems. The objective of this chapter is to establish the context for this research.

1.1 Requirements Engineering

Requirements describe the desired functionality of a system. In general, there are two types of requirements: functional and nonfunctional. Functional requirements describe the behavioral aspects of a system; non-functional requirements describe the non-behavioral aspects of a system.

Requirements analysis is the process by which the purpose and functionality of a system is elicited and modeled [53]. Requirements analysis and specification is one of the most challenging and error-prone activities in the software process. One reason for this is the high level of communication-intensive activity required. Stakeholders frequently lack a clear understanding of what the desired system should do and often tend to change their minds regarding the functionality that the system must exhibit. It is important to identify the issues that lead to instability early in the formulation of software requirements. To clearly identify these issues, stakeholders and analysts must communicate and negotiate the terms for any proposed system. Requirements are useful for such communication, serving as a contractual language understood by both parties.

Requirements engineering is plagued by practical problems such as complexity, volatility, ambiguity, the need to rely on intuition, and disagreements among stakeholders. The discussion of these practical issues sets the stage for the principles and methods presented in this thesis.

Complexity

Obtaining requirements is conceptually complex. It demands grasping a set of nebulous ideas that may be incomplete or in conflict, and transforming that set of ideas into a complete and consistent elaboration of the technical requirements for a software system. These technical requirements must be easily comprehensible to the intended customer or stakeholders. Projects often fail to fulfill stakeholders' goals, however, because the requirements are inadequately explored and described. Analysts usually work from a vague statement of needs or goals. Additionally, each stakeholder has different requirements and priorities which sometimes conflict with the goals of the group as a whole. As a result, requirements are often not representative of the enterprise goals and/or conflict with them. Often the strategies for conflict identification and resolution are inadequate.

Volatility

Requirements creep is defined as the evolution in requirements between the time at which actual coding begins and the time of product delivery [54], with direct reference to changing and adding requirements. *Requirements churn* refers to the frequent modification of the same requirements or their priorities. The volatility of requirements, especially after the requirements "phase," is the source of one of the greatest problems in software development.

Capers Jones observes that "Creeping user requirements will grow at an average rate of 1% per month over the entire development schedule" [49]. For a two year project, this translates into an increase in functionality of 24% or 36% for a three year project. Requirements creep is a significant source of cost and time overruns. While systems are subject to a certain amount of requirements creep, such volatility can be minimized by the extensive exploration and identification of goals and requirements early in the requirements process.

Because requirements are volatile and constantly changing, there is much iteration in the refinement process due to the misunderstanding and/or misinterpretation of the requirements. Discussions transpire in which issues are highlighted, conflicts arise, and resolution is sought; when no record is kept of the analysis activities, omissions and the loss of information are inevitable. Better tools and process models are needed to support the negotiation, conflict resolution, and requirements refinement process [21].

Reliance on Intuition

Requirements elicitation techniques have received attention in recent years [23,28,35]. However, a definitive process for analyzing the gathered materials remains largely unaddressed and is left primarily to intuition. Requirements elicitation techniques have included sequential interviews of individual stakeholders or end-users, questionnaires of the general user group/community, and observations of end-users interacting with a particular system [35]. In large software development projects these techniques often prove inefficient, leading to poor communication among project members, thus necessitating a lengthy resolution process [16].

Ambiguities

When stakeholders write a requirements document, it is usually plagued with informational ambiguities, uncertainties, and gaps. In some cases, a requirements document is not always available, requiring analysts to extrapolate the needs and goals of the stakeholder from only a vague statement of objectives. Often such documents lay the foundation for the requirements analysis process. Analysts and developers must carefully refine and formalize the information provided by the customer while iterating with the objective of producing a functional specification.

Requirements must be stated accurately before they can be implemented correctly. Consider the requirement: `The system should be quick`. Such a requirement provides no intuition into how "quick" the system should be or under what circumstances it should operate. This requirement is neither precise nor accurate. In contrast, the requirement: `The system should respond to a request for help in less than 10 seconds` exhibits much more precision. The expression of precise, clear requirements is essential in the prevention of requirements errors that arise due to ambiguities. Techniques are needed to guide analysts through the process of identifying specific goals and needs from ambiguous starting points.

Disagreements with Stakeholders

Individual analysts have traditionally been responsible for eliciting requirements from users, synthesizing the information acquired, and then modeling or developing a representation of the system requirements. The model developed is typically based on the analyst's personal understanding of the requirements after he has synthesized the information. The stakeholders' only involvement throughout the requirements determination process is limited to serving as an information source. However, when an analyst creates a model within the context of these parameters, the structure of the model is usually understandable only to the analyst [70]. A model jointly developed by both the analyst and a group of stakeholders affords a model that is easily understood and readily accepted by both parties.

In group requirements determination, analysts, developers and stakeholders are represented and included in the process. Researchers have advocated increased user involvement in the systems development process, indicating that this heightened level of participation contributes to the development of better systems. According to Hayes, user participation enables the creation of a 'better' model than that created solely by the analyst and increases the probability of implementation success [43]. To this end, continued interaction with and the involvement of stakeholders throughout the requirements process is strongly advocated.

Given the above problems of complexity, volatility, and ambiguity, we now examine the consequences of poor requirements processes in the context of system development efforts.

1.2 Consequences of Poor Requirements

The consequences of misinterpreting requirements may be very severe. This section clarifies and substantiates the importance of clearly understanding the requirements early in the planning stages of the development process.

The ultimate consequences of requirements errors are expensive. In some large systems up to 95% of the code had to be rewritten to satisfy changed user requirements [9]. Boehm also reports that 12% of the errors discovered in a software system over a three year period were due to errors in the original system requirements. The correction cost can be up to $\frac{1}{3}$ of the total production cost [9]. If a software requirements error is detected and corrected during the analysis stage of the development process, the error is relatively simple to correct since it involves only a correction to the requirements specification. However, if a requirements error is not corrected until the maintenance stage, a much larger inventory of artifacts is affected (e.g. specifications, code, as well as user and maintenance manuals). Additionally, late corrections involve a formal change control and approval process and require extensive re-validation. Fjeldstad and Hamlen estimate that 47% of the time devoted to maintenance activities is devoted to the actual enhancement and correction tasks while 62% is dedicated to comprehension activities [34]. These figures suggest that maintenance costs may be significantly reduced if efforts are devoted to increasing the level of comprehension within the system specifications. Thus, it is critical to ensure that requirements are identified as early as possible and that measures are taken to prevent any requirements from being overlooked. Because requirements engineering is expensive, the objective is to improve the process as quickly and efficiently as possible.

As the software lifecycle progresses, the cost of repairs due to errors made in specifying the requirements increases significantly [9]. Failure to identify errors in requirements can be dangerous; for example, 62% of safety-related functional faults in the Voyager spacecraft and 79% in Galileo were caused due to errors in recognizing the requirements [57]. It must also be noted that requirements errors are persistent. According to Boehm, $\frac{2}{3}$ of requirements errors are detected after delivery [9]. That is, the system is already delivered to the customer before the requirements error is detected by an analyst or the customer.

The consequences associated with poor identification of software requirements, discussed above, are largely due to lack of validation. The next section examines the origin of these requirements errors.

1.3 Requirements Validation

Requirements errors are often due to improper requirements validation. Requirements validation involves assuring that a system meets the actual needs of the stakeholders and customers. Validation is usually defined by asking variants of the following question: "Are we building the right system?" Validation is a subjective process in which assessments are made as to how well the proposed system addresses the stated needs of the stakeholders. It involves iterative reviews involving the participation of stakeholders and the developers. The process is necessary to ensure that requirements are consistent, complete, and realistic. Traditionally, validation includes activities such as walkthroughs and prototyping. Given these constructs, goal analysis is utilized as a key tool for validating software requirements and is presented in this thesis as an approach to validation.

Stakeholders frequently fail to use, or even discard, a system based on their acquired knowledge of the system as an ineffective means of assistance in the achievement of their goals. The cause of this misuse of effort and lack of efficiency is commonly attributed to the need to develop an understanding of the requirements from stakeholders who do not themselves understand the requirements. The problem with requirements engineering stems from stakeholders lack of understanding of their own requirements; this ambiguity is thus passed on to the analyst, resulting in imprecise objectivization of software requirements. Goal analysis improves the understandability of the requirements by enabling the production of a set of requirements that may be validated by those to whom the requirements bear the greatest significance. Since multiple stakeholders have many goals, it can be difficult to develop a clear understanding of the desired diversity. Goal analysis clarifies the stakeholders' goals by tracking the rationale associated with specific goals. Thus, goal analysis becomes a means to validate system requirements.

1.4 Goals in Requirements Analysis

Traditional approaches to requirements analysis focus on the elicitation of specific requirements. As noted in Chapter 1.3 stakeholders usually have a better understanding of the general goals they want to achieve than they do the functionality that should be exhibited by the desired system. The requirements specification, based on formal models and formal specifications, often serves as a contract with the stakeholders. When stakeholders are unfamiliar with these notations, or have not been trained in formal specifications, these documents can be cumbersome and intimidating. Since requirements specification documents serve as a contractual language, it is important to provide stakeholders with information in an understandable language in which they may actively participate. By focusing on goals instead of specific requirements, analysts enable stakeholders to communicate using a language based on concepts (e.g. goals) with which they are both comfortable and familiar. It must also be noted that enterprise goals and system goals are more stable throughout the lifetime of an enterprise than are the requirements defined at any one time. It is therefore imperative to utilize an understanding and structure of goals to derive requirements from a stable starting point (e.g. goals).

The changes that occur throughout the progression of requirements model from an informal representation to a more formal representation may contribute to the difficulties in communication between the analyst and stakeholders. The transition from requirements to design may be difficult for end users to follow due to the changes in representation. Functional modeling techniques do not provide the necessary 'evolutionary formalization' [5] needed to bridge the gap between the analysts' and stakeholders' understanding of the system requirements. Goals are evolutionary and may thus provide a common language for analysts and stakeholders.

This thesis builds upon existing work which addresses the utilization of goals in systems development for the direct transformation of goals into formal specifications. The objective is the direct transformation of goals in natural language requirements. The Goal-Based Requirements Analysis Method (GBRAM) presented in this thesis introduces techniques that force in-depth, methodical, and systematic analysis to clearly elucidate the desired goals.

1.5 Overview of Remaining Chapters

The research discussed in this thesis stems from the need for better heuristics and procedural guidance for initially identifying and constructing goals, with particular emphasis focused on those which involve the relationship between goals and scenarios. The need exists for prescriptive advice in the form of goal-identification heuristics and a set of consistently recurring questions to guide the process. Research conducted in the preparation of this thesis has identified a set of issues which practitioners should consider.

This thesis offers prescriptive guidance for goal identification, providing requirements elaboration techniques centered upon goals and scenarios.

Chapter 2 provides a survey of the related work in the field and discusses the use of goals in requirements engineering as well as the influence of goals in other disciplines.

Chapter 3 presents the case studies which served as the conceptual origin for the Goal-Based Requirements Analysis Method. The discussion in this chapter justifies the heuristics presented in Chapter 5.

Chapter 4 details the Goal-Based Requirements Analysis Method (GBRAM), focusing on the activities with which analysts are involved when employing the method. This chapter discusses the application of the method, illustrating its salient features through examples from the case studies discussed in Chapter 3.

Chapter 5 presents the guidelines and heuristics which guide analysts through goal-driven requirements analysis. Examples from the case studies are provided to elucidate the heuristics; the heuristics are accompanied by basic questions or inquiries for the analyst to apply when using the method. A discussion of the possible changes that result from asking these questions and guidelines for the kinds of refinements the analyst can make are also concomitant themes in this chapter.

Chapter 6 discusses the two main validation efforts for the method presented in this thesis: a large scale industrial case study involving the reengineering of an electronic commerce Web server, and an empirical investigation in which the method was applied to a small system by individuals who were previously not familiar with goal-based approaches.

Chapter 7 summarizes the contributions of the thesis and future work needed to further refine the method.