Lessons Learned From Another Failed Software Contract

Dr. Randall W. Jensen
Software Technology Support Center

Software project failure has been with us for a long time. Volumes have been written about the list of potential problem areas in the acquisition of large, complex software systems. The list includes simple things like the cost of reuse, the acquisition process, unrealistic expectations, and the development environment. The list has not changed much in the last 30 years. Unrealistic cost and schedule estimates are causes for project failure as often as inadequate technology. Source selection is a critical acquisition process step. Proper preparation and diligence in this step is key to a successful software project. There are several activities essential to successful project planning and acquisition, including risk assessment. This lessons-learned discussion is based upon a post-mortem analysis of an avionics software development. The intriguing analysis results show this project was neither unique nor abnormal. The problems that surfaced during the project's inception and following downhill plunge were common in the mid-80s environment and are still common today. The purpose of this discussion is to highlight the major software development and management issues that led to this project's failure. The issues presented here are timeless; that is, they are as likely to arise today as they were at any time in the past.

One of the most dominant and serious complaints arising from the ongoing software crisis is the inability to estimate with acceptable accuracy the cost, resources, and schedule required for a software development. Traditional intuitive estimation methods have consistently produced optimistic results that have contributed to the too familiar cost overrun and schedule slippage.

Several schedule and cost estimation methods have been proposed over the last decade with mixed and partial success due, in part, to capability and stability limitations of the estimation models. A significant part of the estimate failures can be attributed to a lack of understanding of the inner workings of the software development process and the impact of that process on parameters used in the schedule and cost estimates.

For example, a major avionics modernization program was started in the mid-1980s. The development contract award for the system development was issued, but by 1990 it was apparent the software product would not be delivered. The government accepted the incomplete software and completed the software in-house. The failure of another software development is, by itself, not noteworthy.

Unfortunately this example is very common. Industry software delivery statistics are quite dismal. Fifty percent of commercial software products are delivered over schedule, 33 percent are cancelled, and 75 percent are operational failures. Government software delivery statistics are similar.

The following lessons learned discussion is based upon a post-mortem analysis of this avionics software development. The intriguing analysis results show this project was neither unique nor abnormal. The problems that surfaced during the project's life were common in the mid-1980s environment and are still common today.

The purpose of this article is to highlight the major software development and management issues that led to this project's failure. The issues presented here are timeless; that is, they are as likely to arise today as they were at any time in the past.

Lessons Learned

This analysis was a classic study of projects gone awry. There are many lessons that can be extracted from the contract history. Since my experience is largely centered on the relationship between software development and methods for predicting cost and schedule, I focused my attention on the development environment impact on the cost and schedule of the avionics program software. I will not touch upon other areas such as risk management that contributed to this project's failure.

There is no implied order of importance to the lessons enumerated here. Each of these issues contributed significantly to the software development failure. Taken together the issues spelled disaster.

Software Reuse and COTS

The magic elixir reuse was the solution to the industry's software cost and schedule problems in the 80s. That was a time when the new programming language Ada and the concept of reusable software component libraries were very popular. Reused software in a mid-1980s development equated to *free* software much as commercial off the shelf (COTS) software does in a development environment today. Unfortunately, software component libraries never became widely available, and the cost savings associated with reusable software were not as large as predicted.

The concept of COTS software is easiest to understand through a *black box* analogy. A COTS component is a black box that can be fully utilized with no knowledge of the box content. White box behavior, on the other hand, requires some knowledge of the internal box workings. A software component is a white box when (1) modification is required to meet system requirements, (2) the component reliability is in question, or (3) the knowledge of the component and its documentation are inadequate for the application. When the white box condition occurs, the effort to implement the software system must be increased to account for reverse engineering the component, coding the component changes, and additional testing required to assure proper component performance after the modification.

The reusable software baseline proposed for this avionics system development was in development by a competitor for this project. The competitor's system had a different architecture and different operational and performance requirements. The delivery schedule for the baseline system that contained the reusable software was from six months to a year following the start of development for the proposed avionics system.

The contractor defined about 90 percent of the *existing* avionics system software as reusable with only about 10 percent of the source lines to be developed as part of the modernization program. The assessment was made with only high-level design information from the baseline system. In reality, there was little reusable software available for the program. Even if the existing software had been available at the start of development, the new software requirements for the system would have precluded any benefit from reuse. The baseline software was not being developed with reuse as an attribute, nor could its developer have been expected to be more than minimally cooperative with the adaptation of that software to the new requirements.

Lesson 1: Reusable (COTS) software never was, is not now, and never will be free.

There is always some development effort expended in the use of reusable software components to engineer and integrate those components into a software system.

Proposal Evaluation

Source selection is a critical step in the acquisition process. Proper preparation and diligence in this step is key to a successful software project. There are several activities essential to successful project planning and acquisition, including risk assessment. This analysis focuses only on the schedule and cost estimate evaluation.

Proposals are typically divided into technical and cost portions. The technical proposal is carefully analyzed and evaluated by a team of application and technology experts. A second team of financial experts evaluates the cost proposal. There are two potential problems with this two-team structure. First, the two teams often perform the evaluations independently. Technical risks that impact the cost estimate are not communicated adequately, as happened in this project.

Second, the cost evaluation team is often relatively inexperienced in using software estimating methods and tools. This does not mean the team is inexperienced in financial and accounting methods. Software estimating is a specialty that requires training and experience. Training for this discipline is typically little more than keyboard training; that is, “What key do I press to get a cost profile?”

It appears the technical and cost evaluation teams were working independently during the proposal evaluations on this source selection. The reuse issue created by the overlap between the modernization program and the reusable software development should have been a major concern. The high reuse level, or extremely low size estimate, was obviously a key in the contractor’s proposal strategy. Neither the cost evaluation team, nor the technical team, questioned the high reuse percentages. The teams also failed to be concerned about the low size estimate.

Lesson 2: Technical proposal evaluation should be tightly coupled with cost and schedule evaluation. Isolation of the two activities leads to contract disaster.

A *should-cost* estimate should be completed prior to the source selection phase to establish a project plan and provide the cost evaluation team with a sanity check for the upcoming proposal evaluation. The sanity check will vary considerably as contractor capability and risk assessments are refined during source selection. The cost evaluation team should provide the should-cost estimate.

A technique to strengthen the sanity check is through using an independent third-party estimate. This type of estimate is frequently requested by the acquisition team to validate and refine the cost team estimate.

Estimating Practices

It is important to develop a reasonable estimate at the outset of any software acquisition. The estimated cost and schedule projections are vital for proper project planning, source selection, resource management, and risk management. The absence of a valid estimate is a primary cause of cost and schedule overruns, programs that spiral out of control, and failed programs. Estimate importance is often ignored or minimized in the rush to get the project underway.

A significant part of estimate failures can be attributed to a lack of understanding of the inner workings of the software development process, and the impact of that process on the parameters used in the schedule and cost estimates. One of the poorly understood variables in the development process is the impact of management on the ultimate cost and schedule of the delivered product. The style and environment imposed by the project manager is a major driver in the software equation.

Several methods of schedule and cost estimation have been available (academic and commercial) and proven since the early 1980s. These estimating methods generally consider the impact of size and the development environment on the resulting delivery schedule and resource requirements. The methods do not arrive at the resource estimates automatically. The estimator must understand the method to input correct parameters to the tool. This knowledge is only available through training and experience.

The estimating methods can also produce incorrect or misleading estimates. This project is an ideal example of estimate misuse. The contractor’s proposal estimate grossly erred in the size of the development task by overestimating the availability and benefits of reusable software components. Other key issues (other than size) were ignored in proposal estimate. These omissions included the avionics application experience and JOVIAL language experience of the remote development team. Volatility of the development environment and experience with that environment at both development sites were ignored. Communication difficulties between the sites were dismissed.

The proposal cost evaluation team noticed a large discrepancy in the proposed software cost when comparing the cost proposed by the incumbent developer and the new contractor. The cost evaluation team notified the new contractor that the team believed the contractor either did not understand the tasks or that for some other reason had not bid enough engineering hours. The contractor responded that the costs had been verified using *a proven cost model* and they did not believe they made a mistake. The contractor subsequently reduced its bid about 15 percent. An experienced software estimator would have raised a serious cost risk concern following the contractor response.

Lesson 3: Estimating skill and experience is essential in software acquisition and development.

Modern Development Practice

There has been considerable effort in establishing the importance of good software practices and a manageable development process in successful software development. The trail to modern software development begins in the 1950s (before software was born) with the work of W. E. Deming[1]. Deming’s work became a basis for the current Capability Maturity Model®. We all recognize that large-scale software development must be well managed to have any possibility of success. In the mid-1980s, the Waterfall Model represented the most commonly used software development approach. The impact of process and process management was yet to be defined outside of the software estimating methods.

One issue that arises almost constantly is the cost and schedule impact of change. A change can be as simple as changing word processors, or as complex as changing the entire development process. How long does it take to become proficient in the Ada programming language? Thirty days? It is not likely. Historic data places Ada mastery at more than a few years. How long does it take to install a new computer network? A weekend? We have a tendency as humans to trivialize the effort to master any new technology. The larger the number of concurrent changes or magnitude of a single change, the more amount of time and cost it takes to accomplish that change. This project demonstrated the human frailty.

The contractor proposed integrating in-house tools on a state-of-the-art computing system, and supplementing those tools with government-furnished equipment software to complete the development system. The proposal also stated the need to link a remote test subcontractor to the new development system. The computer program development plan (CPDP) was still in outline form at contract award. The new technology and lack of experience present in this development environment should have triggered several risk issues. Each of the issues involved personnel training and experience, system refinement, and testing. None of the environment problems were considered in the development cost.

The contractor added a new geographically remote development site for the avionics software development almost immediately after the contract award. This new organization was not mentioned in the program proposal or in the preliminary CPDP. The new remote staff was unfamiliar with the contractor organization and development process (the CPDP had not been approved), the application area (avionics), the required programming language (JOVIAL), the development tools and environment, or the network connecting the two development sites. The contractor had not successfully ported the avionics software tools to the development computer at the time of this acquisition. No cost or schedule impact was included for this set of circumstances. All major cost estimating methods available at the time assumed a major impact.

Lesson 4: Instant experience is a myth.

Software development is largely a communication problem. Paper and electronic interfaces between software engineers have not proven to be as effective as face-to-face communication. This is primarily due to interface complexity and the development product clarity. The major software estimating tools reduce the software organization's productivity for the use of multiple organizations and/or multiple development sites.

The new software development organization that was acquired at the outset of development was not only new, but was separated by thousands of miles from the contractor's primary site. The communication between these two sites was intended to be electronic – yet another new technology that was not proven. The new personnel were not only unfamiliar with the development environment, but also had no experience or knowledge of the contractor. The organization's development practices and procedures were at best documented, however, seldom followed. The computing network between the two sites was still not operational almost two years after the contract award.

Since the software estimate totally ignored these issues, as well as the experience issues, the logical assessment is they assumed the volatility of the development environment was also no problem.

Lesson 5: Multiple development sites and organizations increase risk and decrease productivity.

Summary

Software project failure has been with us for a long time. Volumes have been written about the list of potential problem areas in the acquisition of large, complex software systems. The list has not changed much in the last 30 years. Unreal cost and schedule estimates are causes for project failure as often as inadequate technology is.

The acquisition team and the contractor must share the responsibility of this classic failure. The proposal never should have been submitted, and the contract never should have been awarded. The contractor's proposal could not have been based on their experience in the development of this type of system. Buzzwords and hype often cloud judgement, but reuse has been around much longer than the hype. Great expectations overrode common sense in their cost proposal planning and estimates.

On the other hand, the proposal cost evaluation team did not have a baseline with which the proposed costs could have been compared. The team did compare the two proposed estimates, noted the large discrepancy, and acted accordingly. The cost evaluation team notified the new contractor that the team believed the contractor either did not understand the tasks or that for some other reason had not bid enough engineering hours. The contractor response was that the costs had been verified using a proven cost model, and they did not believe they made a mistake. The contractor subsequently reduced its bid about 15 percent. The response should have at least initiated a serious analysis of the proposal.

Lesson 6: The major lesson learned from this software acquisition is "never make an uninformed decision."◆

Note

1. Dr. W. Edwards Deming is known as the father of the Japanese post-war industrial revival and was regarded by many as the leading quality guru in the United States. He passed on in 1993.

About the Author

Randall W. Jensen, Ph.D., is a consultant for the Software Technology Support Center, Hill Air Force Base, with more than 40 years of practical experience as a computer professional in hardware and software development. He developed the model that underlies the Sage and the GAI SEER-SEM software cost and schedule estimating systems. Jensen received the International Society of Parametric Analysts Freiman Award for Outstanding Contributions to Parametric Estimating in 1984. He has published several computer-related texts, including "Software Engineering," and numerous software and hardware analysis papers. He is currently preparing "Extreme Software Estimating" for Prentice Hall, Inc. Jensen has a bachelor's of science degree in electrical engineering, a master's of science degree in electrical engineering, and a doctorate in electrical engineering from Utah State University.

Software Technology Support Center
6022 Fir Ave.
Bldg. 1238
Hill AFB, UT 84056-5820
Phone: (801) 775-5742
Fax: (801) 777-8069
E-mail: randall.jensen@hill.af.mil