

Evolving an Engineering Process for Software

In "The Technology Disconnect" (Wall Street Journal, Manager's Journal, 17 Dec 2002) Mr. Faisal Hoque laments the "dismal state of affairs" that "CEOs have squandered (hundreds of billions of dollars) on lousy technology investments", but they do not "feel any smarter about how to get actual value from all that money." He prescribes "communication" as the cure, citing a project at DuPont as evidence.

While communication is necessary to the success of IT projects, it is certainly not sufficient. This can be seen from countless projects that have failed, despite a strong emphasis on communication. So what more is needed beyond communication? Interestingly, Mr. Hoque touches on the answer to this question, apparently without realizing that it is indeed the heart of the issue. He suggests that we "look at one area where technicians and business people successfully collaborate every day: the engineering process."

Communication alone cannot solve our IT project problems. In order for communication to be effective, it must involve the right parties and be based on a vocabulary that all of those parties understand. Mr. Hoque points to the needs of the people who will ultimately use the IT system as the subject of the needed communication. That is indeed a good starting point. But if we do as he suggests and look at the "engineering process", we will see that the requisite types of communication go far beyond that humble beginning. Indeed, the most critical communication is between the technicians and the business people. And the engineering process itself provides the basis and vocabulary for that communication.

With this understanding of the role of the engineering process, we can see that the root of our IT problems lies in the fact that software is only an *aspiring* engineering discipline. Mary Shaw (Alan J. Perlis Professor of Computer Science at Carnegie Mellon University and former Chief Technology Officer of the Software Engineering Institute) has pointed out that just as Mechanical Engineering developed over a number of centuries before it achieved the status of an "engineering discipline", so software as an engineering discipline is still in its formative stages. Consider the fact that it was in April 2001, that the IEEE Computer Society first published the Public Draft (Trial version 1.00) of the Software Engineering Body of Knowledge (SWEBOK). This draft has met with significant controversy and heated discussion that continues to this date. The current state of affairs has even caused some states in the US to make it illegal to label software as an engineering discipline!

Clearly it will be quite some time before the basis for a Software Engineering Discipline is generally accepted. But even in these early stages of software's rise to the status of an engineering discipline, we can still begin to benefit from the application of engineering rigor to our IT projects. The Software Engineering Institute (SEI) has shown us the steps to doing just that.

Disciplined Project Management

The first step is for technicians and business people alike to believe that software is no different from any other technology in its need for disciplined project planning and oversight. Too often

Evolving an Engineering Process for Software

we ignore planning and estimating methods that have been well known in the software industry for decades, and instead build "plans" that are little more than window-dressing designed to achieve arbitrary cost and schedule limits. Sometimes we don't even believe in these "plans", but we adopt them anyway, beginning our own organizational "death-marches". But this need not be the case.

After communicating with the ultimate users of the system, we can use well-known methods for establishing a baseline description of those needs from which the development work can proceed. And since we know that changes to that description will inevitably be required as both we and the end users learn more about the system that is under construction, we can use well-known methods for managing those changes. Ultimately, we can use long-established methods to verify that the system as it is being built will satisfy those needs as they are described.

Little of what we do in IT is groundbreaking. But in those cases where we must build a system that uses unproven technologies, we can plan for and use well-known methods to manage the inherent risks. We can capitalize on the work of the Project Management Institute (PMI), using the risk planning and management methods outlined in their Project Management Body of Knowledge (PMBOK) Guide®. We can also plan for and use proven engineering methods such as prototyping and incremental development to mitigate those risks.

Our plans and schedules need not be arbitrary. Most organizations have enough historical data to estimate the time, effort and resources required to complete a project that is similar to those they have completed in the past. Even if such data cannot be found, using publicly available industry averages will usually result in plans within 30% of actual results (which is significantly better than most organizations can claim). Once we have decided to plan based on historical data, we can begin to collect the critical performance metrics from our projects that will allow us to plan in the future within 10% of actual results. Again, both the PMI and the SEI have provided ample guidance on these topics.

After we have established realistic plans that are sufficiently quantitative, managing IT projects becomes little different from managing any other kind of project. We can monitor leading indicators, seek more information when those indicators suggest problems, and use historical data and our plans and actual results to diagnose problems and take appropriate corrective action.

Neither managers nor technologists need to continue to endure IT project failures that squander precious resources and erode confidence in our ability to tackle tomorrow's problems. Applying well-known methods for planning and managing IT projects can foster communication between technicians and business people that will allow us to be more successful more often. And in the case when a project cannot succeed, the same methods will give us the information we need to pull the plug before a realized risk becomes a full-blown disaster.

Acting More Like Engineers

But disciplined project management is just the beginning. After we have achieved this level of basic control over our IT projects, we can go on to capitalize on the emerging discipline of Software Engineering.

Evolving an Engineering Process for Software

We can keep a critical eye on the methods and processes we use to assure that they are producing the results we expect and are fostering the communication we require. Our engineers can suggest improvements to our methods and processes that would make their jobs easier and their work better. We can investigate new processes and methods as they are published, determine how they would fit into our existing structures and methods and decide if they should become part of how we usually approach software projects. All of these things will result in our organization establishing its set of standard processes and methods that are appropriate for the types of projects on which it embarks.

Beyond that, our programmers can start to mature into true Software Engineers. Using techniques like the SEI's Personal Software Process and Team Software Process (PSP/TSP), they can gain control over their personal engineering work and develop a vocabulary for communicating with business people. They can produce surprisingly accurate estimates and plans based on their personal engineering data. And they can produce nearly defect-free programs using the best engineering techniques (e.g., design, design verification, development, review). And "true" engineers at heart will go on to constantly monitor and improve their own professional performance.

As our software groups mature into software engineering organizations, the benefits will only grow. The data we collect on our engineering work will provide the basis for statistical analysis of our projects and the software they produce. That analysis is the foundation for continuously optimizing our organizations' performance. As the production of software becomes an engineering discipline, our organizations will be poised to capitalize on advances in software engineering as the discipline itself matures.

Getting Started

Organizations that have actually done all of these things did not do them in one grand step. Just as people mature slowly over many years, these organizations have built skill upon skill and process upon process to get to the point of performing software development as an engineering activity. The starting point for all of them was the same: disciplined project management.

The good news is that disciplined project management processes and methods have been well known and used in other disciplines for decades. Many companies have demonstrated that applying these proven techniques to software works well.

The bad news is that it means changing the way we run software projects. Business people and technicians alike will have to be trained to use these techniques, and the company's culture will have to change to embrace and support their use. Although it sounds easy to "just do it", actually making the necessary changes tends to be the hardest part. It requires the active support of the senior executive, determination on the part of the management team, and often, a change in the reward system to encourage the new behaviors.

In short, it requires investment. But that investment will pay big dividends as we avoid the waste of our all-too-common IT disasters, and even bigger dividends as our IT organization matures into a smooth-running software engineering function. It all starts with the determination to move

Evolving an Engineering Process for Software

our organizations from the 20th century methods of software development into the 21st century world of software engineering.