Why Can’t We Manage Large Projects?

Watts S. Humphrey
Software Engineering Institute

Changing managers, procurement regulations, acquisition procedures, or contracting provisions have not resolved the cost and schedule problems of large-scale system development. This article shows the problems that organizations face with large system projects—and how one government organization has succeeded, over a period of several years, using the Team Software Process (TSP).

The Naval Oceanographic Office (NAVO) Systems Integration Division began working with the SEI 15 years ago. Their group produces software for a range of systems that supply oceanographic and meteorological data to the U.S. Navy's worldwide fleet. These enormous terabyte systems that operate 24-7, and their subsystems provide critical operational information to almost every branch of the Navy.

Ed Battle—branch head then, and now Systems Integration Division director—recalled that when they started working with the SEI, projects were always late, requirements were frequently misunderstood or wrong, and there was no cooperation among the many interdependent groups. When critical delivery dates approached, the director tracked the work with regular Monday, Wednesday, and Friday status meetings. While these meetings raised the pressure and took a lot of time, they didn't shed much light on project status.

Battle’s question to us at the SEI was: “Isn’t there a better way?”

The Large System Problem

The problems Battle’s group faced are typical. Large system projects fail all the time and the larger they are, the more likely they are to fail. For example, the new IRS system was five years late when it was first used in 2005, but its costs had exploded to $2 billion. A recent Government Accountability Office defense acquisition assessment of 72 typical weapons programs found that the development costs had climbed 40 percent from the first estimates, there was an average delay of 21 months, and the total systems overrun was $2 billion.

The situation is even worse for truly massive systems programs, as the New York Times also recently reported: Two-thirds of the largest weapons systems ran over their budgets last year, for a combined extra cost of $296 billion. These programs were, on average, almost two years behind schedule.

Problem Causes

Studies show that these development problems are typically not caused by technology issues but are largely due to program management. Unfortunately, the common reaction to program management problems is to replace the program managers. This blame-based culture stifles communication and fosters an opaque and defensive management style. We have been changing managers for years, but it should now be obvious that the problem isn’t bad managers: They are good people put in untenable positions.

“We have been changing managers for years, but it should now be obvious that the problem isn’t bad managers: They are good people put in untenable positions.”

For example, the replacement FBI system was recently killed when it fell three years behind schedule and after the project had spent $150 million. The program had a total of five CIOs and nine program managers. Clearly, changing managers did not fix the FBI’s problems. But neither did changing acquisition systems, reorganizing the Pentagon, or modifying procedures. Projects keep failing. In fact, more and more large projects fail these days than in the past—and the failures are even more expensive and painful.

The common view is that the program manager is responsible for doing whatever is required to get the job done. If new management or technical methods were needed, he or she should put them in place or take whatever steps were needed to do so. But the fact that these large projects keep failing suggests that program managers don’t know what to do. However, we must do something and it should by now be clear that relying on program managers to fix these projects isn’t working. This article suggests how to address these problems in a way that program managers can implement today.

Knowledge Work

We explained to the NAVO that the problems with software work were an early indicator of the problems that would soon plague all aspects of modern engineering work. Software has been hard to manage since the beginning, but the reason has nothing to do with the technology. The reason is that software is a different kind of work.

For the more traditional work of the past, the managers could walk around the lab or plant and see what was going on. This is called management by walking around (MBWA), a very effective way to keep management informed about the work and for keeping the workers on their toes. However, the principal problem with MBWA is that it is only effective for work that one can understand by watching the workers do it. Today, most sophisticated technical work is more like software: A great deal of the creative effort is done on a computer or in a worker’s head, and results are largely invisible to the casual observer. Peter Drucker, the first to describe knowledge work, said that it is work with the mind rather than with the hands. The products, instead of being things you can touch and feel, are ideas. While these ideas may ultimately be embodied in physical products, the bulk of the work, and the true product value, is in the creative effort required to develop

Team Software Process and TSP are service marks of Carnegie Mellon University.

these ideas and transform them into mar- 

**Traditional Management**

Even though the workers and much of 

their work is vastly different from 100 

years ago, today's traditional management 

methods are still based largely on the prin-

ciples from Fredrick Winslow Taylor's 


Management” [5]. Taylor's methods were 

designed for uneducated workers and the 

relatively simple manual tasks of the past. 

The kind of work and the skills and meth-

ods involved in much of today's work are 

quite different, but today's management 

methods still follow Taylor's command 

and control principles. Unfortunately, 

with software and most other sophisticated 

technical work, these methods are not 
effective in controlling project costs, 

schedules, or quality. While the managers 

may try valiantly to manage the work, they 
cannot know what the knowledge workers 

are doing or how they are doing it.

The end result is that today's managers 
cannot truly manage their knowledge-

working projects. That means that these 

projects are not being managed, and 
everybody knows that unmanaged pro-
jects usually fail. Unfortunately, the man-
agers are generally blamed for the failures 

when the real problem is with the man-
agement system—and not the managers. 
The answer is not to replace potentially 

very capable managers, but to change the 

management methods. Program man-

agers, however, typically do not know 

what changes to make and are under-

standably reluctant to change to a new 

management method that is not in gener-
al use by other similar programs.

**Managing Knowledge Work**

In considering how to manage knowledge 

work, Drucker concluded that since man-

agers cannot truly manage such work, the 

knowledge workers must manage them-

selves. While many managers say that they 

already involve their people in their own 

management, involvement is quite differ-

cent from responsibility. To truly manage 

themselves, the knowledge workers must 

be trained in personal and team manage-

ment methods and they must be held 

responsible for producing their own plans, 

negotiating their own commitments, and 

meeting these commitments with quality 

products. The manager's job is no longer 

to manage the knowledge-working teams 

but to lead, motivate, support, and coach 

them.

Software teams like to work this way. 

Where once they struggled to meet man-

agement's schedule targets, they now 
negotiate their own commitments with 

management. The teams feel personally 

responsible for and in control of their 

work, they know project status, and they 
have the data to defend their estimates. 

When they see problems, they resolve 

them or get management's help. Further-

more, when the knowledge workers mea-
sure, track, and report on their work, the 

managers have the data to help them 

resolve problems. Then the entire man-

agement system can participate in making 

their programs successful.

When knowledge-working teams have 

appropriate management, training, and 
support, they can work in this way (see the 

sidebar for the principles of knowledge 

management). Then they consistently meet 

their cost and schedule commitments with 

high-quality products. What's more, these 

identical knowledge-working principles 

can be applied to all of the engineering 

projects in an organization, producing a 

measurable and trackable knowledge-work 

management process across a large pro-

gram or even an entire organization.

**Workplace Objectives**

One of the more fundamental problems 

with current management practices is that 

the workers and managers have different 

views of project success. Studies show 

that product developers view a project as 

successful if the work was technically 

interesting and they worked on a cohesive 

and supportive team [6]. This was true 

whether or not the project met its cost or 

schedule objectives. Conversely, the man-

agers viewed projects as successful if they 
melt their cost and schedule targets with 

little regard for the nature of the technical 

work or the working team environment. 

This difference in workplace objectives 

has a profound effect on program man-

agement. For example, when the program 

manager wants to know when some large 

program will finish, he or she asks the 

project leaders. They then talk to their 

team members. The team members view 

the schedule as management's problem, 

however, and give vague answers such as 

“I'm almost through the design,” or “Just 
a couple more bugs and I'll finish testing.”

While the knowledge workers are typically 

the first to sense that a project is in sched-

ule trouble, they have no way to precisely 

describe job status. Rather than say some-
thing and risk getting involved in a lot of 

management debates, knowledge workers 

would rather concentrate on their techni-

cal work and leave the schedule problems 

for their managers.

**The Surprise Problem**

Fred Brooks once said, “Projects slip a day 
at a time” [7]. To keep their projects 
on schedule, all that managers have to do is 

make sure that their teams recover from 

these one-day slips every day. With large-

scale knowledge work, however, the man-

agers can't see these small daily problems 

and the developers don't have the data to 

describe them. As a result, the managers 

can't take action to recover from the one-
day slips. By the time the schedule slips are 

large enough to be visible, it is too late to 
do anything about them. This is why pro-
jects that are run by very capable and expe-
ranced managers keep having cost, sched-
ule, and quality problems. The managers 

don't have the feedback they need to see 

problems in time to prevent them. It is as 

if they were driving a car at a high speed in 
a dense fog. Once they see a problem, it is 

right in front of them, and they must make 
a panicked effort to avoid a crash. Today, 
in large systems projects, the managers are 

driving fast in a fog—and crashes happen 

all the time.

By the time more senior managers see 

these project crashes, the schedule delays

---

**Management Principles for Knowledge Work**

The management principles for knowledge work are fundamentally different from 

those for traditional engineering. The five management principles for knowledge 

work—which were adopted from my forthcoming book “Leadership, Teamwork, and 

Trust: Building a Competitive Software Capability”—are as follows:

1. **Trust the knowledge workers.** Management must trust the knowledge workers 

   and teams to manage themselves.

2. **Build trustworthy teams.** The knowledge-working teams must be trustworthy. 

   That is, they must be willing and able to manage themselves.

3. **Rely on facts and data.** The management system must rely on facts and dates—

   rather than status and seniority—when making decisions.

4. **Manage quality.** Quality must be the organization's highest priority.

5. **Provide leadership.** Management must provide their knowledge workers with 

   the leadership and support they need to manage themselves.
are typically quite significant. Furthermore, on a large project with many interdependencies, delays in any one part will affect many others. This means that many parts of a large program will probably get into schedule problems at about the same time. The managers of the many parts of the program then face a difficult choice: be the first to admit to schedule problems or wait for someone else to get into trouble first.

Blame-Based Management

Unfortunately, with the current system, senior leadership tends to blame the managers for management problems. By being the first to admit problems, the managers could easily be blamed for the entire program’s problems. Not surprisingly, most managers decide to concentrate on the problems they can solve and wait for someone else to blow the whistle. By the time the problems are visible to senior leadership, the program is in such serious trouble that there is no chance to recover. Then everyone upstairs is surprised.

The combination of a blame-based management system and the lack of precise project status measures motivates both opaque management and a general reluctance to admit to problems. With large and complex systems programs, every part is important: Problems anywhere can delay everyone. That is why every component element of the work must be managed and tracked and why every team must strive to meet all of its commitments. That is also why, without precise status information, all estimates and commitments at the team level (and, for that matter every higher level) are just guesses. Finally, that is why, with today’s typical management systems, large projects are always late and over budget.

The NAVO and the TSP

After we had reviewed these points with Battle and his associates, he agreed that it all sounded very reasonable—but wondered how it would help him and the other managers keep their large programs on schedule. We explained that the SEI had developed a knowledge-working process called the TSP, and that one of its principal features was that its management system was based on precise, operational-level data [8]. With the TSP, the developers gather and use data to manage their own work, and they use their data to accurately measure project status to within fractions of a day. TSP teams report their status to management every week, and management can see exactly where every element of every project stands. With precise status information, management can see small cost and schedule problems before they become serious. They can then take timely action to identify and resolve the problems.

When knowledge workers have been trained and know how to manage themselves, they have detailed plans and know project status precisely. They also feel responsible for managing their own problems and, when they need help, can call on their teammates or, if needed, on management.

“No process can eliminate problems ... But with sufficient warning, recovery actions are almost always possible—and most of the problems can be avoided or resolved without a crash.”

The NAVO Experience

When the NAVO started working with the SEI, they originally used the Capability Maturity Model® (CMM®). It was helpful, but gave them the what when they needed help with the how—and it was difficult to implement. On the other hand, the NAVO found that the TSP was a better fit, with the guidance they needed to properly manage their projects. It also provided for rapid training (initial team-member training takes a week), with teams soon after launching the TSP and managing themselves.

Once the teams were using the TSP, the benefits of better planning, tracking, and reduced test time were immediately apparent. Many organizations even found that the savings from just the first project pay for that team’s entire training and introduction costs. The team can then continue using it without any further training investment.

After using the TSP for several years, Battle reported that their product quality levels have improved by about 10 times and that testing times have been reduced from months to weeks. Schedule and cost performance is much more predictable than before, and the Monday, Wednesday, and Friday weekly status meetings are no longer needed. Team cooperation and coordination was also greatly improved. Battle’s final conclusion was that, “This is the only way to manage large knowledge-working projects.”
Conclusions
The consistent failure of large-scale development programs not only costs a lot of time and money, it delays the introduction of promising new technology and deprives our fighting forces of the tools they need to protect our nation. By now it should be obvious that the U.S. defense industry lacks the motivation to address this problem. For example, a mid-level executive of a major defense contractor recently told me that he could not afford to use high quality development methods like the TSP because it would reduce his revenue. His organization gets paid when they overrun projects and they get new contracts to fix their defective products. If this executive eliminated this source of revenue, he would lose his job. One could argue that the answer to this situation would be fixed-price contracts, but this approach has been tried several times in the last 50 years and has not solved the problem. It merely converts technical issues into contract disputes and the contractors get paid anyway.

Similarly, the program managers can’t solve this problem. Even if they were familiar with the TSP and convinced that it would work, they would be reluctant to try something before it had been widely used by other programs or recommended by acquisition management. The TSP has a proven record of success and it could help to address this problem right now. The DoD—or some other government agency—should evaluate or test the TSP and other promising methods to determine their suitability. It should then determine the best methods to use in managing these large programs and recommend that program managers require their contractors to use these methods. This should not be an expensive or time-consuming effort. Large-scale systems development is too critical a national problem to ignore—and the savings could be enormous.

References

Additional Resources

Software Defense Application
There aren’t many organizations bigger than the defense industry—and none with a bigger need for success in their large-scale development programs—where failure can have billion-dollar financial impacts and, worse yet, present dangerous security vulnerabilities. TSP creator Watts S. Humphrey, whose groundbreaking 2000 report outlining the TSP (see <www.sei.cmu.edu/reports/00tr023.pdf>) was sponsored by the DoD, feels that our defense industry can benefit significantly more from the process. Through past experiences, and the success of an organization providing oceanographic products and services to all DoD elements, Humphrey shows how and why the DoD needs the TSP now more than ever.

Note

About the Author
Watts S. Humphrey joined the SEI after his retirement from IBM. He established the SEI’s Process Program and led development of the CMM for Software, the PSP, and the TSP. At IBM, he managed their commercial software development and was vice president of technical development. He is a fellow for the SEI, the Association of Computing Machinery, and the IEEE. He is also a past member of the Malcolm Baldrige National Quality Award Board of Examiners. In 2005, President George W. Bush awarded Humphrey the prestigious National Medal of Technology for his contributions to the software engineering community. He holds master’s degrees in physics and business administration and an honorary doctorate in software engineering.

SEI
4500 Fifth AVE
Pittsburgh, PA 15213-2612
Phone: (412) 268-6379
E-mail: watts@sei.cmu.edu