

When management can't or won't make needed process improvements, the practitioners in the trenches must take the initiative. The author presents several techniques for implementing SPI from the bottom up.

Bottom-up Process Improvement Tricks

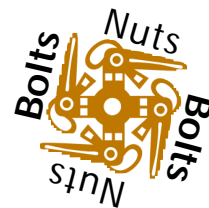
Allan Baktoft Jakobsen, DELTA SOFTWARE ENGINEERING



Most everyone in our field acknowledges that software process improvement cannot succeed unless management is committed to implementing it. Fortunately, the move to SPI can be a two-way street, initiated as easily from the bottom up as from the top down. Taking this bottom-up approach, I present some tricks that practitioners down in the trenches can use to win upper management's approval of and support for SPI.

My first opportunity to introduce bottom-up process changes occurred on a project in which my job was basically coding and testing. Initially, I was assigned to the maintenance coding of the product's previous release, which proved to be a real headache. So I had my motives for making improvements.

According to the waterfall model we were using, the project was about halfway through the design phase. A lot of lower-level design and unit and integration testing remained. I was assigned one part of the system, and the remaining design and unit testing tasks were given to the 15 people we had allocated to us for the next five months. Given the amount of work that lay before us, these resources were not nearly enough. One subproject leader realized that our complex and lengthy design task would prove a huge challenge.



His nose for risk analysis told him that our only hope was to try something new. So he gave a couple of us at the bottom a free hand to do the remaining project planning. Then he closed his office door and started looking for another job. I am still grateful to this man, because his lack of management involvement created a vacuum we were able to fill with bottom-up process improvements.

We began by shifting to an incremental development model. This model suggests doing the lower-level design and test in an iterative manner. The system is grown inside-out, in increments, starting with the core functionality. Each increment has its own small development cycle of design, coding, and unit test. This system-driven approach calls for system-specific planning, which is formulated into the construction plan. The plan is essential because it defines the contents and, more importantly, the order of the increments. The key criterion for a good construction plan is that the system of accumulated increments should, at any time, represent a working whole.

By citing this specific example, I don't mean to appear religious about process models. Each model has its strengths and weaknesses, and any particular model's success depends to a great extent on the specific nature of the project it is applied to. We chose incremental development because it can prepare code for integration testing months earlier than can some other methods. Then again, this method takes much more coordination. By project's end I had received some 1.5 Mbytes of e-mail from my co-workers. Because we trusted our ability to coordinate, we felt that incremental development would be the best model to help us meet our deadline. Along the way, we learned the following tricks for implementing process improvement from the bottom up.

Think success opportunities

Our successes fuel us. Many people become addicted to their successes and thus can be powerfully motivated by a chance to achieve more of them. Fortunately, what constitutes a success and how often it must be achieved varies from person to person. Knowing this, you can study your team members and give each of them plenty of opportunities to succeed.

With this in mind, before I arranged our first common meeting I spoke individually and at length with each of the 15 people assigned to the project. One had extensive spare-time activities and was concerned with the heavy overtime work required during the previous project. Another clearly sought to be a manager some-

day. Some wanted to improve the software's quality, while others who were new to the company wanted a shortcut to competence. Having learned all this and addressed their concerns in advance, I knew that when we had the first group meeting everyone would be receptive to my message. I won the first battle simply by doing some advance legwork.

Avoid Jeopardy

Maybe you've seen the TV show *Jeopardy*—the one in which the quiz master gives the answer and the par-

Many people become addicted to their successes and thus can be powerfully motivated by a chance to achieve more of them.

ticipants must guess the question. Certain software companies also play this game. It's a big mistake.

Too often, I've seen quality departments and other process-conscious people operate with the hidden slogan: "We bring the solutions, you guess the problem." Most CMM level 1 and 2 companies don't have the maturity to play this game. Worse, these organizations receive the solutions packaged as so-called processes that cannot be unpacked or applied to their problems. The problems themselves and the mapping of solutions to them are seldom documented. Thus, as time goes by, people become less conscious of the dangers in misapplying a solution and fall back on just following the process rules.

To prepare the way for a process model change, we tried to focus on the problems, but I admit we could have avoided *Jeopardy's* answer focus more extensively. If everybody had a common and intimate knowledge of the development problems facing us, we could have implemented an SPI that would have taught any rocket scientist a lesson or two.

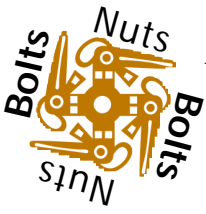
Being there

Among all the various interests we pursue in the software industry, you typically find a handful of us who care most deeply about quality. When management recognizes that a quality focus benefits the company, they often do one of two things:

- ◆ centralize these few quality-conscious people into a special group, such as a software engineering process group; or

- ◆ sprinkle them across the important projects.

Unfortunately, the first action often results in isolation¹ and an inbreeding of ideas, while the second dilutes coordination and lessens effectiveness.



You really must be present when implementing a significant process change. You must watch over it and tailor the initiatives to each team member. You must decentralize your efforts in *space* so that they filter down to each participant.

The trick is you can still centralize your efforts in *time*. Every now and then, gather those few quality-conscious people for a whole day of intensive workshops on coordination and education.

In our project, we chose one person in each team to assume the special role of caring about quality. The team called this person the process manager. He would basically keep an eye on the process and the review records. Because the top project leader considered progress a higher priority than process improvement, he loved it when we referred to these people as the *progress managers*.

Unfortunately, as so often happens in real life, the ideal of theory fell short when applied to the reality of practice. We simply didn't have the time or resources to give these people the proper education, and one team suffered severely from this because they couldn't learn and implement the entire process.

Rhythm's power

Most people resist change, especially frequent change. We feel safe with the everyday rhythm of our lives; this enables us to do many of our daily tasks without much thought. These habits give us time to reflect on the past and plan for the future. But habits can be deadly obstacles to process improvement.

The ancient discipline of martial arts provides a solution: use an attacker's energy against him. In our case, the question became how could we use the energy of routine rhythms to implement process changes? The trick was to change from *process-driven* to *time-driven* activities.

Ideally, teamwork ensures a high degree of interdependency, mutually beneficial cooperation, and a synergistic meshing of complimentary skills.

We applied this technique from the very start of our implementing the incremental development process. First, we urged people to hold off on being overly critical until they gave the new process a chance. At the same time, we encouraged the transition from the old organization to the new one by relocating people so that members of the new workgroups shared adjacent offices. We next insisted on holding review meetings every Monday, Wednesday, and Friday, telling people

"Give us what you have at 9 a.m. sharp and we'll review it by 1 p.m." When some project members objected that they wouldn't have their tasks finished by then, we replied, "That doesn't really matter. Let's review it anyway." After two weeks, people got into the habit of having many small meetings, sometimes for just half an hour. After four weeks, they were hooked and wouldn't pass up a meeting for any reason. We had found our project's pulse.

Reviews as medicine

Small, informal review meetings—in which you focus on some part of a team member's work and allow open discussion of it—can be just the medicine to cure or even prevent the nasty illnesses that often beset a software project.

Such small reviews are excellent for spreading knowledge to less experienced employees and keeping them on track. Moreover, reviews are important social events that promote teamwork, because it's here that team successes are crystallized and shared. As an added bonus, reviews are excellent at finding many kinds of defects.

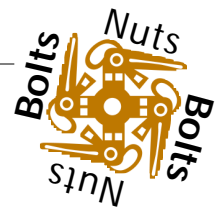
Applied correctly, reviews provide a powerful vaccine for software projects. And they are not time consuming, but merely a matter of converting unstructured activities into structured ones that reduce rework.

High on teamwork

To be effective, teamwork must operate on the principle that your success is my success. Ideally, teamwork ensures a high degree of interdependency, mutually beneficial cooperation, and a synergistic meshing of complementary skills. Seeing other people directly add value to your work can awaken a spirit of respect and appreciation.

But you cannot just command teamwork into being. Team members' individual chances of success must be optimized and their interdependencies balanced. Although this makes teamwork subject to careful planning, there is no certainty that these elements will jell.

In our case, we found that the improvements we implemented didn't take any heroes, but they did produce one: the team. Having watched one team member change during our project from someone who argued red-faced against all comments about his work into someone who begged joyfully for critiques, I became a zealous convert to the teamwork cause.



Pictures, not procedures

I often mention the importance of building a shared vision through pictures. Common examples include a sketch of the system architecture model or an organizational chart. You cannot overestimate the importance of these diagrams; you should put them all over the place. Drawing pictures instead of writing pages and pages of procedures seems more efficient for two reasons: not only is a picture worth a thousand words, it can also be absorbed a lot faster. Who these days has time to read 1,000 words or even 500? Then too, words are ambiguous: one word can conjure a thousand *different* pictures in readers' or listeners' minds. Charts and sketches help keep everyone on the same page.

The challenge lies in picturing the rather abstract concept of a software process. We used an old trick here. Instead of drawing the sequential and parallel activities of the incremental development process in a linear way, we pictured them on a circle. It wasn't hard, because the process depicted was a small life cycle. So we used an analog watch metaphor, with 1 p.m. the requirement specification, 3 p.m. the design, and so on.

There isn't anything revolutionary about this—it's the essential idea of any iterative model. But when I saw one team drawing these watches in their e-mail to communicate the project's state, I realized the strength of this tiny trick.

Adizes' four insights

Observing people work from within a team provides insight into many psychological mechanisms. People don't always deliver the task's intended output. Sometimes, they simply don't do their job very well. Why? The typical answer is a mismatch of task and technical competence, but this explanation isn't always reliable.

To best match our team members' skills with their assigned tasks, we used the trick of looking at our people in terms of Ichak Adizes' psychological competence model.² This model defines four types of characters.

- ◆ *The Entrepreneur*. Good at generating ideas, this type is a visionary, an opportunist, someone who welcomes change and motivates others.

- ◆ *The Administrator*. Good at structuring and coordinating, this type excels at planning and likes order.

- ◆ *The Producer*. Good at completing a job, this type likes achieving goals.

- ◆ *The Integrator*. Good at cooperating, this type is a nurturer who thrives in a relaxed atmosphere in which social aspects are balanced against the demands of the job.

We realized that this classification provided the key to understanding a person's motivation and that the sure way to have a job done ineffectively was to place the person in the wrong role. For example, casting a pure entre-

preneur or producer in an administration job would be unwise. Fortunately, most people can play more than one role, often performing optimally in two of the four types.

For reviews, we found that integrators proved indispensable, whereas producers were hopeless. Planting faults (bebugging³), however, changed the producers' motivation completely. Besides providing a statistical measure of the review meeting's effectiveness, searching for planted faults motivated producers with a concrete goal. Regarding good testers, I am still considering which profile would best fill that role.

To further acceptance of matching roles with profiles, we distributed a small Adizes self-test together with a mapping of profiles to our process. We then suggested that participants be conscious about roles and profiles when doing their team planning.

Further, we used the model for risk analysis, telling the project leader, for example, "Sir, you need to replace these two producers with an administrator and an integrator or the design will be incoherent and the reviews ineffective." The project leader would reply that this was all very interesting, but we could tell he thought we were nuts. However, two months later the predicted problems became evident, vindicating our assessment.

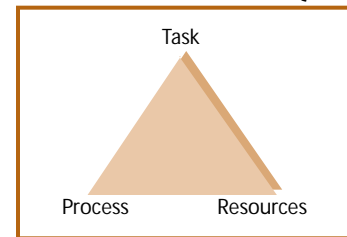


Figure 1. The TPR model, a software project triangle comprised of tasks, processes, and resources.

The software project Trinity

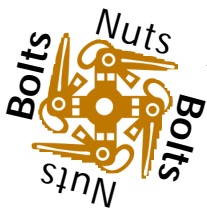
You can look at a project from three essential points of view: the tasks, the processes, and the resources. I call this the TPR model, as shown in Figure 1, and find it very helpful in framing the insights to a project.

The task angle focuses on the input and output of the project activities. The final output is obviously the product. The process angle looks into the transformation from input to output, including the roles involved. The resource angle deals with the people playing these roles.

These three points of view, taken together, provide a more holistic view of the project. They let you look for patterns and set up hypotheses.

Consider the reviews I described earlier. The review's traditional purpose is to remove defects. But that is merely a task-focused point of view. Reviews also strengthen process coordination and spread competence among the resources.

One SPI trick is to shift from the academic concept of focusing on processes alone to a more pragmatic view of matching tasks and resources to a process⁴. For example, if you want your project to benefit from advanced planning you appoint a project manager rather than just define procedures that promote planning.



Alternatively, you could demand that a certain output, such as a documented project plan, be produced.

Did the tricks I've described here lead to a successful SPI implementation? This is like asking the color of a sound. Our project certainly wasn't a failure: We enjoyed working on it, satisfied our customers, met our deadlines, and delivered a product of measurably high quality. We achieved all this not with a magic bullet, but with just a few bottom-up tricks. ❖

REFERENCES

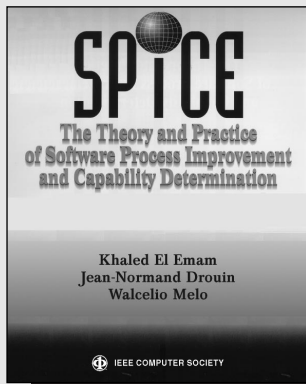
1. M. Bush, "Misunderstanding the SEPG's Role in the CMM: Truth and Consequences," *Proc. 10th Int'l Software Quality Week*, Vol. 2, Section 7, Software Research Inst., San Francisco, 1997.
2. I. Adizes, "How to Solve the Mismanagement Crisis," MDOR Inst., Los Angeles, 1979.
3. D.P. Freedman and G.M. Weinberg, *Handbook of Walkthroughs, Inspections, and Technical Reviews*, 3rd ed., Little, Brown and Company, 1982.
4. J. Bach, "Enough About Process: What We Need are Heroes," *IEEE Software*, Mar. 1995, pp. 96-98.

About the Author



Allan Baktoft Jakobsen works for the Danish company Delta Software Engineering as a consultant in best software practices. His work in the software industry has spanned maintenance, design, coding, test, and quality assurance. Jakobsen received a BS in mathematics and physics and an MS in mathematics from Copenhagen University.

Address questions about this article to Jakobsen at Delta Software Engineering, Venlighedsvej 4, DK-2970, Hoersholm, Denmark; ABJ@delta.dk.



SPICE

The Theory and Practice of Software Process Improvement and Capability Determination

edited by Khaled El Emam, Jean-Normand Drouin, and Walcelio Melo

The SPICE (Software Process Improvement and Capability dEtermination) Project is a joint effort by the ISO and IEC to create an international standard for software process assessment. This book covers both the theory of SPICE and its practical applications, including the lessons learned from the SPICE trials. It includes a valuable automated tool on CD-ROM to help you apply the concepts presented in the book.

The text shows the evolution of the most recent developments in the SPICE project. It documents the major products and the empirical evaluations that have been conducted thus far. The book is jointly written by the key experts involved in the SPICE project. The theory chapters describe the rationale behind the architecture and interpret the contents of the V1.0 and V2.0 document set.

Contents: Introduction to SPICE • The SPICE Practices • Process Assessment Using SPICE • Developing, Selecting, and Using Assessment Instruments • Process Improvement Using SPICE • Process Capability Determination Using SPICE • Training and Quality of Assessors • Introduction to the SPICE Trials • Empirical Evaluation of SPICE • Assessment Ratings from Trials • Analysis of Observations and Problem Reports • Using SPICE as a Framework for Software Engineering Education - A Case Study • The Future of the SPICE Trials

450 pages. 7" x 10" Hardcover. November 1997. ISBN 0-8186-7798-8.
Catalog # BP07798 — \$48.00 Members / \$58.00 List



Go to the
Online
Bookstore
<http://computer.org>
and order using
the online shop-
ping cart
and the secure
order form

IEEE Computer Society
10662 Los Vaqueros Circle
Los Alamitos, CA 90720-1314
Toll-Free: +1.800.CS.BOOKS
Phone: +1.714.821.8380