CHAPTER SEVENTEEN

Memories of TRW's Software Productivity Project

A Beautiful Team, Challenged to Change the Culture*

Barry Boehm and Maria H. Penedo

Background on the Software Productivity Project

In the early 1980s, I (Maria) was looking for a job after finishing my doctorate in computer science at UCLA. I was full of enthusiasm for the field I was in and a little concerned about the kinds of jobs I would find. I wanted a place where I could follow my passion and make contributions. I had met Barry at one of the conferences where I presented some of my research results, and thus went to interview with him at TRW. During the interview, Barry described the environment around the company and told me about this new project he was starting whose objective was to revolutionize the way the company developed software.

Barry told me that he had conducted a software productivity study [1] in the company, which performed an economic analysis to determine whether a significant level of investment into software productivity aids would be justified. The factors that led to the analysis

* Editors' note: if you've worked on a software team in the past 20 years, you have been influenced by Barry Boehm. He was one of the first people to take a systematic approach to estimating and planning software projects. And many people (including us) believe that his pioneering Spiral Model is the direct predecessor to the modern idea of iterative development.

were driven by an overall corporate focus on industry and international competitiveness, but also included increased demand for software, limited supply of software engineers, rising software engineer support expectations, and reduced hardware costs—do they all sound familiar even today? This study led to the project that he wanted me to work on, the Software Productivity Project (SPP) [2,3]. He told me it would not be easy, since we would be trying to *change the culture*. Well, it seemed the perfect challenge for my first job in industry, so I accepted the offer and went to work on the project, together with a fantastic team of TRW veterans and newbies. We were going to attempt to change the current ways of developing software by providing a software development support environment for the aerospace part of the company.

Before it was acquired by Northrop Grumman, TRW was a conglomerate which included a large auto parts company (Thompson Products in Cleveland) and a large aerospace company (Ramo-Wooldridge in Los Angeles). It included a number of software-intensive product lines, including TRW Credit Data, and divisions producing point-of-sale systems, telecom switches, and industrial process control systems. In the mid-1980s, TRW was ranked #2 in world software sales in the annual Datamation 100 lists, well behind IBM.

In the 1960s, TRW's aerospace sector pioneered in going from the code-and-fix approach to software development to the requirements-driven waterfall model, as described in Winston Royce's classic 1970 paper [4]. In the 1970s, TRW developed a set of waterfall-oriented software development policies, standards, review procedures, training courses, and requirements-driven software tools. These were a good match to TRW's engineering, science, and real-time control systems of the time, and the culture became strongly waterfall-oriented. By the late 1970s, though, TRW's applications became much more people-interactive, and the waterfall model didn't work well with requirements that were emerging as the project progressed.

At that time, the environment for projects included managers coordinating the people and activities, secretaries who typed the project's documents, meetings to plan and produce the project's activities and tasks, and system engineers and developers producing the designs and the systems. Only developers used software development tools and they had to work in batch processing mode or go to special bays with limited numbers of terminals for interactive development, using mainframe computers. Well, in the SPP we developed a new work environment where all members of the project (managers, system engineers, software developers, secretaries, and controllers) had individual offices with workstations, and communicated electronically via a local area network (LAN). Its architecture was based on the Unix operating system and it included commercial off-the-shelf (COTS) and locally developed tools in support of the full life cycle (e.g., requirements definition, traceability, design and development, forms management, and office automation tools). For this new project, we had to cope with significant technical and stakeholder-conflict risks, and with emergent requirements that could not be specified in advance; thus we had a great opportunity to apply an early version of the risk-driven, concurrently engineered spiral model developed by Barry [5]. Those changes created culture-clash challenges, especially for the waterfall-oriented veterans and interactive-oriented newbies on the SPP

itself. We can still recall heated meetings where TRW veterans would say things like, "But the policies don't allow you to prototype this early! Prototyping is coding before you have passed your critical design review!"

The project was very successful (with significant productivity gains—a factor of 2 or more, depending on reuse), but it was not easy to institutionalize the changes and convince the personnel. We also learned that productivity gains require an integrated program of initiatives in several areas and an ongoing and sustained effort. Even though this project happened long ago, the stories we tell in this chapter describe points that are still valid today, in spite of all the advancement in tools and technologies, both in planning and in executing productivity activities. *Changing cultures is difficult, and it is even harder to keep up with the fast pace of technology in large organizations.*

Making the Project a Reality

Getting Started: Being Ready with Options When Management Calls

Scene: Bob Williams' office, late 1979. Bob is the vice president/general manager of the 2,000-person Software and Information Systems Division, one of six divisions in the TRW Defense and Space Systems Group (DSSG). Barry is his chief engineer and advanced technology business area manager.

Bob: I've just come back from a DSSG General Managers off-site about improving productivity. Corporate in Cleveland is making a big push to get the auto parts division to be more competitive with the Japanese, and wants everybody in TRW to focus on improving their productivity. It looks like the company will put up money for productivity initiatives if there's a good business case for them. I think it's worth a try. Do you think you can put something together for us?

Barry: Definitely. This fits with a lot of improvements we've talked about but haven't found funding for. Our Constructive Cost Estimation Model (COCOMO) provides us with a good framework for a business case. It shows how much our productivity goes up or down as we change some of the cost drivers like tool support, turnaround time, reusing components, and people factors. This last driver would fit with your ideas about multiple career paths for our people. We could probably use some of our local area network technology to get everybody interactively working and communicating. And we could probably get added support from some of the Defense Department's Ada initiatives. Are they looking for a full-up proposal?

Bob: Well, if we were proposing to the government, that's what we would do. But since this is an internal company initiative, the sponsors want a clearer idea of their options, before they commit to spend a lot of money. So we have a couple of months to put a white paper together. Why don't you do a part-time study with Ray Wolverton and a couple of the Ada guys and put a draft together? And let's get everybody involved by doing a survey of what people think would best help them improve their productivity.

Barry: Great. We'll get right on it and give you a progress report in a couple of weeks.

Evaluating and Selecting Options: Applying the Spiral Model

The project's incremental versus total upfront commitment was the first opportunity to fully apply the spiral model of software development [4] to incrementally explore options, refine scope, and obtain higher levels of management and user buy-in at the end of each spiral cycle. After the first white-paper cycle ended with approval to proceed, the second cycle of the spiral involved visits to advanced technology centers such as Xerox PARC and IBM Santa Teresa, LAN and workstation architecture and market analysis, more detailed manager and developer needs surveys, demos, and prototypes. Three operational concept options were prepared at expenditure levels of $2K, $10K, and $50K per person, resulting in selection of the $10K/person alternative, although the Xerox PARC workstations were tempting.

Scene: Bob Williams' office, mid-1980.

Bob: Good news! We've been selected to develop our proposed Software Productivity System. You guys did a great job in the Round 1 white paper, which got us support to develop the operational concept options. With the help of the productivity data from IBM and AT&T's initiatives, they bought into our $10K-per-person option with the local area network, the lower-cost version of the IBM private offices, the Unix-based support system for not just programmers but everybody on the project, and the complementary management and career path initiatives. So we're funded at $1 million per year to do this, but we'll need to pass a review based on a set of prototypes, specifications, and plans, and find an early-adopter project to work with you on building what they need. I think I can find a good pilot project. Do you have anything further you need to get started?

Barry: That's great! I'll need some help in working with the facilities people on reconfiguring and wiring the private office complex, and on letting the TRW LAN people know that we'll have to do a competitive analysis of their product's maturity and performance. Also, we'll need to hire some top technical people with extensive Unix experience. There are a couple of people at UCLA and UC Santa Barbara that I think will be very interested.

Bob: Fine. You can put together a presentation on this for the next staff meeting, and I'll identify people to work with you on those.

Getting Started: A Balanced Team, a Committed Pilot Project, and the Niceties of Unix

The productivity project was fortunate that its early-adopter partner project was a portion of a very large real-time application with an open-minded manager and many performers who, after some resistance, appreciated the ability to have dedicated interactive workstations and electronic communication. And the Software Productivity Project team had a good balance of experienced TRW software developers and new-hire Unix environment experts. The Unix key experts from UCSB (Art Pyster) and UCLA (Maria Penedo) were very good in coming up with creative and new ideas, engaging the team members, and showing how Unix capabilities could improve automation, support rapid prototyping/development for both tools and target software, and shortcut some of the frequent diffi-

culties projects would encounter. The pipe and filter and other features of Unix made it possible to respond regularly to those kinds of customer requests; we could put together simple solutions to problems and respond quickly to customer requests because of the flexibility of the Unix environment.

Scene: An early Unix demo to the partner project people, mid-1981.

Skeptical old-timer: All of that is very interesting, but can it do anything useful for the project, like create a specialized telephone directory?

Art: Sure. We'll just do a "grep" on the names in the project roster and the names in the company phonebook, "pipe" it to a new file…and there it is.

Project Stories

The Team

Yes, we formed a team, a beautiful team, with 20–25 performers. Some were experienced TRW veterans, but most members of the team were in their 20s (the GEN-Ys of today), a few fresh out of college including undergraduates, some with master's and PhD degrees, a good mix of male and female personnel, a few non-natives (from South America, Asia, and the Middle East), and a lot of enthusiasm. It did not take a long time for the team to gel, with many having lunch together most days (do people still do that these days?), others even becoming good social friends. And of course, we also had your typical "nerds" who produced tons and did not like socializing much (we will tell you more about that later). Keep in mind that most of the members were totally comfortable with the new technology since they were coming from universities that were pioneers in the Internet (like UCLA). The target community, however, represented much of the standard company personnel, used to the "old" ways.

The Challenge

Our challenge was to convince the typical project personnel to adopt a new style of development and collaboration as a team and use the new technologies we were bringing. The SPP was aiming at changing the culture, establishing LANs and bringing all members of the project to communicate via email and to use automation for the development and non-development activities. (Remember: that was in the '80s… On the other hand, things haven't changed; we can observe the same issues with established personnel in large corporations which hesitate at the introduction of the latest technologies like social networking, Wikis, and virtual worlds.) We were matched up with another project that was to use the new tools/processes that our team generated and we have many interesting stories about this matchup.

Educating the Boss

We had a wonderful boss, very friendly and open-minded—after all, he took the challenge to manage this "revolutionary" project. However, he had been in the company for more

than 20 years and was very accustomed to the culture. We, the young, set out to change his habits. He was enthusiastic about most of the proposed tools to automate the life-cycle process, but we noticed that he never used email (none of the managers did at that time). So one of us played a trick on him; she scheduled a meeting with him and, upon arriving in his office, she promptly said she was going to teach him how to use email. You should have seen his face; the look of astonishment. He was used to telling his staff what to do, not the other way around. To make the story short, he could not refuse, and after learning, he really became a part of the team by participating in the online discussions and conversations, understanding the issues, and sharing the news. By avoiding the change, he was missing on much of the project chemistry. Sometime later he confessed that he "couldn't type" and therefore was embarrassed of showing his shortcoming. But this staff member and her boss became very good co-workers and friends, as the project progressed due to his openness to change. Those were key ingredients toward the success of the project.

Can We Have a Private Office?

The status quo of projects was to have offices for managers and cubicles for project personnel; developers used a development bay where a pool of shared computer terminals were located. Among the "new" ideas being implemented was to give each project person a private office independent of his or her status, and access to computers via individually assigned terminals connected to a LAN. Good ideas. The LAN idea caught on like fire; they had access to the computers from their offices at any time. Unfortunately, economics didn't allow the "individual office for all" idea, not then or now. And we tried; at that time we knew that when the crunch for spaces hit, that wouldn't hold. Thus, the proposal was made to make the individual offices just big enough according to company policies so as not to allow more than one person. That worked for the project, and folks really enjoyed their privacy while it lasted. Unfortunately, that was one of the change proposals that did not get migrated into the rest of the organization; we were not able to prove the value proposition or to fight the economic pressures.

As a side comment, there was a lot of jealousy from those outside the project because we had private offices, terminals on our desks, and so on. We had to find ways to not aggravate that jealousy by pointing out that we were the "guinea pigs" for something that might not work, and that if we were successful, they would eventually get what we had.

Choice in Technologies: The Importance of Trade Studies

The selection of technologies didn't run that smoothly all the time. And the choice of a commercial versus in-house bus interface unit for our LAN was an issue. There was an in-house product that was immature but being pushed by upper management, which is quite understandable since it was our own product. A careful trade and testing experiment was carried out, but there was a need for plenty of political outreach to convince management that there would be less risk if we went the commercial route. Does that sound familiar?

Converting the "Guinea Pig" Project

As we mentioned before, a project was assigned to use the new environment/tools being developed. The LAN idea was very welcomed, but the operating system (we were introducing Unix into a DEC VMS-oriented culture) and some of the new tools were not welcomed by all. To make it harder, the partner pilot project was using the VMS operating system, which already came with a few good automated tools. Also, several personnel in the assigned project complained that Unix on the VAX ran much slower than did the VMS operating system. That was true, but they missed the fact that that was a development environment and that overall productivity was improved because the people were the more valuable resource at that point (they still had the mindset that machine time was more important than people time). Over the course of the project, some became converts, some never did.

What was interesting, however, was to observe how sometimes something very small can have a huge effect on user acceptance. We have two examples to tell, both involving showing the value added in the new automation. The first one was about a secretary who typed most of the documents for the project and dealt with quite a few of the project personnel; many of the documents she typed had many equations for which she had to leave space to type, even if she used an automated word processor. That was both cumbersome and slow for her and her customers. Once she found out that Unix had the "eqn" tool which automated equations, she became a convert and was able to convert many more in her circle of work, including managers and system engineers, who were by then also being converted to use word processors. The other example was a savvy administrative assistant/data manager, who figured out she could automate some of their forms needs using Unix "scripts" as well as providing configuration management with the new tools; her boss was so impressed with the results that he mandated all his staff to use the system/tools. She used her creativity to solve a problem and had an open-minded boss who also dared to mandate (sometimes that is the only way to get some people to change).

Standardization

Technology adoption has many facets, but we found that, even if the technology is good, sometimes adoption happens only by standardization or mandate. It happened for this project after the tools proved themselves, and it still happens today within the organization. Examples include standardizing on PCs, using Microsoft products, and other examples including SharePoint, eRoom, Livelink, and so on. Management made those decisions and the users just went along with it. On the other hand, where possible, standardizing on interfaces is better, particularly for custom software houses like TRW that have to deliver software and support tools that are compatible with different customers' environments.

Users Should Be Part of the Team

Technology acceptance is easier if users are invested in its success. If they are involved from the very beginning and participate in the requirements definition, analysis, and design processes, or if they participate in the review process during its development, they

will feel ownership and are more likely to defend and less likely to criticize and/or reject the technology. This was a lesson we learned the hard way. Some team members expected to spend their time developing state-of-the-art research tools, and were somewhat disappointed when users said they wanted simple (for Unix) things like putting change bars in the margin to indicate where the next version of a document had been changed. But they felt a lot better when they saw the positive impact that their amenities had on the project users.

Making All Inclusive

Every project has its lone wolf, who, for reasons unknown to most of the team, appears to not want to be part of the team, lacking bedside manners. We had one of those, an extremely smart and talented person who had no patience for the "normal" people. He was extremely fast and constantly surprised us with his innovation. We remember that he got the Forms Management package to speed up performance by a factor of 4 or 5 because he just dug around into Unix until he found a few obscure interface calls. All that was said to him was that the package was running too slowly for many. He grumbled about how they should appreciate what they had, walked away, and came back two days later, all smiles and with a new version that ran four or five times faster…and we didn't even know that he was working on it.

But his aloofness turned out to be very difficult for many. It bothered the team; we wanted to bring him in, we felt it would really improve team collaboration since he was one of the most knowledgeable in the group. We don't know exactly what caused the change, but one of the members of the team, a smart lady with degrees in both music and computer science and with great wit, became his friend and broke through his glass shell. We think after that, he started trusting and actually enjoying some members of the team. Life in the project became so much better after that. The lesson here is that if people care and take their time, ways can be found to include everyone.

Training Managers, Not Your Usual Student

The project started a training program about the operating systems and the many tools. We then noticed that the managers were not coming to the class. Was it lack of time or hesitancy to show their lack of knowledge of the capabilities? We found out that the latter was true for many. As risk mitigation, we started separate classes for managers (which were not that different in content, maybe just a little more high-level) which made them feel more comfortable being within their peer group; once they were more familiar with the technology, they were more prone to use, recommend, or mandate it.

Tasting Our Own Cooking

A great lesson learned was to make the SPP use the tools that it was developing. That wasn't done at first, which led to the inability to understand some of the users' needs and which hurt the acceptance process. Once the project started to use the tools, such as the requirement traceability capability, a better understanding of the user community

occurred and that led to many improvements to the tools themselves, which then led to better acceptance.

Is Email a Boost or a Hindrance?

What would we do without email these days? (Actually, the email of yesterday may equate to the Instant Messenger or social networks of today.) We institutionalized email across the projects with the new environment. But of course, that brought about issues, which still apply to date. Email has many benefits: it enables people to communicate or broadcast messages instantaneously to multiple recipients across a great distance, it allows one to leave messages when the recipient is not at his or her desk, and so on. However, it can also incur negative productivity in that, instead of picking up a phone to talk to another person to solve a problem or to discuss some ideas, one defaults automatically to writing an email message. That can waste time on both sides, taking longer for resolution of issues when a short phone call would satisfy. There were many frustrating moments of people not addressing issues posed in an email because they either didn't read their email carefully or they misunderstood the email.

Also, the problem of sending cryptic email can create hurt feelings and misunderstandings. Due to the non-interactive and non-personal nature of email, the "quality" of communication often suffers. Misunderstandings develop and sometimes are not clarified or remedied until significant cost in time and labor is incurred. A side effect of the SPP environment occurred when some people stopped stepping out of their private offices to talk to each other. They sent cryptic email messages rather than popping their head into their neighbor's office to ask a question or discuss an issue. That led to some team issues that had to be solved by an all hands where management encouraged the team to "talk" to each other rather than defaulting to email as a means of communication.

Also, email is not efficient for assigning action items or important items. One should never assume that the email was read correctly on the other side, if read at all. And sometimes over-reliance on email multiplies the inefficiencies—for example, if "Reply All" is used indiscriminately or accidentally.

Becoming Word Processors

The same negative productivity applied to tools like word processors (e.g., LaTeX and others). Whereas being able to type their own documents made this task much faster and great for maintenance purposes, users complained that they ended up spending too much time on the aesthetics (getting the formatting just right), rather than the content of the documents. Because the document generation switched to being prepared by the engineers themselves (except for managers who had secretaries), the cost per hour was ultimately higher and the content suffered because more time was spent on debugging the document format. Of course, the advent of WYSIWYG word processors solved that problem; well, almost.

The Difficulties of Innovation, Timing, and Commercialization

Led by the projects' needs, a forms management system was created which included many of the fourth/fifth-generation DBMS capabilities of today; those did not exist at that time. It was an excellent system which could have been commercialized, had our company been in the commercial market. That did not happen since we were (and are) a large aerospace company who works mostly on contracts. Keep in mind, as mentioned before, that the team had many young, entrepreneurial, and very creative folks who thrived in the environment being brought about by this innovative project. As a result of the lack of interest in commercialization, a few of those folks left the company, some to try new endeavors, others to work on their own to build tools that could be commercialized.

The Iron Law of Software Maintenance

The Iron Law of Software Maintenance says that for every dollar you put into development, you will have to spend $2 to keep the product viable over its life cycle. The Software Productivity System began as a free service to projects, and it became too hard to switch to having projects pay for its maintenance. Thus, as time went by, the project had fewer and fewer resources to add new features. For example, the forms management system was the world's best system of its kind in 1982; by 1986, it was still the world's best 1982 system of its kind, but had become eclipsed by commercial forms management systems with broader financial and user bases.

Champions

Throughout this and many other projects, we identified the need for "champions," that is, people who will adopt and are passionate about the technology, both on the development and the user sides. We all know that sometimes passion can move mountains. And many times, without such champions, the momentum tends to die. However, the life of a "champion" is not always easy; it depends on the need for the technology, the economics, and the goals of the management in charge. Champions are at the heart of change; they need to be nurtured and rewarded.

The Challenges of the Rapid Change of Technology

The reason the project started in the first place was to bring new automated tools into the organization for the purpose of enhancing software engineering productivity. However, we found out that the pace of technology changed very rapidly and had to create means to keep looking toward the future in order to fulfill the goals of the project, to continuously bring in and deploy new technologies (that was before IT took such central places in enterprises). It wasn't (and still isn't) trivial. In parallel with the program, a future-looking activity evolved, and from workstations we moved into bitmap displays, computers at the desktop, and so forth. As this project moved on into other improvement activities, we found out that in-house technology or tools development within the organization was generally short-lived and invariably overtaken by commercial products. Each one of the efforts started with high hopes, made some progress, and caused changes and improve-

ments, but organizations need to be constantly reinventing themselves. Many of those efforts served an excellent purpose in introducing the concepts into the consciousness of the user community and in illustrating the need and the benefits of the technologies, but many of them ultimately lost momentum due to the Iron Law of Software Maintenance and the relatively small user base across which to amortize changes to the custom in-house tools. That is the reason why, in today's world and age, most tools in use come from the commercial world, and those by themselves carry their own challenges of robustness, cost, adaptation to user needs, and evolution.

Learning and Assimilating Changing User Behavior

Both the productivity project manager and the pilot user project manager commented on the ability of electronic communication to flatten traditional management hierarchies. An example comment was, "On my previous project, I was never sure what was happening two levels down, and I always felt that I was too late in responding to project problems. Now that I see a lot of the email traffic, I have more advance warning of problems coming up and a clearer picture of how each part of the project is doing." Also, the fact that all of the artifacts were electronically analyzable meant that we could better identify and fix bottlenecks in such processes as change management and defect closure. A key advantage of the private offices coupled with passionate people was the ability to achieve the state of productive flow emphasized in Tom DeMarco and Tim Lister's *Peopleware* [6]. As the project went along, a sort of door code emerged. A closed door meant "I need to concentrate," and an open door meant "Come in and talk." This led to examples such as the following:

Scene: The productivity project office suite, 4:00 p.m.

Steve emerges, saying, "I'm hungry. Anyone ready for lunch?"

Sue responds, "Steve, it's four in the afternoon. You've been in there programming all day."

Steve: "Well, I guess it's true that time passes quickly when you're having fun."

Conclusion

The productivity project was fortunate to have a number of the features that distinguish successful from unsuccessful software projects: top management support, capable and enthusiastic team members, realistic budgets and schedules, concurrent requirements and solution development, and iterative development. However, many projects have had all of those factors, but have fallen short of having a *beautiful team* experience. In comparing the productivity project with some of these other projects, we would say that some of the key *beautiful team* enablers were:

- Identifying and involving all of the success-critical stakeholders
- A lot of work upfront on listening, exploring, and team building

- Developing a shared vision for the product and its results
- Identifying a manager with an open mind, good hearing, and team building and management skills
- Encouraging creative ideas from outside and within
- Paying attention and addressing the team's needs
- Respectfully redeploying incompatible performers
- Negotiating win-win resolutions of stakeholder conflicts
- Carefully monitoring progress and proactively addressing win-lose threats

References

1. Boehm, B., M. H. Penedo, E. D. Stuckle, et al. "A Software Development Environment for Improving Productivity." *Computer Magazine*, May 1984, pp. 30–44. Also in R. Selby (Ed.), *Software Engineering: Barry W. Boehm's Contributions to Software Development, Management, and Research*, IEEE Computer Society Press/Wiley Interscience, 2007, pp. 245–268.
2. Bitar, I., M. H. Penedo, and E. D. Stuckle. "Lessons Learned in Building the TRW Software Productivity System." *Proceedings of Spring CompCon*, San Francisco, February 1985.
3. Penedo, M. H., and E. D. Stuckle. "TRW's SEE Saga." *Proceedings of the International Workshop on Environments*, Chinon, France, in *Lecture Notes in Computer Science*, No. 467, Springer-Verlag, September 1989.
4. Royce, W. W. "Managing the Development of Large Software Systems: Concepts and Techniques." *Proceedings, WESCON*, August 1970.
5. Boehm, B. "A Spiral Model of Software Development and Enhancement." *Computer Magazine*, May 1988, pp. 61–72. Also in R. Selby (Ed.), *Software Engineering: Barry W. Boehm's Contributions to Software Development, Management, and Research*, IEEE Computer Society Press/Wiley Interscience, 2007, pp. 345–365.
6. DeMarco, T., and T. Lister. *Peopleware: Productive Projects and Teams*, Dorset House, 1987 (2nd Edition, 1999).

Acknowledgments

Our thanks to the contributors and reviewers of this chapter, including Christine Shu, Frank Belz, Art Pyster, and others who participated and made this a beautiful team. We would also like to dedicate the chapter to the memory of Don Stuckle, the productivity project manager and a truly beautiful person.