Making the Software Factory Work: Lessons from a Decade of Experience

Harvey P. Siy[1], James D. Herbsleb[1], Audris Mockus[1], Mayuram Krishnan [2], George T. Tucker [3]
[1]Bell Labs, Lucent Technologies, Naperville, IL 60563
[2] University of Michigan Business School, Ann Arbor, Michigan 48109
[3] Bell Labs/Lucent Technologies, Holmdel, NJ 07733

Abstract

At the heart of proposals to use process-oriented techniques for creating organizations that are capable of creating high-quality software at low cost is a focus on software process maturity, organizational learning to foster continuous improvement, and contractual arrangements that support an exclusive focus on software construction activities, as opposed to a broader focus on end-to-end development of an entire product.

We study an organization that was to provide fast, low cost, high quality software development services to product teams within Lucent Technologies. The vision called for an organization with a culture distinct and isolated from the rest of Lucent, characterized by a commitment to a well-defined software development process, use of state-of-the-art technology that fits into the process, and use of various forms of feedback to recognize and take advantage of opportunities for process improvement.

The organization has operated for nearly a decade now, and has evolved over the years as the basic principles have been put to the test in actual product development work. We use a rich collection of data from interviews, questionnaires, software metrics, and software process assessments to advance our knowledge of how to create and sustain an effective, medium-size process-centered software development organization.

1. Introduction

There has been much discussion in recent years of process-oriented techniques for creating organizations that are capable of creating high-quality software, very quickly and at low cost e.g., [2]. At the heart of this proposal is a focus on software process maturity, organizational learning to foster continuous improvement, and contractual arrangements that support an exclusive focus on software construction activities, as opposed to a broader focus on end-to-end development of an entire product. The Advanced Software Construction Center (ASCC) of Lucent Technologies[1] was created to provide fast, low cost, high quality software development services to product teams within Lucent. The vision called for an organization characterized by

- a commitment to following a well-defined software development process,
- use of state-of-the-art technology that fits into the process, and
- use of various forms of feedback to recognize and take advantage of opportunities for process improvement.

The Center has operated for nearly a decade now, and has evolved over the years as the basic principles have been put to the test in actual product development work. This experience has produced many lessons about what works and what does not work, about problems, workarounds, and solutions. The goal of this paper was to use a very rich collection of data from interviews, questionnaires, software metrics, and software process assessments to advance our knowledge of how to create and sustain an effective, medium-size process-centered software development organization.

2. The site

Work on the innovative "Silver Bullet" [3,4] techniques for interval reduction began in 1990, and ASCC was founded in 1991, with the goal of putting these principles into practice to achieve breakthroughs in increasing productivity and reducing cycle time. This was to be accomplished with a process-oriented "software factory" [2] that dealt only with architecture, design, code, and test, not systems engineering, requirements, or maintenance. The organization was to function as a sort of software subcontractor, with no contact with the end customer.

The first step was to define a detailed software development process, for which a number of experienced internal consultants were engaged [3]. The desire was to create a process-oriented culture, quite different from the culture prevalent in many other parts of the organization. To this end, only a few experienced developers were transferred, and the staff was filled out with new bachelor's level graduates. All staff members were given extensive training

[1] Formerly a part of AT&T.

in the new processes, and detailed on-line documentation was available.

At the time these data were collected, ASCC had a staff of about 100 people, with plans to continue growing at an increasing rate. By this measure ASCC has been very successful. But this success did not come easily, and progress was as often achieved by changing or adjusting the philosophy as it was by implementing the original intentions, as our results below indicate.

3. Principles put into action

We were primarily interested in understanding what happened when the basic principles, which underlie the efforts of many software organizations, were put into practice. For each of the basic principles, we describe how it was initially implemented, the kinds of successes and problems arising from the initial implementation, and the adjustments made in order to address problems. In some cases, the results were also supported with quantitative analyses.

The basic principles had to do both with the process orientation of the organization, and with the software factory-style business arrangements:

Process orientation:

- Task of process specification is separated from the task of process execution
- Detailed process specification created before actual development work
- Assigning responsibility and assuring accountability for process improvement

Software factory arrangements:

- Software-only contract shop business arrangements
- Implementing organizational learning for continuous improvement

4. Data collection and analysis

We combined qualitative and quantitative methods to investigate the results of applying these principles. The qualitative data allowed us to understand how process improvements were actually implemented, and investigate in some depth how the improvements, their consequences, and adjustments were perceived by staff and managers at all levels. The assessments allowed us to view the organization at several distinct points in time from the independent, external[2] viewpoint of assessors who gathered data and interpreted it according to standard models and methods. The assessment results gave us considerable insight into the strengths and weaknesses of the process, and a firm basis for understanding the improvement actions that were taken in response. Finally, we have quantitative data that we used, when possible, to test hypotheses derived from the qualitative data. In most cases, quantitative results supported conclusions tentatively reached on other bases.

4.1. Interviews

We conducted interviews with 12 individuals from the organization in order to develop a balanced and relatively complete picture of the organization's history, its improvement efforts, what had worked, and what had not. We were careful to include a variety of management levels as well as technical staff, to include individuals known to be very enthusiastic advocates of process as well as those who were more skeptical, and we included two process consultants from outside the organization who had worked closely with it for a period of years. All of the interviews were semi-structured, i.e., they were organized around a set of prepared topics. They were asked what they thought were the most significant changes in the process over the years. They were also shown a list with a dozen of the organization's major process improvement initiatives and asked about their effectiveness. The questions were open-ended and the interviewees were given the opportunity to bring any relevant information into the discussion. To ensure accuracy, all interviews were tape recorded and the recordings transcribed.

4.2. Software process assessments

We had available to us the results of four software process assessments, conducted in 1994, 1995, 1996, and 1997. These assessments were conducted by Lucent consulting group, and combined modified versions of the Software Productivity Research (SPR) CHECKPOINT™ methodology, and the Software Engineering Institute's Software Capability Maturity Model℠ (SW-CMM) [12]. Each assessment was conducted by two experienced assessors from the consulting group. Questionnaires were administered ahead of time, and two days were spent on site collecting qualitative data. For each assessment, a report was produced, showing where the organization stood on each of the CMM Key Process Areas (KPAs), as well as each of the areas in SPR's Software Development Profile. Strengths and weaknesses in each area were identified, and a set of recommendations was presented.

These assessments must be carefully interpreted. The CMM assessments of the organization, as became clear to us, may not objectively reflect what one would intuitively consider as an organization's process maturity. At the first

[2] The assessors were external in the sense that they were from another Lucent organization, and were therefore able to be more objective than an organizational self-assessment might be.

™ CHECKPOINT is a trademark of Software Productivity Research.

℠ Capability Maturity Model and CMM are service marks of Carnegie Mellon University.

assessment the CMM Level was 1, but a whopping 85% of requirements to reach Level 5 were implemented. Not surprisingly, in just two years the assessed CMM Level reached 3 with virtually the same percent of requirements to reach level 5 being satisfied. So there were only a few KPAs that varied over the organization's history. We also collected more detailed data on the nature of the process used for each release of the project. These data, discussed in the next section, allowed us to understand and investigate these effects more thoroughly.

4.3. Process attributes for each project

To reconstruct the process history of each project, we conducted a survey of process-related activities specific to that project. The questionnaire was developed from a subset of the CMM key process areas (KPAs). [12] We asked one of the consultants who conducted the process assessments to pick out certain KPAs whose answers he felt had changed over time. We included all the Level 2 KPAs except subcontract management (which did not apply to this organization). We also included process focus, intergroup coordination and peer reviews from Level 3, and defect prevention from Level 5. Unlike assessment questions which are mostly binary valued (e.g., "Did you do this or not?"), we asked the respondents to rate the frequency of usage. The assessments measured on this scale are reported to exhibit higher reliability [7]. The survey had from 2-5 questions for each KPA, in order to ask about different facets of the goals for each KPA. We selected an average of three respondents per project to answer the survey.

Many doubts arise as to the validity of any survey instrument. First is the potential ambiguity of the questions asked, because some terms may have subtly different meanings. We mitigated the effect of this potential ambiguity by conducting the survey as a highly structured interview, with one of us sitting down with each respondent to clarify any ambiguous point.

Second, there is the problem of accurate recollection, especially for projects that have terminated years ago. We lessened the effect of this potential recall problem by bringing along information about each project, and having it on hand during the survey to refresh the respondent's memory about which project we were asking about, what release it was, and who else was on the team.

Third, we cannot expect perfect accuracy in the answers to these types of questions, making it hard to analyze given small number of respondents evaluating each project. We addressed this concern by using regression methods that take into account errors in predictor variables.

With all the cautionary steps taken, analysis of the survey's validity shows several results. Most respondents perceived the process to be the unchanging across releases of the same project, giving the same rating to most process areas for all releases. The respondents' answers to related questions were consistent. We tested this in a number of ways: We tested the inter-rater reliability coefficient across the respondents within a project. A confirmatory factor analysis and hierarchical clustering of responses to KPA questions revealed that the different aspects, or questions, of the same KPA were related and loaded on a single factor. For example, all the questions in the defect prevention KPA form a single cluster.

For many projects, however, the reliability across respondents was low, i.e., respondents reported varying perceptions or memories of the project. A close examination of the responses revealed several respondents who did not appear to be giving accurate answers. Several respondents (4), all relatively new to the organization, had very little experience with software process improvement, having, for example, never experienced a software process assessment. We judged that they did not have the necessary experience with the concepts and terminology to give good judgments. We also had several others (3) who told us they had only minimal exposure to the projects or releases they were reporting on. Based on these considerations, we excluded a total of 7 respondents from the analysis.

We collected survey data on 9 projects with a total of 42 releases. (ASCC has been involved in substantially more than 9 projects, however, we were unable to get data on several projects that dated earlier than 1994 because the developers involved were no longer available.) To reduce the number of predictors, we sorted KPA questions into four broad categories, based on a cluster analysis of the results and intrinsic similarities among activities: Software Process Management, Tracking, Defect Prevention, and Planning.

We first identified the process characteristics that actually changed over time, on the assumption that these were the only process factors that could contribute to changes in things like quality, cycle time, and efficiency. As we expected, based on the Software Process Assessments, most KPAs did not show substantial change over time. Recall that the organization, even in its first assessment, satisfied 85% of the Level 5 KPAs. The exceptions were in the process areas of Defect Prevention and Tracking. Defect Prevention activities increased over time, while Tracking decreased.

These trends are illustrated in Figure 1. Each release of every product for which we have data is plotted as a point. The location of the point with respect to the horizontal axis represents the date the project was officially begun. Its location on the vertical axis indicates how consistently the release met the goal requirements included in Defect Prevention and Tracking. Lowess [1] smoother line is shown to illustrate the trends. Both trends are significant at 0.05 level using ordinary linear regression predicting response using starting time of the project.

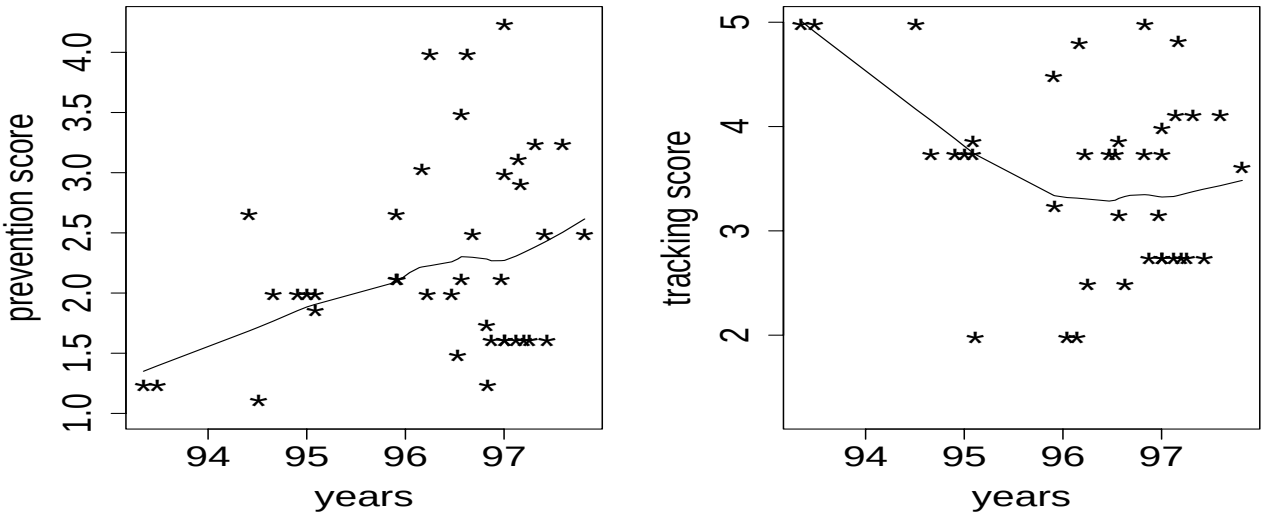

Figure 1. Trends in Defect Prevention (left panel) and Tracking (right panel) over time, for ASCC releases.

4.4. Configuration Management System (CMS) data

The organization used Sablime, an internally developed configuration management tool, which evolved from Extended Change Management System (ECMS) [9] change control system, which in turn uses Source Code Control System (SCCS) [13] for source code version control. There were two levels of source code changes that were distinguishable in the organization: delta, or atomic changes to the source code files produced by SCCS whenever individual file was submitted and Maintenance Requests (MRs), or logical changes needed to perform a task like a bug fix which may touch several files and generate numerous delta. We used change summaries extracted by the SoftChange system [10] that included time, developer, size, file, and MR associated with each delta and interval, developer, and purpose associated with each MR.

4.5 Project tracking data

The reported project information included size, staff months, number of faults, and interval. Unfortunately, the reported data was not measured consistently across projects and sometimes was not consistent with change based-data; therefore it was not used in the models. Some projects measured size in function points (FP) and some in lines of code (LOC). The reported function point and reported LOC measures did not correlate well with the amount of code developed (as obtained from change history data based on all changes associated with a project) or with the reported staff months. Furthermore, the reported interval did not correlate with the duration of the coding phase measured by time difference between the last and the first change. These serious validity problems made the reported data unsuitable for further analysis. ASCC example highlights some of the pitfalls of using project tracking data for benchmarking.

5. Results

The following sections summarize our results in both the areas of the organization's process orientation, and operation as a software factory.

5.1. Process orientation

The Silver Bullet vision also called for ASCC to be culturally distinct from Lucent (then AT&T). It was recognized early on that having a culture with a process-conscious mindset would be difficult to achieve with veteran developers from existing AT&T organizations who already know instinctively what needs to be done. Thus the decision was made to hire mostly people fresh out of college who could be trained to follow the process. (This had the added benefit of lower salaries.) To emphasize ASCC's separate culture, the organization was placed in a separate geographical location as well. North Carolina was chosen because of the resource pool, as well as infrastructure and cost advantages. The result was a culture that lived by the process.

Every person we interviewed, which includes a few who are critical of many aspects of the organization's process orientation, expressed the view that the emphasis on process is good for the organization and its customers. We heard many comments suggesting, for example, that the consistently high level of product quality was directly attributable to the emphasis on process. Many people commented that the "atmosphere" of discipline and rigor created by the process emphasis was probably the most important result of focusing on process. Specifics of how particular tasks are done, how processes are defined, and so on, were thought to be much less important than the expectation, shared essentially by everyone, that processes form the foundation for the way work is done.

In spite of this broad endorsement of the process emphasis, there were many difficulties in the execution. In the

remainder of this section, we focus on how ASCC brought about this process orientation, things that worked well and others that didn't, and lessons that can be drawn from this wealth of experience.

5.1.1. Process specification separated from process execution

In order to give adequate attention to the development and maintenance of an effective, efficient process, the project decided early on that process engineering should be a distinct position, and should be separate from those whose responsibility is to execute the process, i.e., to develop the product. This explicit assignment of responsibility and unambiguous allocation of resources was intended to ensure that development processes received sufficient attention. In addition to this stance on assignment of responsibility, the organization decided to write a detailed process up front, so that product development could ramp up quickly and be executed in a disciplined way from the beginning.

These principles were put into action by bringing in several "process engineers," with considerable experience and expertise in defining software processes. They were commissioned to define the development process, and to design a development environment in which that process could be executed efficiently. The process engineers were Lucent consultants, separated from the developers both organizationally and geographically. They were brought in to define the process before the organization actually existed. At the peak of process definition activity, 7 or 8 process engineers were working on various aspects of the work. The highly detailed defined process they created included not only tasks to be executed (represented as data-flow diagrams), but also training curriculum, measurement, project management, tools, staff functions with required skills, and platform (focusing on cataloging reusable software).

The process was represented in a commercial tool that required considerable time desk checking and auditing to make sure it was consistent. Training was provided for the developers, who were to execute this process.

There was some resistance to the external process engineers from the beginning. Some developers felt that since they were the ones doing the job, they best understood how it should be defined. There was also a sense among some of them that communication with the external process engineers was too difficult since they were located elsewhere. Further, the developers didn't feel a sense of ownership of the processes, did not believe the processes always reflected a good understanding of how the work should be done.

Several developers also expressed the view that the process engineers often seemed relatively unresponsive to the developers' concerns. There was concern, very early on, when the processes first were applied to actual projects, that the numerous requests for changes seemed to be getting lost. A system for tracking change requests was implemented, and almost immediately over 180 modification requests were submitted, overwhelming the process engineers.

The situation seemed to improve considerably when the organization hired local process engineers. Several developers remarked that it helps to have process engineers who are immersed in the local culture, who "know how we do things." As one developer put it, with local process engineers, "It is like the people here own the process, and it is part of [our] culture." There is also a widespread belief that the local engineers are more responsive, and that developers now have more confidence that the on-line process descriptions reflect reality. The local process engineers are also in a better position to champion process issues, to "beat the drum to do process."

5.1.2. Pre-defined process and the need for simplification

One of the primary threads of work in improving the organization's process was to simplify it, to remove steps and work products that did not add value. As mentioned above, the process was written before development capability of the organization actually existed. So even though the process was designed by experienced, highly regarded process engineers, it was written in the complete absence of any experience with the specific development organization in which it was to be executed.

Defining a process, particularly for a new organization, is a very challenging undertaking. In a certain sense, it is very like programming, and software processes can themselves be thought of as software [11]. Recommendations for defining a software process, on the other hand, seldom advocate programming these new processes from scratch, without first thoroughly understanding the existing process [6]. While we are not aware of any published recommendations that advocate skipping over this step, it is often left out in practice. The feeling often is, "We know the current process is no good, and we want to change to the new process as quickly as possible. Let's go right to the new one."

The ASCC experience is an excellent chance to evaluate the risks of this strategy. The ASCC did not have the option of writing down an existing process, since process definition began before the organization existed. But they did make the choice to try to define a complete, detailed process at the outset, rather than starting with a bare-bones process and iterating to evolve it as needed. Again, this was done in order to ramp up as quickly as possible, to put a process that would allow them to "hit the ground running."

Over time, as the ASCC process was applied to actual software projects, many serious shortcomings became apparent. There were many holes, i.e., things not addressed adequately or at all. The process focused on the technical tasks, such as writing requirements, doing the design,

inspecting, testing, and so on. Other essential tasks, in particular, many of the front end and project management tasks, were left out or dealt with only cursorily. It was not clear, for example, how the organization committed to a project, lined up resources, or what a project plan should look like. In addition, there was no provision for administering the code, including tasks like configuration management.

In other cases, even where a task was covered in detail by the defined process, the definition was perceived not to be adequate to give sufficient direction about how to carry out various tasks. The original idea was to have a process that was clear enough and detailed enough that someone right out of school, with little or no experience, could begin working productively with relatively little mentoring. This goal was overly optimistic; even where the process was quite detailed, it did not achieve this result. New people needed considerably more mentoring that was expected, in order to become productive. As one engineer put it, the process "told them the flow, but it still didn't tell them how to do the actual work itself."

The immediate response to these shortcomings was to add to the process. As one engineer said, "We tried to make our process reflect every experience that our project had. … we tried to over-engineer everything." Many of these additions, however, proved to be project-specific, and did not apply or did not work well in a new project. This "first attempt," particularly after additions were made, seemed uniformly to be too "big," too wordy, require the production of too many documents, and was too difficult to change.

This process expansion began to be reversed when a customer for one project insisted on incremental delivery of features on a 12-week cycle. Rather than producing the full set of features for the project at the end, the customer required delivery of a subset of the features 4 times a year until the project was complete. In order to meet this challenge, a simplified, stripped-down version of the process was created. Templates were scaled back only to what was essential, many work products were identified that only needed to be produced once for the entire project, rather than for each delivery, other work products were eliminated entirely. In describing this paring back of processes, many developers used the famous quote from Mies van der Rohe, "less is more." It was described as a big step, a revelation.

Important lessons were learned from the "12-week process," and were quickly applied on a larger scale. As one developer said, "I think this was a really a good stepping stone, and I think probably very important now for the evolution of the processes." Process simplification was, and continues to be, applied to the processes used by other projects. In one case, the simplified 12-week process was tailored to a different project in a matter of days, once the process had been reviewed and feedback from its previous instantiation provided. The simplification seems also to have made it much easier to apply in new projects.

There is some quantitative evidence that the simplified process increased the throughput. We tested this claim in two different ways. First, we modeled the release interval, correcting for size of the release and level of project tracking, comparing the 12-week process to the process used more generally within ASCC. Although the estimated coefficient was negative, hinting at higher throughput for the 12-week process, the standard deviation was large enough (due to small sample size of releases with 12-week process) to make the result inconclusive.

However, we then checked if the 12-week process decreased the interval between the open and close of each individual MR. We found that the 12-week process significantly decreased the interval (p-value $<< 0.01$) of individual MRs. This is important because every release consists of a large number of MRs. Reducing MR interval keeps project on track and reduces change dependencies that may happen when a large number of MRs are open at the same time. This fact might be the principal cause of satisfaction with the 12-week process. It also strongly suggests that the simplified 12-week process increased efficiency at the MR level, which could result either in lower cost or shorter interval.

Our analysis of the effects of decreasing levels of tracking is also relevant to the issue of simplification. Recall that tracking was one of the measures that varied substantially over releases. We found that lower scores on tracking, i.e., "less" tracking, was associated with shorter intervals, both for the entire release and for individual MRs, even when we adjusted for size (of MR, of release) in fitting the models. While this may seem somewhat surprising, remember that project tracking was performed consistently throughout all releases in the sample, so we are not concluding that it is better not to track projects. Rather, we think it likely that there is some optimal level of tracking, and exceeding this level may generate enough overhead to actually increase interval.

5.1.3. Process improvement: responsibility and accountability

Assigning responsibility for designing and implementing process improvements has proven to be a difficult problem. Originally, process improvement work was expected of each team, including the process team, build team, delivery team, and so on. This was less than satisfactory, because people tended to do relatively trivial things, like a minor update of a template, in order to demonstrate some process work. But there were no focused efforts attacking problems in coordinated ways.

The next stage was the creation of a process council that was charged with the responsibility of orchestrating process improvement. They posted process improvement opportunities, and had a complicated scheme of allocating

credits for those undertaking the work. The council had the responsibility of tracking who had done how much work. This was probably an improvement in focusing the work, but it was a major administrative headache. It also did not resolve the fundamental problem of the conflict between process and project work. There was another council, composed of project team leaders, and the two councils found themselves in contention for resources.

The next stage attempted to solve the contentious council issue by merging the two, creating the project and process council, which had both responsibilities. This council was charged with achieving center-wide objectives, cast in objective terms of interval, quality, and cost. Each team was required to contribute some significant process. This seemed to work better, because everyone had the same objectives, and what had been two groups had to work together toward these objectives.

The final stage created a quality council, rather than the project and process council, and began to rely primarily on root cause analysis to identify process work that needed to be done. They are following a common approach to quality with four steps: identify a problem, identify root causes, take corrective action, and verify that the problem has been corrected. It is too early to say how successful this approach will be, but it clearly has strengths and weaknesses. The strength is that now responsibility is clearly vested in the council, so it is hoped that this will help motivate effective action. On the other hand, the sentiment was often voiced that people now do not regard process improvement as everyone's responsibility. Many developers now appear to see it as the "Council's problem," and they no longer have as strong a sense of ownership and responsibility for the process.

There seem to be issues that recur, and are extremely difficult to resolve. One is that process work and project work generally experience a resource conflict in practice. Many interviewees argued that in the big picture this should not be so, since the process work defines what one does in a project. So process work ultimately contributes in a very fundamental way, to project success. Yet in practice, this conflict is difficult to resolve. Process work represents an investment for the future, and the short-term payoff of project work (product out the door sooner) will always be higher when resources are scarce. Focus on the longer term view is difficult to sustain.

The other recurrent issue we noted is the question of how widely to distribute responsibility for process improvement. Resources are less likely to be pulled from process work if there is staff dedicated to these efforts. On the other hand, this approach can leave everyone else feeling as if they have no responsibility at all for improvement. This can be unfortunate, since those executing the process have the best sense of what will actually work, and must in any event cooperate with improvements or they are unlikely to make their way into actual practice. Distribution of responsibility more broadly can result in the feeling that since everyone is "responsible," no one is really responsible. It is very hard for anyone to resist project pressures, or to plan improvement efforts so coordinated action can be taken.

5.2. Software factory arrangements

ASCC was effectively an internal software subcontractor, created to provide software development services to other Lucent product organizations who have software needs but cannot afford to develop it themselves. The motivation was that ASCC could build their software more cheaply and deliver at more predictable schedules. This "software factory" arrangement had several implications in terms of product ownership, dealing with organizations with different process requirements, and forming domain expertise.

5.2.1. "Software Only" Contracts and Product Ownership Issues

The model for ASCC's process seemed simple enough since it did not deal with customers directly, but relied on whatever requirements was passed to it by the product owners. Also, most of the software it built was transitioned to the product owner upon completion. Based on these assumptions, the process engineers created a process that focused on the technical activities, as mentioned above. They had serious difficulties, however, with some of the non-technical activities, which include writing proposals, acquiring projects and negotiating requirements. The missing processes can be classified in two categories: front-end and integration.

Front End Management

One problem that came up at the front end was sorting out project requirements. Oftentimes, ASCC works in collaboration with several other development organizations. In early projects, the work distribution was not clear, e.g., either the requirements specification was unclear or incomplete, or there was miscommunication about group assignments. This resulted in a lot of rework effort. Later projects mitigated these issues by assigning dedicated people to work with systems engineers early on. This made the development smoother. The problem that has to be managed, of course, is the intimate relation between software and other activities, such as systems engineering, and other parts of the product, such as hardware. Interfaces need to be clear, handoff points well defined, and there must be some effective mechanism for the inevitable negotiations about changes.

Integration Problems

ASCC's early projects experienced a host of integration problems not unlike other organizations involved in geographically distributed software development projects [5]. One problem was the need to do integration remotely, at another site outside of ASCC. In many cases, integration had to be done at the owners' site because the size of the integrated product was beyond the capacity of ASCC to

handle. This physical separation slowed down the integration testing, debugging, and fault fixing cycle. Aside from this, product owners typically subcontract only a part of their product development to ASCC. Once finished, the ASCC portion is then integrated at the owner's site. This led to problems in testing for feature interaction and assigning responsibility for particular faults.

Several countermeasures were set up to fix the integration problems. The importance of intergroup coordination was emphasized, especially after the first software process assessment. (Our survey data indicate high scores for intergroup coordination, especially, keeping track of issues across multisite teams.) ASCC also realized the importance of the Operations, Administration, and Maintenance (OA&M) subsystem as a platform for incremental integration, in which product features are added to a skeletal system and tested piecemeal. The result was that few of the latter projects mentioned integration being a serious problem.

We should note here that working closely together across major organizational boundaries is not necessarily addressed well by Intergroup Coordination, as conceived in the CMM. Intergroup coordination extends only to groups within the same organization, e.g., software and hardware design, and does not apply as well to the complex relationships with external organizations. Software factories, generally set up as specialty contract shops with customers external to the organization, suffer from the more complicated "virtual enterprise" type problems, where the groups do not necessarily have a common management chain, culture, and so on. This is an increasingly important area not addressed well by the CMM.

5.2.2. Adjusting to the product owner's environment

In many instances, ASCC worked with organizations that have different software processes and tool environments. For the most part, adjustment to the product owner's environment did not cause big problems. In some cases, the product owner's processes were defined at a coarser level of granularity, which allowed ASCC to use their own processes. In other cases, ASCC had to follow the product owner's existing processes, which means, at a given time, different development groups within ASCC would be following different processes. Eventually, ASCC developed a simple, lightweight process model which can be easily customized to work with different customer processes.

With respect to working in different tool environments, the biggest impact occurred when the product owners used different configuration management systems. Since its inception, ASCC has been using Sablime for configuration management. Because of this, many of their process tools are also tied in with Sablime. However, several recent projects are using other configuration management systems like ClearCase [8] and ECMS. In addition to costs of retraining development teams to be familiar with these systems, these have severely limited the usability of their process tools, some of which have been hardcoded to extract data from the Sablime database. This is an example of the important point that it is not wise to tightly couple processes and tools -- it makes both harder to change.

5.2.3. Leveraging the software factory advantage

ASCC tried several business strategies to leverage its core software competence. At its inception, ASCC worked with one or two product owners. This carried a high risk for ASCC as there would be a major funding problem if one of them were to pull out. They quickly modified their strategy by going after several small projects from different product owners. This approach enabled them to obtain a constituency among several business units within Lucent. However, this prevented them from gaining expertise in a particular application domain. (Some of the past projects included expert systems, information systems, operating systems, graphical interfaces, database managers, broadband applications, etc.) In the past couple of years there was a trend toward settling down into a few niches. Now ASCC has gone back to working with a few large projects.

It has been pointed out previously, e.g., [2] that one of the primary advantages of a software factory derives from planned economies of scope, i.e., "cost reductions or productivity gains that come from developing a series of products within one firm (or facility) more efficiently than building each product from scratch in a separate project. (p. 8)" Reaping the benefits requires some control over the work that is accepted, however. There are obvious economies of scope for successive releases of the same product. Accumulated domain expertise, as well as acquiring specialized tools and know-how in a domain, is also likely to produce a substantial advantage. However, economies of scope are lost when the contracting organization does not, or can not, build on past experience in some substantial way, but rather ramps up one novel effort after another.

5.3. Organizational learning for continuous improvement

Learning from experience is an important element of continuous improvement. In order to learn effectively, ASCC has instituted some formal mechanisms. In addition, informal means of learning is also encouraged.

5.3.1. Formal learning

Formal learning efforts were carried out in two ways: postmortems were conducted to learn how to improve the process and defect root cause analyses were conducted to learn from past software faults. The quantitative evidence available to us fails to indicate any positive effect of prevention activities on software quality, as measured by the number of bug fix MRs after the release date, or measured by the total number of bug fix MRs (the models adjusted for size of the release). In this section, we look at

the ways in which ASCC strove to learn from their experience, and the difficulties these efforts encountered.

Postmortems. The postmortem process was designed to support the process of process improvement. It focuses on identifying gaps between expectation of what the ideal process ought to have dealt with a particular situation and the reality of how the situation had been handled. After the gap is identified, countermeasures are specified. Usually, countermeasures simply document the gap as a process MR. If the gap requires a larger, concerted effort, then a task team may be formed to propose a solution.

Postmortems generated a mixed response from the developers and managers we interviewed. Some were very positive about the benefits of postmortems. Others see postmortems as a lot of effort with little return. Our conclusion was that developers who have worked across a variety of projects seem to find postmortems to be ineffective while developers who stayed with the same project seem to find postmortems effective. Therefore postmortems appear to be effective in carrying over key learnings to succeeding releases of the same project but are not effective in carrying these over to other projects. There are several reasons cited for this. What worked on one project may not make sense when applied to another project. Many of the issues that come up in the post mortem are very specific and only apply to a certain domain. In addition, the same persons who learned from the postmortem frequently move on to other projects where the results may not be applicable or leave the company altogether, thus putting away the postmortem results without taking further action.

Root Cause Analysis. Root cause analysis on software faults is conducted in order to prevent recurring software faults from happening again. Causal analysis of faults have been done in other companies, identifying the cause of the fault, steps for preventing the fault and removing similar faults that may exist in the system. These recommendations ideally result in more effective testing and inspection processes. In other cases, statistical analysis of faults leads to identification of fault-prone modules that are then rewritten.

In ASCC, root cause analysis is relatively new. Hence there hasn't been much of an impact in terms of improving existing QA processes. Root cause analysis data is entered into a field in the change/defect tracking database but nobody knows what to do with it once it is there. Occasionally, root cause analysis has led to the identification of problematic files that have been rewritten.

5.3.2. Informal learning

In the course of doing certain things repeatedly over several projects, opportunities arise which encourage a more informal means of learning. One of the biggest breakthroughs was being able to quickly ramp up new projects and ramp down old ones. Key to this was the Operations Administration and Maintenance (OA&M) subsystem. The OA&M subsystem is the foundation on which the rest of a software product is built. It provides the framework for process execution and interprocess communication, error handling, and installation of the system. It is often added as an afterthought after the rest of the major functionalities have been planned and implemented. ASCC developers found themselves repeatedly building this framework for every new project. They discovered that, if the OA&M subsystem is created in advance, then it can provide scaffolding for early unit testing without waiting for the whole system to be put together. It also provides a platform for incremental integration so that not all pieces have to be there at once. Because of ASCC's success in doing OA&M, it became one of their domains of expertise and other business organizations had ASCC do their OA&M.

5.4. Lessons Learned

Separating the function of process engineer from developer can be a successful tactic for ensuring that process definition and maintenance receives the attention it needs. But the people who fill those roles should not be *too* separate in the sense that the process engineer should be available, steeped in the local culture and environment, and considered "one of us" by the developers, who thereby share a sense of ownership of the process. At a minimum, it seems that process engineers should be co-located with the developers, and should be within the development organization rather than an external group. Job rotation between process engineering and product development, although not observed in this case study, may be an effective way of maintaining this vital link.

Maintaining and improving the process should be treated much like maintaining and improving physical facilities, or the computing infrastructure. While an out-of-date process is less visible than, say, a leaking roof, pulling resources from either one to take care of short-term needs will simply allow the problem to grow worse, and probably more expensive to fix.

Process improvement must be everyone's responsibility, but there must be a small core of people with process as their primary responsibility, to lead and plan process efforts. These leaders should work to find slack resources, e.g., just after a major release, when others are experiencing less project pressure, and may be more willing and able to spend time on process work.

Economies of scope are easiest to obtain across multiple releases of a product, an advantage that presumably extends to product lines. There are likely to be substantial advantages for development of different products within a domain. When new projects continue to crop up in new domains, however, the organization is continually suffering from sparse domain knowledge, and benefits little from its previous experience. It is very difficult to carry learning across domains. There must be some control over the work

that is accepted in order to take advantage of economies of scope.

Even a contract shop cannot ignore the need to have processes for non-technical activities as well as technical activities. Work with systems engineers up front to agree on requirements. Invest time in automating repetitive tasks.

Continuous intergroup discussion and incremental integration can offset problems of widely distributed software development.

Invest time and effort in developing a portable toolset and a simplified common process that can be easily customized. Keep processes and tools as separate as possible. A tight coupling will make it much more difficult to evolve or customize either one.

Starting an organizational culture from scratch is a feasible alternative to attempting to engineer a culture shift within an existing organization.

Postmortems are effective in carrying over key learnings from one release to the next within the same project. Additional mechanisms would be required to transfer learning across projects.

The principles of the software factory and process improvement do not automatically lead to a successful development organization. One must, for example, take into account the goals and circumstances of the particular organization in order to

- achieve a useful level of detail in process definition,
- define separate but not disconnected roles of process engineering and development, and
- strike the correct balance between collective and individual responsibility for process improvement.

References

[1] W.S. Cleveland, "Robust Locally Weighted Regression and Smoothing Scatterplots," Journal of the American Statistical Association, Vol. 74, No. 1979, pp. 829-836.

[2] M.A. Cusumano, Japan's Software Factories, Oxford University Press, New York, 1991.

[3] S. Gelman, "Silver Bullet: An Iterative Model for Process Definition and Improvement," AT&T Technical Journal, Vol. July/August, No. 1994, pp. 35-45.

[4] S.J. Gelman, F.M. Lax and J.F. Maranzano, "Competing in Large-Scale Software Development," AT&T Technical Journal, Vol. November/December, No. 1992, pp. 2-10.

[5] J.D. Herbsleb and R.E. Grinter, "Splitting the Organization and Integrating the Code: Conway's Law Revisited," Proc. International Conference on Software Engineering, 1999, pp. 85-95.

[6] M.I. Kellner, L. Briand and J.W. Over, "A Method for Defining and Evolving Software Processes," Proc. International Conference on the Software Process, 1996, pp. 37-48.

[7] M.S. Krishnan and M.I. Kellner, "Measuring Process Consistency: Implications for Reducing Software Defects," IEEE Transactions on Software Engineering, Vol. 25 No. 6, 1999, pp. 800-815.

[8] D.B. Leblang, "The CM Challenge: Configuration Management That Works," Configuration Management, W.F. Tichy ed., John Wiley & Sons, 1994.

[9] A.K. Midha, "Software Configuration Management for the 21st Century," Bell Labs Technical Journal, Vol. 2, No. Winter, 1997, pp. 154-165.

[10] A. Mockus, T. Graves, A. Karr and S.G. Eick, "On Measurement and Analysis of Software Changes," Technical report, Bell Laboratories, 1999.

[11] L. Osterweil, "Software Processes Are Software, Too," Proc. International Conference on Software Engineering, 1987, pp. 2-13.

[12] M. Paulk, B. Curtis, M. Chrissis and C. Weber, Capability Maturity Model for Software (Version 1.1), Technical Report, CMU/SEI-93-TR-024, Pittsburgh, Software Engineering Institute, Carnegie Mellon University, February, 1993.

[13] M.J. Rochkind, "The Source Code Control System," IEEE Transactions on Software Engineering, Vol. 1, No. 4, 1975, pp. 364-370.