Lessons Learned From Software Engineering Consulting

Dr. David A. Cook[1]   Theron R. Leishman
AEgis Technologies Group   *Software Technology Support Center/Northrop Grumman*

The Software Technology Support Center (STSC) has provided consulting services for Department of Defense organizations since the late 1980s. During the last 15 years, literally thousands of visits have been made to various organizations in an effort to help them build and buy software better. This article talks about some of the lessons learned by two of the STSC's many consultants.

During the past years, many Software Technology Support Center (STSC) consultants have conducted program reviews and provided consultations for numerous Department of Defense (DoD) programs. After having observed many different organizations and analyzed their problems, certain patterns emerge. One of these is that there really are not any new problems. In almost all organizations that we visit, we hear, as part of the briefing, "We have unique problems here." It is difficult not to smile. In almost 100 percent of the cases, the problems are neither unique nor even difficult to uncover.

The Problems

The problems organizations have are usually grouped into the following categories:
*   Requirements.
*   Schedule.
*   People.

According to G. Weinberg, "No matter how it looks at first, it's always a people problem" [1]. Of course, it is a bit simplistic to say that it's always a people problem. *All* problems are related to people. However, certain types of problems, while people-related, are actually specific to requirements and schedule.

Requirements Problems

It comes as no secret to developers that requirements problems are common in almost every large project. It should come as no secret to managers, either. However, we are still attempting to build software by using models that require exact requirements. While organizations often speak glowingly of iterative and spiral models, and of the Rational Unified Process, the software continues to be built using waterfall models, at least from a high level. For one thing, money tends to trickle down in a single lump sum, at least in terms of a high-level budget.

Unfortunately, budgets often are set and programs are funded prior to a full understanding of the correct requirements. Requirements issues continue to plague projects, often past the coding phase and *well* into testing, integration, and even sustainment [2, 3].

The simple rule is this: Expect to have incomplete requirements well into design, and often even well past it. To mitigate this risk, these obvious actions can be taken:
*   Use a life-cycle model that permits iterative requirements gathering and incremental releases. In most projects, software needs to be developed and delivered in small increments. Yes, this often increases the cost. However, as we have learned many times, you will have to throw away at least one release – so do not make the final copy the throwaway [4]! It is usually not critical that the first version of the software is correct; it is probably not going to be, and you are going to end up throwing it away. It is much more important that the last version is correct.
*   Management must be able to shift, reallocate, and if necessary request additional budget as requirements are eventually ferreted out and documented. Early in the requirements process, it is difficult to know the extent of the complete requirements; you are unlikely to know the implementation cost. The old way – padding the budget in hope of covering unforeseen requirements – has proven inadequate. Allowing the budget to grow as requirements expand allows ramping-up of the initial version or iteration of the development, with room (and money) to grow as needed.

Schedule Problems

One of the authors recently attended a software estimation workshop that covered a few well-known rules of thumb in the cost and schedule estimation world. One is that the average programmer produces 100 lines of code per month. Another is that by taking the cube root of the lines of code and multiplying by 3, you can estimate the basic schedule of a project in months.

For example, a project estimated at 100,000 lines of code (LOC) would take approximately 1,000 staff months (100,000 divided by 100). The cube root of 1,000 is 10, and multiplying this by 3 equals a nominal time schedule of 30 months. (Again, these are very rough rules of thumb – and good software developers will have more accurate and validated rules for their particular organization. Still, as general rules of thumb, they have been independently *discovered* and validated many times).

These rules of thumb can be used to establish a rough estimation of man months and time, given a reasonably good estimate of code required. One premise of these rules is that the cited nominal schedule cannot easily be shortened. The multiplier *3* used in the example varies among schedule estimation experts. However, most experts agree that anything below a multiplier of *2.25* creates an impossible schedule. In other words, if you know the line-of-code estimate of the project, it is relatively easy to establish a minimal schedule, which according to leading experts, cannot be lowered. This permits a relatively easy *idiot test* of the schedule.

The remarkable thing is that many projects do not have rough estimates of LOC (or function points, or some other size measurement), and therefore cannot justify (or even explain) their schedules. If your project has a schedule (and delivery date) dictated by anything other than a reliable (size, schedule, and cost) estimation method, then you are likely to have an impossible schedule. Just because the software is needed by a certain date does not imply that it can be ready by that date.

Develop an understanding and respect for the complexities of software development. This understanding is invaluable in comprehending the challenges of software estimation. Have you ever noticed how simple things are, if you are not the one actually doing the work? While remodeling their house, the wife of one of

the authors considered many of the tasks associated with the remodeling as *simple, little projects*. Such simple, little projects included moving a load-bearing wall and extending an exterior wall to make rooms larger. The author's wife could not understand why the project schedule and budget were so large. These are, after all, simple, little projects! Simple – but only in the mind of a person who does not understand the principles of construction and will not be doing the work. Software development is like this!

Program/project managers and customers of software-intensive systems need to understand that software development is not a *simple, little project*. Estimates and schedules developed using sound software estimation processes and approaches should be respected by managers and customers alike, and not randomly adjusted due to whims and desires of external influences.

As a final note, Brooks Law [4] (which paraphrased, says that adding additional people to a late software project only makes it later) does not just apply to adding people. Reorganization, changing contractors, or reassigning personnel in a late project will most likely not improve the situation. However, it will provide a convenient excuse for management when things start to fail!

When the schedule starts to slip in a major way, there exist only two viable solutions. First, change the schedule. Second, reassess the requirements and provide less functionality. Of course, a poorly designed system that is into development or testing cannot easily have functionality removed without major recoding. Therefore, often the only viable solution is to change the schedule.

People Problems

As consultants at the STSC, the authors have participated in many programs' reviews. In addition, we have been asked to consult and provide help for many programs. If there is one common theme that runs throughout all of the problems we have examined, it is this: Bad management (and bad management decisions) can cripple even the best programs. The interesting thing is that in literally all of the consultations in which we have participated, the problems are known by the *folks in the trenches*. Bad management decisions, unreasonable (and impossible) demands, and poor staffing decisions could easily be *discovered* by asking developers.

Unfortunately, developers often do not have an avenue to anonymously report problems and/or concerns. Afraid of

complaining (and equally afraid of retribution) the developers gripe among themselves, but have no way to resolve the conflicts. When the problems become severe enough, outside consultants are called in, discover the problems, and summarize and report to management. The interesting thing is that management is often aware of the problem and typically makes such comments as, "We knew there were issues, but didn't know how severe the problems were." Unfortunately, by the time consultants are called, the problems are usually so severe that significant opportunities have been lost.

Do not be afraid of the truth! We have experienced some programs where it was apparent that the program mangers did not really want to hear the truth. They were content to manage the program with their heads in the sand. This type of program management is sure to kill almost any program.

To put it simply, not only are people your most important resource, but also they are your source of information. Managers, you need to set up avenues for your developers to voice issues and address concerns. Some of these avenues need to be anonymous. If you can convince your developers that valid complaints and problems are going to be addressed – with no fear of retribution – then you will have a handle on learning what the problems are.

Other Ways to Improve Your Chance of Success

As consultants, we are always amazed that the problem, while seemingly hidden from the program, is almost blindingly obvious to an outsider. This observation and others lead to the following solutions.

Do Not Forget the Simple Things

Once you successfully step back from the problem to observe, it is amazing how often the solutions rely on simple things. Past issues of CROSSTALK have covered the many simple things necessary to manage a program: risk management, requirements management, and configuration management [2, 3, 5]. Not that any of these topics are simple (far from it), but it is a simple fact that most programs will not succeed unless you have implemented risk management, requirements management, and configuration management.

Yet as consultants, we often see programs that are ignoring very fundamental areas. Someone once said of metrics that "they don't have to be 100 percent right to be useful." Well, when it comes to manag-

ing your risks, requirements, and configuration , you do not have to be 100 percent correct to be useful. As long as you manage with the right goals in mind, your project has a much better chance of succeeding [6].

Do Not Be Afraid to Ask for Help

As consultants, we often find that a major problem in programs is that many levels of the development effort, from programmers to upper management, feel that it will make them look bad if they ask for help. Managers do not want to admit that they do not have a handle on all facets of their program. Developers do not want to admit that they are not absolute masters of the nuances of either the development or target environment. The problem is rooted in the fact that most DoD projects are different from prior projects, so managers and developers alike do not have prior projects to make comparisons.

The solution is to find someone with experience. Experienced developers know that having an experienced program manager improves the chances of successfully completing a project on time and under budget. Unfortunately, experience comes from two sources – good experience, and bad experience. While there is certainly an argument to be made that a bad experience teaches very good lessons, nobody wants bad experiences on their project. To find others with good experience you have to admit that you need help. This is often easier to do with an impartial, outside observer than with peers.

Impartial observers do not necessarily have to come from outside your company – only from outside of your immediate group. In most large organizations, you can find mentors or other sources of help. This type of *cross-pollenization* not only helps your group, but can also help the entire organization by sharing viewpoints and experience.

Do not be afraid to request (and in some cases, demand) necessary training. Having new tools and languages do not help if your people do not know how to effectively utilize them.

Call In an Outsider

When you are deeply part of the problem, sometimes you cannot step back far enough to see the solution. In many software projects, independent verification and validation (IV&V) is used to assure quality. However, IV&V is usually called in after the software is finished. It helps, instead, to have an outside observer assist prior to program completion. Some projects are like this scene in Winnie-the-Pooh:

Here is Edward Bear, coming downstairs, bump, bump, on the back of his head, behind Christopher Robin. It is, as far as he knows, the only way of coming downstairs, but sometimes he feels that there really is another way, if only he could stop bumping for a moment and think of it. [7]

We could often solve our problems if only we could stop banging our heads against a wall and think about it. However, without a truly objective outside observer, we often cannot see how to stop banging our heads.

It is a well-known fact that outside consultants are more influential. This feeling goes back to the days of the Bible: "Prophets are honored by everyone, except the people of their hometown and their own family" [8]. Outside observers are often useful in getting across the same message that you have been trying to convey for years – their status as an outsider gives them the credibility to get your message across.

As with outside observers, mentors, trainers, and expert advice, consultants can be used to *help you over the rough spots*. The cost of hiring a consultant is often paid back many-fold by the savings in preventing rework.

Summary

Are these guidelines enough to keep your program on track? Of course not! In reality, they are at best, general guidelines. They are based on an examination of many software projects under varying constraints. Many important issues such as design and implementation have not been covered – these usually are not major issues that the STSC sees. Other issues such as testing and verification and validation are very important, but the authors feel these issues have been adequately covered in other CROSSTALK articles.

We do feel that the above guidelines are valid advice that might help you in your project. Certainly, software development is far too complex to be summed up in a few so-called simple rules.

• Have a strong risk management and risk mitigation plan.
• Implement configuration management.
• Use risk management and configuration together to proactively predict upcoming changes, thus mitigating major cost and schedule impacts.
• Focus on the product and quality, and modify the process to accommodate the needs of your program.

Do not be afraid to ask for help – it is far easier to learn from the experiences of others than to be forced to repeat all of the same mistakes. Also it is usually far cheaper to use a consultant's experience to help you avoid mistakes and errors.◆

References
1. Weinberg, Gerald M. The Secrets of Consulting. New York: Dorset House, 1995.
2. Leishman, Theron R., and Dr. David A. Cook. "Requirements Risk Can Drown Software Projects." CROSSTALK Apr. 2002: 4-8.
3. Van Buren, Jim, and Dr. David A. Cook. "Experiences in the Adoption of Requirements Engineering Technologies." CROSSTALK Dec. 1998: 3-9.
4. Brooks Jr., Frederick P. The Mythical Man-Month, Anniversary Edition. New York: Addison-Wesley, 1995.
5. Leishman, Theron, and Dr. David A. Cook. "But I Only Changed One Line of Code!" CROSSTALK Jan. 2003: 20-23.
6. Cook, Dr. David A. "Confusing Process and Product: Why the Quality Is Not There Yet!" CROSSTALK July 1999: 27-29.
7. Milne, A. A. The Complete Tales of Winnie-the-Pooh. 1926. Penguin USA, Oct. 1996.
8. The Bible. Clear English Version, Matthew 13:57.

Note
1. At the time this article was written, Dr. David A. Cook was the principle engineering consultant for Shim Enterprise, Inc., and under contract for the U.S. Air Force Software Technology Support Center.

About the Authors

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 modeling and simulations area. He is currently the modeling and simulation liaison between the Missile Defense Agency and the Airborne Laser System Program Office. Previously, he was the principal engineering consultant for Shim Enterprise, Inc., and under contract for the U.S. Air Force Software Technology Support Center for over six years. 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 former deputy department head of the Software Professional Development Program at the Air Force Institute of Technology, and has published numerous articles on software-related topics. Cook has a doctorate degree in computer science from Texas A&M University, and is an authorized Personal Software Process℠ instructor.

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

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. This experience has provided a strong background in systems analysis, design, development, project management, and software process improvement. 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