Microdynamics of Process Evolution

James Bach, SmartPatents

Although punditry has its rewards, I've found that practical learning isn't really one of them. As a full-time consultant, speaker, and writer, my talk-to-walk ratio as of late last year was getting uncomfortable. So in October, I took a very hands-on position as the manager of quality assurance, software configuration management, and technical support. Once again I am immersed in the day-to-day details of running software development projects.

SmartPatents develops software for managing intellectual assets. We're a typical technology startup in most ways—a bunch of energetic and committed people without a bunch of defined development processes—which makes us a good laboratory for software process evolution.

BIG OUTCOMES FROM SMALL EVENTS

After three years as chief scientist at a company that mainly did short testing assignments, there were a lot of little things I'd forgotten about working day to day on a full-blown software project. Most of all, I'd forgotten just how many little things influence real projects and just how influential they can be. I'm talking about things like how meetings are run, what e-mail is sent, how documents are written and distributed, who talks to whom, who cares about what, where people sit, and all manner of details that are too specific—or too messy—to be mentioned in most textbooks. But each little detail can trigger major differences in how events turn out. When I'm paying attention, I notice such details in just about every working hour.

If process evolution is foiled on the micro scale, it will be foiled utterly.

Chance conversations

In October my team identified the need for several expensive servers for use as testing platforms, but word came down that money was too tight for us to afford such a large and unexpected capital expense, so I decided we'd make do with the equipment we already had. But soon afterward, a fellow manager, Adam, identified the same need and mentioned it to me in the hallway. We chatted a while, a few other engineers joined in, then our vice president of software engineering, Luke Hohmann, walked by and we bent his ear too.

We achieved a consensus about our hardware needs right there in the hallway, and that same evening Adam wrote such a compelling justification that a week later we took delivery of two new top-flight servers. The servers allowed us to perform much-needed performance testing that in turn alerted us to much-needed changes in our product architecture. The servers made it possible for us to handle more users at one time, which allowed us to hold a company-wide "bug bash" testing event, which in turn led to the rapid evolution of several reusable system test scripts.

Please note: This is not a story about the importance of equipment for testers. We all know equipment is important. It's an illustration of how a trade-off decision can have far-reaching impacts, and how that decision can be influenced by a factor as simple as a chance meeting in a hallway. By their nature, trade-off decisions are particularly sensitive to small effects, and software projects are rife with trade-offs.

Casual suggestions

Another example of a small thing that eventually became important was a suggestion by one of our developers about how we could reduce the time it takes to do formal builds. The suggestion, offered casually during a project meeting, was to replace our one slow build machine with two fast ones that would operate in parallel. Luke picked up on the idea, amplified it, and it was implemented within 24 hours.

This small change shrunk our build process from nearly eight hours to a little more than two hours. From that point on, we were able to schedule formal builds during the day, instead of doing them exclusively overnight. Since we base our testing and release decisions strictly on formal builds, same day turnaround means we can fix and test all in one day, if necessary.

MACRO PLANS FROM MICRO DESICIONS

Macro plans and processes depend on micro decisions, methods, ideas, and individuals. This concept has important implications for people who want to drive meaningful process evolution in software projects. One implication is that it's easy to look back on an important project failure and not remember the critical moments and details that led to that failure. It's also easy to look back on success and not see all the ways things could have gone wrong. These are good reasons to look upon the anecdotes and conclusions of pundits like me with a skeptical eye.

Still another implication is worth special emphasis: *Small actions and corrections can make big successes possible.* The microdynamics of software projects strongly influence their outcomes.

Process standards encourage acceptance of some static benchmark defined by some unidentified, unavailable, and unaccountable process maven.

Opportunistic process evolution

Naturally, process improvement takes people, skill, motivation, equipment, time, and coordination. At the same time, there's always a lot going on. For example, we've shipped three versions of our product since I joined the company a few months ago. We can't just stop everything to retool and retrain, not even for a single day. So we feel obliged to inch toward those goals bit by bit. The hard questions are: Which bit? When? How?

This is where I part from mainstream software process literature. I find it useful to drive all process evolution with an opportunistic, problem-solving approach. That means I look for undesirable or otherwise painful outcomes, then look for practical and affordable opportunities to improve those outcomes.

One of the keys to this approach is to avoid separating out an improvement activity as if it were a project unto itself. Whenever you treat process evolution as a project, you have to bear the grumbling and passive sabotage of practical people who want to get "real work" done. Although it is tempting to impose standalone quality initiatives and slogans about Total Quality This or That, doing so creates more stress in the system.

Treating quality assurance or improvement efforts as special and different activities risks severing them from their context. Focus then shifts to process for its own sake—a phenomenon called *goal displacement*, where people are more likely to fall into the trap of thinking of practices as inherently good or bad, rather than thinking of them as good or bad relative to a particular situation.

The basics

If you weave process evolution into a development project, it can be presented and perceived in a more motivating context: the importance of making this project succeed or making this product better. By weaving I mean focusing on an actual undesirable situation that is occurring in a project—say, schedule slippage—and doing something to improve it.

Here are the seven basic steps of opportunistic process evolution:

1. Notice problems in products or outcomes.
2. Choose an important or chronic problem and look for a way to solve it, in whole or in part.
3. Conceive of a new, borrowed, or modified process that could solve the problem at an acceptable cost and in an acceptable time frame.
4. Try the new process on a real project.
5. Adjust the process in light of experience and in light of the new problems that were created by the process.
6. If the solution is worth perpetuating, look for a way to make it more efficient or durable by adding infrastructure (such as documentation, training, or tool support). Otherwise, try to understand why it doesn't seem to work.
7. Return to step 1.

Success with this approach requires that we be able to detect problems and gain a consensus about which problems are important. This approach is substantially different from the approach advocated by process standards like ISO-9000 or the CMM, which mandate that certain processes and institutions be put in place, regardless of the actual problems faced by the companies and projects.

To illustrate the contrast between process standards (like ISO-9000 and the CMM) and process evolution, consider one of the activities mandated by the CMM at Level 2: "The sizes of the software work products (or sizes of the changes to software work products) are tracked, and corrective actions are taken as necessary." The problem I have with this is that nowhere in the CMM can I find where it tells me why this is a useful activity. I can imagine why it might be important to somebody, but I don't see how it intersects with any of my objectives.

I'm sure there are those of you who will take issue with my indifference to size metrics. But whether size matters is not the point. The point is that I don't understand why size matters. So how can I be effective at convincing anyone that it does matter? If I were to attempt to implement this practice, how would I know if I were doing a good job?

Not knowing whether I'm doing a good job is one reason I feel that process standards are so dangerous. Rather than encouraging critical thinking and supporting the education of the software project managers and personnel who are on the scene, process standards encourage acceptance of some static benchmark defined by some unidentified, unavailable, and unaccountable process maven.

That leaves me with a sticky problem. Does opportunistic evolution mean I can't borrow the solutions from someone else? Certainly, I have been hired in large part because I think I know the solutions to certain problems. I do make use of textbooks, standards, and preconceived ideas. I collect process fragments and ideas of all kinds. How can I reconcile opportunistic process evolution with the fact that I also have preconceived ideas of how to get things done?

I reconcile process evolution with preconceived ideas by abstracting each process solution of mine back to the problems it's supposed to solve, or objectives that it serves, then looking to see whether it is an important problem or objective in my current situation. If I have a practice in mind and catch myself thinking that we should adopt it, I ask myself questions like these:

1. What objectives are served by this practice? What pain will it resolve?
2. Are those important objectives? Important to whom?
3. In what way are those objectives already served by some other means?
4. What would a highly successful implementation of this practice be worth?
5. How much energy will be required to make it happen? Is there a simpler, cheaper solution?
6. What are the prerequisites for adopting this practice (like special training, methods, or tools)?
7. How will this practice disturb or interact with existing practices or processes?
8. What problems or risks will this practice create?
9. How will we know that the practice is helping? How will we assure its quality?
10. If it isn't helping, what will we do then?
11. How much of this practice will be enough, or too much? Can a little of it make a big impact?
12. What alternatives are there to this practice? What if we do nothing?
13. What simple, achievable, self-contained step can be taken toward the new practice?

I don't have to answer all these questions, or any of them, but I've found that the quality of my work is proportional to how well I can. I suppose these questions are common sense, but for each of them, you have to consider answers that are not merely apparent from a distance or from an ideal model of projects.

Pay attention as well on the scale of hours, cubicles, people, and episodes. In complex cognitive processes like software development, if process evolution is foiled on the micro scale, it will be foiled utterly. ❖