Risk Management (Is Not) for Dummies[1]

Lt. Col. Steven R. Glazewski
Air Force Institute of Technology

Software program managers crave a silver bullet *in the form of a comprehensive checklist of things to watch so the program does not suffer from bad surprises. Highlighted in this article are some prime examples from almost 15 years' experience acquiring software in Department of Defense programs, from identifying broad areas where software risks tend to hide to describing an eight-step risk management process. While there are no silver bullets to be found, there are a few* golden nuggets *if you make the focused effort to look!*

What is risk management? We have all heard the saying, "Give a man a fish, and you feed him for a day. Teach a man to fish, and you feed him for a lifetime." Let me revise that from a risk management standpoint: "Put out a manager's fires, and you help him for a day. Teach a manager fire prevention, and you help him for a career." If a manager understands good risk management, he can worry about things other than firefighting.

Unfortunately, most people who look for risk management help are seeking to know the steps to put fires out. After all, being a good firefighter has its rewards! Take a look at your organization's person-of-the-quarter listing for the past few years. Who is on it? Typically listed is the person who put out the worst fire. What about people who avoided the fires in the first place? Therein lie the problems with good risk management: *people who avoid fires* do not get noticed, and the risks they *avoid* do not get documented.

Risks that are well understood and controlled tend not to become full-blown problems, and thus are rarely documented in risk databases. To this day, some people mistakenly believe the millions of dollars spent on Year 2000 mitigation were wasted because "nothing bad happened." This is the irony: If people die or property is destroyed, then preventative measures are deemed inadequate; if nobody is hurt and nothing is destroyed, then preventative measures are deemed valueless!

We can do a lot of damage in the name of *process* and *standardization*. Some things lend themselves well to both such as building a car on an assembly line. Some things do not such as creative, knowledge-based work like design and management. Yet we sometimes delude ourselves by creating *templates* for something like a risk management plan. Look carefully at such templates: 80 percent of the outline tends to be *boilerplate* or context setting. The meat is contained in sections that comprise only 20 percent of the table of contents entries. What does the template tell you about those *meaty* sections? Almost nothing! The real meat of a risk management plan – assembling a qualified team, devising ways to discover risks, devising methods of quantifying or categorizing the risks, and monitoring the risks – cannot be completed by simply following a checklist.

In contrast, template instructions for the non-meaty sections tend to be far more explicit (e.g., "state your funding authorization by appropriation for each fiscal year"). Usually, this information is readily available and easily culled from program management plans, status reports, organizational charts, etc. We delude ourselves into thinking that a plan is 80 percent complete when in fact we are just getting started.

"Risks that are well understood and controlled tend not to become full-blown problems, and thus are rarely documented in risk databases."

There is a subtle yet critical message implied in the above: Nobody can give you a simple risk checklist. The reality is, when people want to learn/know *how to do risk management*, they are looking for Dick-and-Jane instructions for the *meaty* 20 percent. That is, they are specifically looking for detailed steps on those things that *cannot* be determined in advance by someone who is not intimately familiar with the project and its domain and environment. Simply put, they are looking for *steps*, words like "go to the financial department and get last month's numbers and look for expenditures that lag the fiscal year spending plan." They do not want *tasks* like "monitor the expenditures to verify claimed accomplishments." The message I get is, "Do not tell me what to think about or investigate, tell me *exactly* who to see, *exactly* what to ask, *exactly* what to record, and *exactly* what to do about it. Don't make this hard – just tell me exactly what to do."

There *can* be value added from a template. But this is far more likely when the template is based on a process or procedure that is absolutely relevant to the program. For example, if you are managing an avionics modernization project, your risk plan template should come from another avionics modernization project. Not only that, but also the template should have been assessed and revised by the last project. This feedback loop is critical! If there was no feedback, then you have no idea if the template's prior users benefited from it or not. In the worst case, the very template you propose to use may have *hindered* their ability to discover, quantify, categorize, prioritize, and manage project risks – and you do not know that! Ideally, the prior users reviewed and updated their risk management plan throughout their project, and all of their lessons learned were captured – you should do this, too.

Speaking of lessons learned, I am often asked for databases of risks, or more simply, where an interested party should look for risks experienced in past programs. The answer always disappoints the inquirer for two reasons. First, the historical data that exists is typically a list of *problems*, not *risks*. Risks are undesirable events that could happen: The concern over possible glitches associated with Year 2000 is a great example. Problems are risks that came to fruition. Problems are well documented in post-mortem analyses. But good risk management – risk that did not turn into problems – is forgotten.

Second, risks – and even problems – experienced by past programs are *tuned* for the environment that existed for that program and the unique circumstances of that program. What may have been a high-priority risk for a past program may not be worth your investment of resources to monitor or track. Most people who request lessons learned do not really want a database anyway. They want the 15 or 20 items from the database that are most likely to happen to them. And they do not want to read hundreds of items to find those 15 to 20 nuggets. They are really asking me for a five-minute answer to a two-week question.

That is not to say that there is no value added in researching history. My experiences show that there is a fertile ground for finding risks – we know this because problems have consistently arisen from these areas. I have learned to focus some risk identification energy on three areas (if they are present in a project): test and integration hardware, interfaces, and reused code.

- Test and integration hardware tends to be a capacity-constraining resource. If you have a system or software integration lab (SIL), you have a potential resource conflict. Many efforts in the program seem to demand SIL time simultaneously, and usually the software developers do not have top priority. I worked on a program where the same test hardware was used to validate test software *and* to test hardware that was about to be sold to the government. Needless to say, the chance to generate revenue trumped the software developer's needs until we were able to prove that the impending delays to the project would negatively affect the contractor's bottom line by more than a little delay in cash flow. While working on a different program, I discovered that the developer's detailed schedules required over 30 hours *per day* in the SIL to meet the schedule. Scheduling tools are great, but they fail when you disable or ignore the resource conflict warnings.
- Interfaces are historically a source of error, and therefore risk. A recent big example was the Mars Climate Orbiter that crashed into the planet in 1999 because one group coded as if the measurement were in feet while the other coded as if it were in meters. Most bugs in a program are problems found while integrating modules or communicating between objects. On a grander scale in systems of systems, the biggest risks are where the independently built systems must interface. System test engineers always praise a good interface control document (ICD) more than the project managers bemoan the ICD's cost. We have a proverb that "good fences make good neighbors" and the same is true in software: If everyone knows the boundary conditions and interfaces, things go much smoother. The hard part is resisting the temptation to cut or minimize the typically large expense of creating good ICDs. ICDs are used for *inter*-system interfaces, but there are analogous – and equally valuable – design products that should describe the *intra*-system interfaces in detail.
- Reused code, which includes commercial off-the-shelf code, is often *sold* to the program as a means of drastically reducing development and test costs. Code reuse can certainly reduce costs, but only within the very narrow circumstances where you make absolutely no changes to the code, and you use it for *exactly* and *only* the purpose for which it was designed. Many potentially dangerous commercial products like pesticides now carry a standard warning such as "Use of this product in a manner other than described below is a violation of federal law." Yes, the spray is flammable – no, you should not use it to light your barbeque grill. A similar warning should accompany all attempts to reuse code, albeit only a warning that it violates sound reuse strategy, and maybe the laws of good sense. It is not a bad idea to reuse code, but you have to accept the limitations when you do. If your plans call for reusing code and you are assuming substantial time and cost savings or test simplification, you had better not tinker with the reused code (or code products) in any way, or you violate your plan/assumption and incur risk.

Of course, the risk manager must look beyond these three areas, and must apply knowledge of the project's details to determine whether any of those three areas are applicable and worthy of investing resources.

Risk management is much like being the manager of a mutual fund or a stock analyst on Wall Street. Risk managers are asked to peer into the future – to make predictions with better-than-average accuracy – to not only *be* right, but to know what to do when they *are* right. Risk management goes beyond predicting risk; it also demands planning to handle the risk once it materializes. (As a side note, think of how well paid mutual fund managers and Wall Street stock analysts are, especially the successful ones!)

How do fund managers and analysts become successful? They dig into the details of a company. They may not have complete data because the company may not release any more than the minimum required by law. Yet the manager can assemble current information about this particular company, as well as information from its recent and not-so-recent past. Information can be gathered about similar companies over time, and about the segment of the economy that affects this company. This information can then be used to make an educated guess at future earnings, profits, and trends. In other words, they develop detailed knowledge about the specific company, and compare it with a solid general knowledge about the industry and the economy. This helps them more accurately foresee profitability, which can then be used to make sound investment decisions.

This is the essence of risk management! The risk manager combines detailed knowledge of the project with general knowledge of the technical domain and the acquisition environment to foresee potential undesirable events, and to plan and take actions accordingly.

Asking a complete novice to do risk management is, well, risky. Risk management involves thoughtful, determined, and creative work to implement the following eight-step process.

Step 1: Get Time to Do Risk Management

If you are spending 95 percent of your time doing day-to-day operations, you do not have enough time to sit and think (or plan or just be creative). You need slack time – that is, time away from operations – to plan and think. For a great discussion on why, read Tom DeMarco's book *Slack* [1]. It even contains a few chapters on risk management. Sometimes, this seemingly simple step can be the hardest part. Next comes the creative part.

Step 2: Plan Your Risk Management Program

What method will you use to discover/elicit risks? Who will help? (Hint: you need those people who are intimately familiar with the project, the domain, and the environment.) What are the desired outputs of your risk analysis? How will you categorize or quantify risk? What information must be recorded for each risk? Who will use the data and how? Now comes more creativity (problem solving) and some tedium.

Step 3: Identify Risks

Gather the team and identify potential risks. Remember that the team should consist of people with lots of project and domain experience. These people tend to be senior members and are very busy, so these identification sessions should be short and controlled. Excellent administrative support is absolutely necessary! So is follow-up and coordination of results. For each risk identified, the team should describe what data they need to assess the risk. Much of that data will probably *not* be available at this meeting, which is okay. This first session is identification only.

Step 4: Assess Risks

The risk team does risk assessment. It involves a facilitator doing lots of research and legwork before another meeting with the experts. Once the data is available and pre-distributed, the team can reconvene to assess probabilities and impacts, determine indicators that a risk may be *coming true*, and prioritize the risks according to the documented procedure. The indicators are used to select metrics so the decision-maker can be proactive when choosing whether to implement handling strategies.

Step 5: Plan to Handle Risks

With the decision-maker and the team, decide how each risk will be handled. Determine what, if any, mitigation efforts are prudent; what alternative approaches or procedures are available; and/or how to share the risk. It is a good idea to identify thresholds (or trigger points) associated with the metrics selected in Step 4 so it is easier to initiate action.

Step 6: Monitor Risks

Conduct operations and periodically check to see if any of the risks show signs of turning into problems, or if any of the risks change because of the dynamics of project and environment. This period could be daily or weekly or something different, depending on how dynamic the project and environment are.

Step 7: Account for Changes in the Environment and Project

Periodically go back to Step 3. This period could be weekly or monthly or something different, depending on how dynamic the project and environment are.

Step 8: Improve Your Risk Management Process

Periodically go back to Step 2. This period could be quarterly or annually or something different, depending on how successful your program is at giving sufficient notice of things that may go wrong. This is the part that everyone hates, but it is the critical feedback loop that improves the process – for you and for the next project that uses your project as a template.

General Ideas

Here are some general ideas on risks. They must be general because I do not (and cannot) know the details of every reader's situation.

- If you cannot assign a probability, assess an impact, or draft a unique action plan, then the risk you have identified is too generic, or not a risk at all. For example, stating that the risk is "our budget will get cut" is meaningless because you cannot say what the impact is or what you would do about it. A better risk would be "next year's budget will be cut by 5 percent, which means we cannot fully fund long-lead spares." Document why you chose the numbers you did. Why 5 percent and not 8 percent or 2 percent? Why impact spares and not tech orders?
- If a risk is a near-certainty, then it is not a risk, it is something that the project's execution plan should already address. Does it?
- Risks should be prioritized according to an agreed-upon scheme. The risk team may track 100 risks. Project managers may only have time to track the top 10. Of those, the senior acquisition officials probably have time and attention for only the top two or three. Know how these lists will be derived. Are they based on probability of occurrence? Are they based on severity of impact if they do occur? Are they based on some combination of the two?
- A top 10 list should have exactly 10 items. Having 15 different *No. 1 priority* items may look good when spreading the wealth for performance review bullets, but it does nothing for helping senior people prioritize their time and the *favors* they would like to call in.
- Good risk descriptions include indicators, or some method of foreseeing that the risk may actually be coming true. The better these indicators are, the better you can prepare the contingency plans.

Finally, there are many approaches and processes to manage risk. An Internet search will turn up dozens. But remember the rule of domain applicability: If the risk management process was built by those making and assembling automobiles, it may not be well suited for a different environment such as software development. Risk management, when done correctly, consumes the time of the most experienced, most project-knowledgeable people who also happen to be the busiest and highest-paid. However, the cost and effort to prevent a fire is almost always far less than the cost and effort to rebuild after the fire is out.◆

Reference

1. Demarco, Tom. Slack: Getting Past Burnout, Busywork, and the Myth of Total Efficiency. Broadway, 9 Apr. 2002.

Note

1. The views expressed in this article are those of the author and do not necessarily reflect the official policy or position of the Air Force, Department of Defense, or the U.S. government agency.

About the Author

Lt. Col. Steven R. Glazewski is an instructor at the Air Force Institute of Technology. He teaches Professional Continuing Education courses in software project planning and execution, and software system maintenance as part of the Software Professional Development Program. Glazewski has more than 18 years in weapon system acquisition, including assignments acquiring software for the Advanced Cruise Missile, Embedded GPS/INS navigation unit, and C-5 Avionics Modernization Program, as well as experience maintaining an accredited computer model/simulation. He is an Institute of Electrical and Electronics Engineers Computer Society Certified Software Development Professional.

Air Force Institute of Technology
3100 Research BLVD
Pod 3
Kettering, OH 45420-4022
Phone: (937) 255-7777 ext. 3274
DSN: 785-7777 ext. 3274
E-mail: steven.glazewski@afit.edu