Risk Factor: Confronting the Risks That Impact Software Project Success

Theron R. Leishman
Software Technology Support Center/Northrop Grumman

Dr. David A. Cook
The AEgis Technologies Group, Inc.

Wednesday, 21 April 2004
Track 5: 2:25 - 3:10
Room: 150 G

Every systems and software project involves risk. Often, how you manage your program risks is a deciding factor in the eventual success or failure of your program. If you ignore the risks, your program has a higher chance of failing. On the other hand, if you try to track and manage all possible risks, you can expend your entire budget managing the risks, and never produce a deliverable. Risk management, like any other element of systems and software development, requires forethought and careful planning. This article explains what risk management is, and then discusses some common developmental risks.

There is a popular television program that appeals to the brave of heart called "Fear Factor." During this program, contestants compete against each other in events and activities that cause considerable fear for most people. Although it is considered reality TV, the things the contestants are required to eat, drink, touch, dive into, or navigate through are not considered reality by the average person. Contestants are eliminated with each fear-defying competition, until a single winner remains.

In the *reality world* of software development and/or acquisition, there are circumstances, events, activities, etc. that instill fear in the hearts of software developers, acquirers, and managers. Too often these fear factors are ignored and have devastating impacts on software projects. The authors of this article have participated in the review and assessment of numerous software-intensive systems from the perspective of both developing software and purchasing software. From our experiences, we have identified several issues that are common to many software-intensive programs. These issues, *risk factors*, are keys to either the success or failure of any software-intensive system.

What Is Software Risk Management?

Risk is defined as, "A possible future event that, if it occurs, will lead to an undesirable outcome" [1]. In the context of software then, software risk management would logically imply the managing of possible future events that could have undesirable effects on software projects. That sounds simple enough. What future events could possibly negatively impact my software project? Risk management is simply defined as a generalized process for managing risks [2]. However, to be an effective risk management process, it has to be both accurate and usable; that is, it must provide results to a manager in time to allow the manager to make informed decisions that allow risks to be minimized or avoided.

Every development project has risks. The risks can range from the common, "We might not be able to find a JOVIAL programming language expert by next month," to the uncommon, "A hurricane might destroy all of the prototype aircraft." How you approach these risks is what is important. In the rest of this article, *risk* refers to software risk.

The Two Extremes of Software Risk Management

There are two extremes of software risk management: too little and too much. Each one can be explained in terms of popular operating systems.

The first extreme, the too little school, is much like older versions of UNIX. While UNIX is a fine operating system, its roots show a lack of disciplined software development. For those accustomed to a single version of an operating system, there existed numerous versions of UNIX – well over 100[1].

Here is an example of the *too little* school. One of the authors used to work on a Burroughs mini-computer back in the mid-80s. It ran a version of UNIX referred to as BNIX, which has an interesting property. The memory table that kept a list of all currently running jobs (the *process table*) had a limited size, but there was no mechanism to protect it. The size was somewhere around 1,023 entries. If you tried to run a process after the table was full, BNIX just *bumped* the counter to 1,024 and started entering process data.

The fact that the process table was full did not deter it – it just wrote to whatever memory was at the 1,024th position. Since the process table was stored in memory, it was likely (in fact, it was certain) that this 1,024th process just overwrote something critical in the operating system. Assuming that the system did not crash immediately, you could start a 1,025th process. Eventually, the operating system would crash in some bizarre fashion.

By not having mechanisms to deal with the problem, which was a limited size process table, BNIX used the *ostrich-head-in-the-sand* method of risk management. This method means you assume that all will go correctly, and you just pretend that no risks exist. Many current software development projects use this method – the equivalent of believing in magic, which often occurs because of a new and untested silver bullet. For example, "Now that we are using commercial off-the-shelf software in our program, we're going to cut the schedule by 25 percent." This is unrealistic, unproven, and head-in-the-sand thinking.

The second extreme of software risk management, the *too much* school, is more like Windows 98. Windows has many mechanisms to ensure you do not overflow process tables, overwrite system memory, or use memory that is currently allocated to other programs. However, Windows 98 sometimes gave a message that said something like, "Unable to run program – not enough memory. Try quitting a running program and try again." Unfortunately, this error message occurred when only running a single program, or even no program. What typically happened was that a previously running program had used memory incorrectly (or crashed prior to *cleaning up* after itself) and left the system memory corrupted. Windows 98 was unable to ensure that a new program would not be overwriting memory still marked as *in use*, and would not let the new program run. It usually required a reboot to set things right. This method, being *over-cautious* of the risks and always assuming that the *worst* is going to happen, is the other extreme.

To effectively manage risks, you have to

take the middle ground. You cannot ignore risks and pretend that all will go well. However, you cannot micromanage all possible risks; you do not have the time and the resources. To take the middle ground, you have to be aware of some common risk factors.

Critical Systems and Software Risk Factors

To adequately manage risks, it is essential to evaluate the program's/project's unique situation. There are, however, certain risks that tend to impact many programs/projects. Having evaluated several programs, we have identified the following critical risk factors as issues that impact the success of many programs. By addressing the key risk factors, programs can make great strides in managing risks that may impact them.

Inadequate Planning

Recently while assisting in the review of a large Department of Defense (DoD) software acquisition program, we asked to review the program's planning documents. In response to the request, the program office produced a Microsoft Project schedule. This response is common when we ask about planning documents while performing program reviews.

There appears to be a lack of desire, interest, ability, and attention to performing adequate program/project planning. Not all programs/projects require the same level of planning; however, they all should include enough planning to adequately assess their issues. Program/project plans should consider the following:

- Date and status of the plan.
- Scope of the plan.
- Issuing organization.
- References.
- Approval authority.
- Assumptions made in developing the plan.
- Planned activities and tasks.
- Policies, etc., that dictate the need for this plan.
- Micro-references – other plans or tasks referenced by the plan (other plans or task descriptions that elaborate details of this plan).
- Schedules.
- Estimates.
- Communication chains.
- Resources and their allocation.
- Responsibilities and authority.
- Risks.
- Quality control measures.
- Cost.
- Stakeholder identification.
- Interfaces among stakeholders.
- Environment/infrastructure, including safety needs.
- Training.
- Glossary.
- Change procedures and history.
- Schematics, diagrams, and architectures to further clarify the intent of the plan.

A lack of adequate planning tends to indicate a lack of forethought and direction. Inadequate planning at either a program or project level greatly increases risk to successful project completion. Elaine M. Hall [2] has a good explanation of how to plan for risk.

Unrealistic Schedules

The daughter and son-in-law of one of the authors of this article recently built a house. The couple was anxious to have the home completed so they could move in early enough in the fall to complete the yard before winter. To them this seemed a reasonable request.

As the couple negotiated the deal with the chosen contractor, they were told the house would not be completed in the time schedule desired by the couple. The contractor showed them the list of tasks to be completed and the dependency of each task with other tasks. He further explained to them the dependency on outside factors such as weather, availability of special materials that were part of the couple's design, availability of subcontractors, whims and schedules of building inspectors, and the priority of their house in relationship to other homes the developer was building. The bottom line: the couple was not able to eliminate critical steps in the building process, shortcut the availability of critical resources, or circumvent required inspections.

Have you ever been coerced into or managed a software project that begins with a predetermined schedule? One that is unachievable, to start with, that requires sound processes to be altered or eliminated? One that ignores the requirement for critical resources, eliminates critical reviews, and allows only minimal time for testing? If so, then in talk-show host and author Dr. Phil's words, *"What are you thinking?"*

Why do we agree to and/or impose software schedules that require the abandonment of everything that we have learned over the past several years as being essential to the success of software development or acquisition projects?

The authors have seen numerous software projects with schedules we consider to be unreasonable, in our opinion, due to the following:

- Schedules based on product need rather than a realistic assessment of engineering effort. This includes schedules based on an arbitrarily imposed due date.
- Effort estimates and resulting schedules based on hallucinations (or unrealistic hope) rather than on historical basis.
- Schedules based on unrealistic, incorrect, or unknown requirements.
- Schedules developed without adequate understanding of sound software engineering practices.

For these reasons, we have included unrealistic schedules as one of our software risk factors.

Unfortunately, managers who attempt to transform unreasonable schedules into accurate and reasonable ones run the additional risk of extreme upper management displeasure. This (and the risk of losing one's job) creates great pressure on lower-level managers to continue to pretend that unreasonable schedules are, in fact, achievable.

Unconstrained Requirements Growth

There have been volumes written over the years about the impact of incomplete, inaccurate, growing, unstable, and on, and on, and on, software requirements problems. In the April 2002 CROSSTALK, we enumerated several requirements risks that can drown software projects [3].

Research conducted by Capers Jones [4] identified the top five risks that threaten the success of software projects in various sectors. Figure 1 summarizes the approximate percentage of projects that suffer from creeping user requirements.

As indicated in Figure 1, the majority of management information systems and military projects suffer from requirements creep. We further saw that nearly half of the outsourced projects included in the study also suffered from requirements creep.

Our experience shows that nearly all projects suffer from some form of requirements risk. The impact is often catastrophic to the success of the project. We see requirements issues as a substantial risk to the success of many software projects.

Dysfunctional Organizational Culture

The following is the parable of the Happy and Productive Worker:

> *Once upon a time in a company not far away, there was a worker. He was a productive, happy worker, but alas, he was unsupervised.*
>
> *The company saw that the worker was unsupervised and made a supervisor.*
>
> *The coordinator saw that the worker was productive and happy and made a lead worker to make the worker more productive and happier.*
>
> *The company saw that the department in which the productive, happy worker worked*

had grown and made a manager to manage the department in which the productive, happy worker worked productively and happily.

The manager saw that the worker was productive and happy and made an assistant manager to help manage the department in which the productive, happy worker worked.

The company saw that the department where the happy, productive worker worked had grown and made an administrator to administer the department in which the productive, happy worker worked.

The administrator saw that the worker was productive and happy and made an assistant administrator to assist in administering the department in which the productive, happy worker worked.

More people were added until the director saw that the department was losing money. So he consulted a consultant. The consultant examined the department in which the productive, happy worker worked productively and happily and advised there were too many people in the department in which the productive, happy worker worked.

And the director paid heed to the counsel of the consultant and fired the productive, happy worker.

In the parable of the Happy and Productive Worker, the organization valued and rewarded the wrong things. As you think of the organizations you work in, what is the culture that exists? Is it a culture plagued with high turnover? Do you enjoy going to work each day, or does it take an act of God to get you out of bed and into the office? Do you have the training and skills required to perform the tasks that are expected of you? Are meetings useful with something actually being accomplished, or are there too many long, useless meetings that just distract from what you are trying to get done?

The organizational culture of many software companies appears to be that of working harder to get out of doing work rather than actually getting things done. This risk factor ties closely to the management. There is an inherent risk with over-managing – and perhaps this risk is greater than under-managing. Over-management forces workers to spend too much time justifying what they do (and do not do). Workers need to be allowed to fail occasionally, and learn from their failure. If you create an environment of *one mistake and you're out*, workers will spend so much time trying not to fail that they will not have the time to succeed.

During a recent seminar, we asked the attendees how many of them had ever been part of what they considered to be a high-performance team. Of 30 attendees, five indicated that they had. Upon discussion with the group, we identified the following characteristics of a high-performance team:

- Value for all team members.
- Open communication.
- Common understanding of team goals.
- Recognition for valuable contributions.
- Clear understanding of job expectations.
- Shared desire to meet or exceed expectations.
- Demonstrated commitment to the team by management.

It is difficult to successfully develop software in a dysfunctional organization. Management must look honestly at the organization to determine if it is dysfunctional. If your culture is one similar to that of the Happy Productive Worker, that culture is a serious risk to your organization's success in consistently providing quality software.

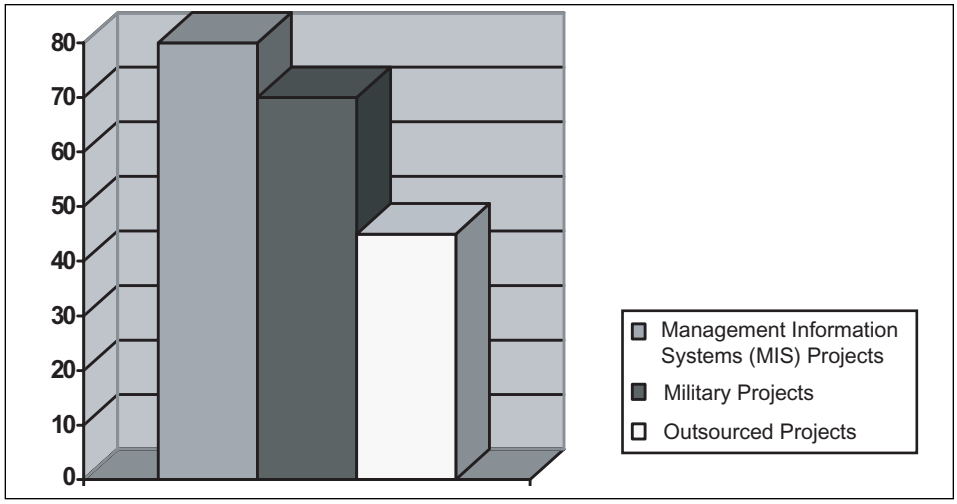

Figure 1: *Percentage of Three Types of Projects That Suffer From Requirements Creep*

Not Having or Following Processes

Having and following a good process is essential to the consistent development of quality software systems. It has been said that a software product is only as good as the process used to develop and maintain it.

There seems to be common agreement on the value of having and following sound processes in the development and maintenance of software, yet in the authors' experiences, this continues to be a weakness of many software development organizations. We see many organizations that either lack processes or do not follow their existing processes. Some have processes in place but are quick to abandon them when hit with schedule pressure.

When this happens, the good, smart things that help ensure quality software fall by the wayside and programs/projects gradually spin out of control. Not having or following processes is on our list of software risks because this is an area that is often only given lip service. If processes exist, and are required to be followed, many other risks immediately become less critical.

Failure to Actively Manage Software Risk

Have you ever thought about how much we depend on software? The world has evolved to the point where almost everything we do involves software. Lt. Gen. Jim Fain (U.S. Air Force retired), during his tenure as F-22 program director, described software's importance by saying, "The only thing you can do with an F-22 that does not require software is to take a picture of it" [5]. Now in 2004, there is probably not even a non-disposable camera in use that is not at least partially operated by software.

Software has made great technological advancement possible. This dependence on software has also brought with it consequences. Software failures have resulted in significant financial losses and even the loss of life. Software has become the very heart of the new economy, and business risk management must include software risk management to survive.

During a recent program review, one of the authors of this article asked how software risks were managed on the program. The program office produced a list from their risk database of software risks that included risks at a very high level and were generic to the point that they could be applied to any program. When asked how the list was generated, it was explained that a tool had generated the risks based on information provided. We call this risk management in a can. This was a large DoD program that included software with potential life-threatening consequences.

Our dependence on software has pushed us to the point where a proactive software risk management process is essential. Managers must determine whether any

unwanted events may occur during the development or maintenance of software, and make appropriate plans to avoid or minimize the impact of these negative consequences. Failure to do so may have devastating consequences. A good software risk management process should include the following steps:

- Identification of program-/project-specific software-related risks.
- Detailed analysis of each software risk.
- Development of plans to address software risks.
- Active monitoring and tracking of software risks.
- Use of metrics to monitor the risk-management process.

By actively managing software-related risks, the probability of experiencing first-hand the consequences of a software failure can be greatly reduced. While failure to actively manage software risk is the last risk mentioned, it is critical.

How detailed should your risk management be? On one recently performed assessment, the *risk list* on the project ran more than 400 pages, and had a complex formula showing each risk status. Such a complex risk list is almost useless. It is difficult to see the overall project risk status, and it takes a huge amount of time to keep it current.

The authors recently performed another assessment, and saw the risks briefed as a 25-item list, each with either a red, yellow, or green light (standing for high, medium, and low risk). This method was easy to use, easy to understand, and reasonably easy to update. The key point is that the program manager, using this simple 25-item list, has the information to understand the risk exposure of the program and make good decisions.

Conclusion

This article is not intended to present a complete list of risk factors for a particular program/project – it would take much more space. Likewise, it is not intended to be a primer on correct risk-management practices. What this article is intended to do is heighten your awareness of risk management, and give you a starting point in creating an appropriate risk management strategy. Risk management is usually a task best done by somebody who has some experience in the area. Locate an appropriate source of training or experience, and learn from other's mistakes. If you proactively manage risks properly, then you will spend little time reactively *putting out unexpected fires*, thus freeing up more time to build your system based on current, accurate, and easy-to-understand risk information.

During a recent "Fear Factor" episode, contestants were required to leap from one boat to another while the two boats were speeding next to each other. This resulted in some of the contestants experiencing very traumatic falls into the water.

In this article, we have identified six risk factors that, based upon our experience, can have significant impact on the success or failure of software programs and projects. By confronting these risks, many inherent problems can be avoided. We recommend the following:

- Take program/project planning seriously.
- Do not lie to yourself; develop schedules based on sound facts and proven approaches.
- Stringently control requirements growth.
- Establish a success-oriented organizational culture.
- Develop and follow sound software development, maintenance, and program/project management processes.
- Actively identify and manage risks specific to your program/project.

By doing these, you can reduce the probability of your program or project having a traumatic fall into the pit of unsuccessful software programs and projects. *The biggest risk of all is failing to manage your risks!* ◆

References

1. Leishman, T., and J. VanBuren. "The Risk of Not Being Risk Conscious: Software Risk-Management Basics." STSC Seminar Series, Hill AFB, UT., 2003.
2. Hall, Elaine M. Managing Risk. Addison-Wesley, 1998.
3. Leishman, Theron, and Dr. David A. Cook. "Requirements Risks Can Drown Software Projects." CROSSTALK 15.4 (Apr. 2002): 4-8 <www.stsc.hill.af.mil/crosstalk/2002/04/leishman.html>.
4. Jones, Capers. Assessment and Control of Software Risks. Prentice Hall, 1994.
5. Naval Postgraduate School. "Importance of Software to the Military." U.S. Navy, 4 Mar. 2000 <www.nps.navy.mil/wings/acq_topics/AcqTopics.htm>.

Note

1. See <www.scs.org/confernc/astc/astc04/cfp/astc04.htm> for a starting point.

About the Authors

Theron R. Leishman is a consultant currently under contract with the Software Technology Support Center at Hill Air Force Base, Utah. Leishman has 19 years experience in various aspects of software development. He has successfully managed software projects and performed consulting services for the Department of Defense, aerospace, manufacturing, health care, higher education, and other industries. He is a Level 2 Certified International Configuration Manager by the International Society of Configuration Management, and is employed by Northrop Grumman. Leishman has a master's degree in business administration from the University of Phoenix.

Software Technology Support Center
6022 Fir AVE, BLDG 1238
Hill AFB, UT 84056
Phone: (801) 775-5738
Fax: (801) 777-8069
E-mail: theron.leishman@hill.af.mil

David A. Cook, Ph.D., is a senior research scientist at AEgis Technologies Group, Inc., working as a verification, validation, and accreditation agent in the modelling and simulations area. He is currently supporting the Airborne Laser System Program. Cook has more than 30 years experience in software development and management. He was formerly an associate professor at the U.S. Air Force Academy, a deputy department head of the Software Professional Development Program at the Air Force Institute of Technology, and a consultant for the Software Technology Support Center. Cook has published numerous articles on software-related topics. He has a doctorate degree in computer science from Texas A&M University.

AEgis Technologies Group, Inc.
6565 Americas PKWY NE
STE 975
Albuquerque, NM 87110
Phone: (505) 881-1003
Fax: (505) 881-5003
E-mail: dcook@aegistg.com