Presented At
The Fifth International Society of Science and Applied Technologies (ISSAT) International Conference on Reliability And Quality In Design,. Las Vegas, Nevada, USA August 11-13, 1999

HOW RELIABLE ARE THE REQUIREMENTS FOR LARGE DIGITAL SYSTEMS?

Herbert Hecht **Myron Hecht**
SoHaR Incorporated
Beverly Hills, California

Key Words: Requirements, requirements engineering, system engineering, figure of merit

Abstract

Requirements for large information systems are not reliable if we interpret that term to mean complete, consistent and current. A General Accounting Office examination of 10 major systems shows deficiencies in generating complete requirements and in keeping them consistent and current. Requirements analysis tools, formal methods, and recursive development formats are beneficial but do not address organizational and contractual issues. As an alternative to the current methodology we discuss a figure of merit as the key requirement and giving the developer more responsibility for trade-offs during development.

1. Introduction

Because of the importance of information to any enterprise, the term "mission critical" has been attached to almost all corporate on-line computer-based systems. In spite of this acknowledgement of the importance of these systems, all of us are acquainted with horror stories of systems that had to be abandoned after huge sums had been spent on planning and development, systems that were accepted but did not meet user requirements, or were many years late and significantly over budget. An examination by the General Accounting Office (GAO) of problems in the government arena concluded "Information systems are now integral to nearly every aspect of over $1.5 trillion in annual federal government operations and spending, yet, despite years of experience in developing systems, agencies across the government continue to have chronic problems harnessing the full potential of information technology to improve performance, cut costs, and enhance responsiveness to the public…GAO reports and congressional hearings have chronicled numerous system development efforts that suffered from multimillion dollar cost overruns, schedule slippages measured in years, and dismal mission-related results."[1]

The GAO report, like practically every article dealing with these problems in the popular press, cites "poor management" as the cause. The GAO lists more than 10 detailed studies to support its case. But not all managers are incompetent or lack motivation, and thus it we should inquire why the development of large software-intensive systems is so difficult. Specifics cited in the GAO report are failure to implement "a process for selecting, prioritizing, controlling, and evaluating the progress and performance of all major information systems and investments, [including] *disciplined, consistent procedures for software requirements management"* [italics added]. Many of the problems are a legacy of the waterfall model of system development which assumes that complete requirements can be formulated at the outset of a project, that milestone reviews can establish that the requirements are correctly implemented, and that faithful execution of this process will assure a successful outcome.

As will be shown, it cannot be assumed that requirements for a large software based system will be complete, and that requirements stated at the beginning of the

project will still be valid once the development gets under way. Thus, what is needed is a paradigm that views requirements changes as the rule rather than the exception.

2. Requirements Formulation

Among the obstacles to being able to generate complete requirements for large software projects we will focus on three:

- the novelty of the objectives, and the resulting uncertainty of budget and schedule estimates
- the gap between the user and developer cultures
- restrictions on staff size for requirements formulation

In a given instance we can usually identify means for overcoming these problems (particularly in retrospective), but in the aggregate they raise doubts whether we can expect complete and correct requirements for large software based systems. Specific reasons follow.

Novelty of Objectives

When you contract for having a house built, you have a point of reference in the home in which you live now, the home in which you grew up, and the homes in which your friends live. In addition you will probably visit model homes and read magazines and books about current trends in the building industry in your area. Your requirements are largely based on existing examples, with at most minimal innovations. Contrast this with the situation in large information systems where there are usually no existing equivalents and where the objectives usually include a very high degree of innovation. The primary objective of the new features may be well researched, but this cannot be said for auxiliary functions or side effects. Cumbersome methods for handling corrections, for recognizing false data, and for security have required extensive modifications to many systems. When a prominent internet service provider changed from time-based charges to a flat rate, the resulting increase in usage effectively disabled the system for several weeks. This (unforeseen?) side effect caused huge financial losses that could have ruined any but a well-capitalized company. Although this is not a software problem it is cited here to show how focusing on a primary objective can lead to neglecting even obvious side effects.

Cost and schedule uncertainties in a software project are closely tied to the extent of the innovation, just as the uncertainty in the construction cost of a house increases with the extent of changes from the builder's standard plan. Does that mean that we should discourage innovation in order to increase the predictability of software efforts? This can hardly be the complete answer if we want to benefit from the productivity gains, the higher standard of living, and the enhancements in our quality of life that have been made possible by large software projects. While the risk associated with major innovations has to be recognized, we also need to structure the development process to accommodate innovation and the attendant need to change requirements during the development.

Culture Gap

The user community provides a service that usually has considerable legacy, such as air traffic control, inventory management, or tax collection. Inevitably, within this community there are unwritten rules, a special vocabulary, and accepted wisdom. As part of the formulation of requirements this culture must be conveyed to the developer community that has software skills but usually little background in the user domain. Obviously, the gap can be the source of missing requirements and misunderstood terms. When a major electric utility wanted to convert from an analog feedwater system to a digital one for its nuclear power plants, these problems were recognized and a well written instruction manual for the feedwater control, including diagrams and extensive reference material was prepared for the developer. Triplicated water level sensors, installed as part of the analog system, were to be retained for the digital system. What could be more natural than to also retain the method for identifying failed sensors? Thus, "Any sensor

that deviates by more than 5% from the average reading shall be declared invalid" Assume that the level is 0.5 and that one sensor fails to null output. The average reading will be 0.33 and all three sensors deviate by more than 5% from the average and are declared invalid in the digital system (in the analog system the average was obtained by a resistor-capacitor summing network which provided sufficient lag to avoid this problem). Accepted wisdom caused considerable problems.

Again, in hindsight, the problem can be avoided. But both the user and the developer were major and highly respected companies that put their best people on this project. We can recognize the culture gap but we cannot completely eliminate the risk that it poses of missing requirements or misunderstood terms.

Small Staff for Requirements Formulation

It is quite common to assign only a small staff to the project when requirements are being formulated with the intention that experienced personnel from the actual users will be made available to provide the "hands on" guidance. But experienced personnel have allegiance to their parent organization and seldom see their participation in a project that is many years from being operational as their main obligation. These employees will make sure that the mainline functions are correctly represented in the requirements, but they are seldom motivated to go into the details of error handling and other exception conditions that are the cause of so many serious problems in major software projects.

The funding for the start-up of software projects within the user organization usually comes out of an overhead budget and it is understandable that only a small dedicated staff can be assigned. In addition, the requirements must frequently be generated within a compressed time schedule in order to meet budget cycle deadlines for the funding of the project. These organizational constraints are difficult to overcome, and this leads to the conclusion that requirements for large systems are likely to be incomplete and sometimes incorrect.

3. Requirements Management

In the preceding paragraphs we have seen that the initial formulation of requirements for a major software project is rarely complete and correct. We will now examine how the management of requirements during the development can overcome these obstacles. The conventional wisdom is to rely on the review of milestone products by the user organization, sometimes augmented by a specialized independent verification and validation (IV&V) team. In most cases the reviewers furnished by the user organization are on part-time assignment and must balance commitment to their permanent position against effort devoted to the review. They will seldom have the time or motivation to uncover deficiencies in the details of the requirements (and, as usual, that's where the devil hides).

Once a need for a requirements change is identified, the real obstacles posed by the waterfall paradigm become evident. Typically the sponsor must request an Engineering Change Proposal (ECP) and pay the developer to identify the details of the change and to generate cost and schedule estimates. Once these are submitted to the sponsor they must be evaluated and funded. It is not at all unusual that by the time the changes are implemented, circumstances have arisen that require further changes that may conflict with the original ones. Neither the technical nor the contractual procedures facilitate cooperation between the sponsor and the developer to resolve what may well be termed "normal" deficiencies in the requirements that are to be expected in such circumstances.

We have now concluded that most software projects will be developed from requirements that are incomplete, sometimes incorrect, and usually out of date. That some succeed is due to minimal innovation, unusually dedicated teams in both the user and the developer

organizations, and particularly to the willingness to accept a less than perfect system. Let us now look at ways by which more complete, correct and timely requirements can be generated.

4. Aids to Better Requirements

Methodologies for disciplined requirements formulation and analysis have been available for over twenty years [2,3]. More recently, requirements formatting has been incorporated into overall software development tools[4]. This reduces the effort and removes some obvious sources of errors and omissions. We want to acknowledge these benefits but they don't address the key difficulty of dealing with a new capability for which side effects and response to unusual situations are very hard to define.

Formal methods have also been proposed for many years as a systematic way of assessing the completeness and correctness of requirements[5]. Within a defined scope of action they do identify errors and ambiguities, but many of the problems described earlier arise from the difficulty of defining the scope. As a trivial example, if the need for security protection is not recognized, formal methods are not likely to detect this omission.

Methodologies for recursive evolution of requirements, such as spiral development[6] and rapid prototyping[7] recognize the difficulties described in the earlier sections and the need for periodic updating of the requirements. But they are limited in that they do not address the administrative obstacles at the sponsor-developer interface. When missing requirements are identified during an iteration, the correction will in most cases require going through the ECP process with its delays and inefficiencies.

In the sections on Requirements Formulation and Requirements Management we have demonstrated the many obstacles to documenting complete and correct (current) requirements for the development of large software based systems. In the current section we have seen that in spite of decades of efforts in research and tool development essential difficulties remain. Is it possible that we are asking too much of the sponsor too early in the development? If the answer to this question is in the affirmative, then we must look for mechanisms by which effective development can be carried out with – at least at the outset - much less complete requirements. There are several ways of accomplishing this: in-house development by the user (or user surrogate, such as a free market vendor), establishing a long-term relation with an organization experienced in system development, or providing motivation for the developer to keep refining the skeletal requirements in the direction desired by the user. The in-house development is frequently used for shrink-wrap software and for specialized commercial products such as flight-control systems. This approach avoids the interface problems, but the projects are generally small and the practice does not appear scalable to large systems and particularly not to government procurements. The long-term relationship has been used by many government agencies for the early phase of major projects, e. g., the FAA employs a highly respected non-profit organization for much of their air traffic control systems requirements development and planning. This has not prevented very major problems with the Advanced Automation System (AAS). Once a contract for development is in place the flexibility gained by working with the long-term resource is lost.

Thus there is ample reason to search for a means to motivate the developer to "grow" the requirements along the lines desired by the user with a minimum of direction (contractual or detailed technical). We start with the mechanism by which an initial concept becomes a budgeted planning item. Usually this requires showing that the new concept is significantly better than the current (or a planned alternate) implementation of a function. How is "better" demonstrated? In most organizations there is a figure of merit (FOM) that is the criterion for "better". It may be system capacity per investment $,

transactions handled per labor hour, or payload weight delivered into orbit per $ launch cost. If this FOM is the basis for selecting a concept initially, why not use it as the key requirement? The FOM does not address constraints, secondary objectives, quality of service (QoS) attributes, and details of measuring the elements in the FOM. Thus, it is not sufficient to tell the developer to build a system that can track 100 aircraft per million dollar investment. But if this was the basis for selecting a project, why omit it from the requirements for the development? The constraints (statutory, system and human interfaces, and environmental) and the details of measuring performance and cost must be spelled out just as they are in conventional requirements and if these change there will be cost and schedule impacts. But secondary objectives and QoS attributes can in many cases be expressed in terms of the FOM [8], and thus the developer can perform required trade-offs without detailed direction from the user. The FOM is also an excellent vehicle for determining profit incentives for the developer. It is not a cure-all, but it can reduce the obstacles that are in the way of generating reliable requirements for major systems.

5. Conclusions

In the initial sections of this paper we have shown that there are substantial difficulties, mostly rooted in the waterfall specification and development procedures, to generating complete and correct requirements at the start of a project, and that these difficulties continue into the development cycle and interfere with timely updates. Some established tools and methodologies can reduce the severity of these deficiencies but do not address the basic cause. Based on these findings we are proposing the system figure-of-merit as a key element in requirements formulation, thereby giving more flexibility to the developer while keeping the primary focus on delivery of a useful product.

References

[1] General Accounting Office, *High Risk Series: Information Management and Technology,* GAO/HR97-9, February 1997

[2] D. Teichroew and E. A. Hershey, "PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Systems", *IEEE Transactions on Software Engineering,* Januaary 1977, pp.41 – 48

[3] M. W.. Alford, "A Requirements Engineering Methodology for Real-Time Processing ", IEEE Transactions on Software *Engineering,* January 1977, pp. 60-68

[4] D. Harel, "Statecharts: a Visual Formalism for Complex Systems", *Science of Computer Programming*, vol 8, pp. 234-274, 1987

[5] K. Heninger, "Specifying Software Requirements for Complex Systems: New Techniques and their Application", *IEEE Trans. Softw. Eng., SE-6*, pp. 2 – 13, Jan 80.

[6] B. Boehm, "A Spiral Model of Software Development and Enhancement", *IEEE Computer*, May 1988, p.61

[7] V. Scott Gordon and James Bieman, "Rapid Prototyping: Lessons learned", *IEEE Software*, Jan1998, pp. 85-94

[8] Herbert Hecht, "Systems Engineering for Software-Intensive Projects", *Proceedings of ASSET'99,* Dallas TX , March 1999