Software Process Improvement in Web Time[1]

Karl E. Wiegers

Process Impact
716-377-5110
www.processimpact.com

Much of the software process improvement literature describes how large corporations and government contractors have changed their software development and management processes over a long period of time. Commercial application developers and Web development groups sometimes conclude that process improvement doesn't apply to, and isn't feasible for, their time scales, product types, and cultures.

Not so! This article relates the five-month process improvement experience of a team of about 25 people doing Web development at a major corporation. We tackled areas including project management, change control, requirements engineering, and quality practices. The tangible and cultural benefits we achieved have the potential to help this group meet its commitments to deliver Web applications ("sitelets") in the very public eye: the company's 41,000-page Web site averages more than one million hits per day.

The software practice areas the group chose to improve are neither novel nor profound. Indeed, that is a significant message. Even a group that undertakes fast-paced Web projects can benefit from the application of traditional software process improvement approaches. Like any other project, Web projects have users, requirements, schedules, resources, and quality goals. Superior technical and management processes can yield superior results in the world of Web time, just as in any other software development setting.

THE GROUP

The Web development group at Eastman Kodak Company began several years ago with just a few people developing simple sites. They soon experienced rapid growth in team size, application size and complexity, and the number of new sitelets requested by business units. The group came to include a software architecture team of about a dozen highly experienced and talented developers and a "customer experience" team with about 10 young, creative visual designers and several human factors specialists.

Unfortunately, development and management processes did not evolve in parallel with work demands. Practices that worked well for a few people doing small projects were not adequate for the larger teams, more complex projects, and backlog of requests that came about over time. This led to the group's internal commitment to focus some energy on process improvement. Their goals included

- managing the huge influx of new work
- developing a foundation of common practices to help new team members quickly become effective
- maintaining the high quality of work while improving productivity
- exchanging knowledge effectively among group members

Each project is assigned a multifunctional team, to avoid the expectation that everyone be a fully

[1] This paper was originally published in *IEEE Software*, July / August 1999

proficient generalist. A typical project team might include a project leader, a technical lead, a design lead, a business unit representative, a software developer, a database expert, and a human factors expert. One consequence of this structure is that each team member works on several projects at once, with the accompanying challenge of setting priorities and the inefficiency caused by frequent task switching and many communication interfaces.

Some development infrastructure and process was already in place in this group, including a recommended Web site creation process. However, there was considerable variation in the ways different individuals performed common activities, and many holes remained in the development, management, and quality practices being used.

OUR APPROACH

The software architecture team was responsible for developing the software behind each Web sitelet. As their customers' quantity and quality expectations increased, the software architecture team members realized that their current approaches were not adequate to the influx of work. A brainstorming session pointed to several areas for possible action: peer reviews, testing, problem tracking, requirements gathering, and project planning. Additional hot spots soon became clear, including shortcomings in the Web site creation process and the need to prioritize the many incoming requests for new work. As an internal Kodak consultant, I joined the software architecture team as a full-time process improvement leader and worked with them hands-on for five months to address these and other issues.

Team members were honest about acknowledging their problems, an essential precondition for effective process improvement. While they had the usual concerns about "process" adding unconstructive overhead to their work, they appreciated the prospect of some increased structure. The structure we eventually provided ranged from more focused post-project review meetings to templates for documenting project plans and requirements.

As with most development groups, many of our problems related to the key process areas found at level 2 of the Capability Maturity Model for Software.[1] However, we chose not to pursue a pure CMM approach. We were not concerned about specifically satisfying the criteria to achieve CMM level 2, although we certainly wanted to enjoy the level 2 results of predictability and stability. We would have been foolish had we not used the CMM as a guide for addressing our problem areas; it would have been equally inappropriate to apply the CMM dogmatically.[2] While my understanding of the CMM helped direct our activities, we never even discussed the CMM during the 5 months I worked on this process improvement project.

We focused on a few improvement areas at a time, with two or three group members collaborating with me in a small working group for each area. We wrote an action plan for each improvement initiative using the template illustrated in Figure 1. The action plans identified goals for the improvement activity, measures of success, the participants, and up to 10 individual action items. Every action item identified an individual as the owner, a target date for completion, deliverables to produce, and resources needed.

A simple, focused improvement action plan makes it easy to know what tasks must be executed and provides a way to track progress. If you need a full-blown project management tool to manage your tactical action plans, they are scoped too large.

The methods for delivering process documentation to the group must match the culture. As this was a Web development group, publishing the process documents on our group's intranet was an obvious choice. We published the more complex procedures in Adobe Portable Document Format (PDF) so that users could print a complete, formatted copy. The procedures were also broken into hierarchically sensible components and published in HTML format. Templates to help team members create project documents, such as project plans and requirements specifications, were published as Microsoft Word documents.

```
Project: _____________________________   Date: ________
Estimated Completion Date for All Activities: _______________
Goals:
Measures of Success:
Scope of Organizational Impact:
Staffing and Participants:
Name                Role                        Time Commitment
_____________    _____________________       _________________
Tracking and Reporting Process:
Dependencies, Risks, and Constraints:

Action Item Number: __  Owner: _____________    Due Date: ________
Description of Activity:
Deliverable(s):
Resources Needed:
```

Figure 1. Process improvement action plan template.

The specific improvements we undertook in each process area are summarized in Table 1. While I cannot claim that every improved practice is being applied religiously by every member of the team on every project, each technique has been used enough times to demonstrate success and to lay the foundation for future benefits. Whether the group can sustain these improvements in the face of continued growth and time pressures will depend on management commitment and the education of new team members.

PROJECT MANAGEMENT

As the number and size of our projects increased, we realized we needed improvements in our project management practices. Many small software groups equate a project plan with a work breakdown structure or a schedule of major tasks. To succeed, though, even small, fast-paced projects need more detail in their plans. The trick is to develop plans that are just detailed enough to make sure you have a thorough understanding of your project and thereby the ability to control it.

Project plan template

We began by defining a standard template for our project plans. Document templates remind the author to think about aspects of the project that might otherwise be overlooked; they also provide sections in which to capture the myriad bits of information necessary for effective project execution. We started with IEEE Standard 1058.1, a template for software project management planning, and adapted it to fit our needs.[3] (The current version is IEEE Std 1058-1998.[4]) Templates should always be tailored to your specific circumstances.

One of the first people to try out the template on a real project came to me one day, somewhat discouraged. “I wrote the plan, but I didn’t know what to put in many of the sections,” she lamented. She was concerned not because the template contained a lot of sections that weren’t important to her project, but because the sections *were* important and she simply did not have the information. The project plan template was well received because it provided a valuable framework for discussions among project stakeholders and helped elicit the necessary information.

TABLE 1. IMPROVEMENT AREAS PURSUED AND APPROACHES USED

Improvement Area	Approaches Used
Project management	Project plan template Project prioritization model
Post-project reviews	Post-project review process description Collection of lessons learned
Risk management	Risk documentation template Informal team risk analysis Risk management action plan
Change control	Change control process description Change control tool
Requirements engineering	One-day requirements training class Requirements specification template Workshops to elicit use cases
Development life cycle	Defined life cycle for Web site creation process Procedures, checklists, templates, examples of deliverables
Peer reviews	Half-day peer review training class Peer review process description and work aids

One section of the project plan template clarifies the roles and responsibilities of the various project participants. Confusion about such roles emerged as a problem at two post-project reviews. For example, on one project, various participants from the customer experience team thought several different people were the prime point of contact from the software architecture team. Clarifying roles and responsibilities enhanced the group's culture by helping diverse people collaborate more effectively on these multidisciplinary projects.

Prioritizing projects

Software process improvement research usually addresses the planning and tracking of large projects. The Kodak Web group, along with many software maintenance groups, faces a different challenge: how to deal with a large number of new project requests. At one point, our queue contained more than 150 requests for new or enhanced projects. We decided to develop a project prioritization model by adapting some principles from Quality Function Deployment.[5] With this model, we hoped to identify the most appropriate projects to undertake using the limited resources available, chosen from the many worthy projects that our customers had requested.

We identified a dozen factors that indicate how favorable a proposed project would be for us to undertake. The factors include

- immediacy of the need
- level of technology risk

- extent to which the project could exploit our current Web development capabilities
- business value
- alignment with known demographics of our current Web site visitors
- degree of user interface complexity

We assigned a relative weight to each factor; the weights totaled 100. Then we defined a rating scale for each "favorability factor," whereby each proposed project could receive a score on each factor. By multiplying factor score times factor weight and summing the results, we could generate an overall favorability score for each candidate project.

We calibrated the model using several completed projects. We adjusted the factors, their weights, and the rating scales until the model yielded results consistent with our after-the-fact assessment of how appropriate each of those projects really was. The model had to generate a significant range of scores so we could distinguish the great project candidates from less desirable ones.

Once calibrated, the model did help us get a better handle on the projects that were most appropriate for us to undertake and helped us manage the large request backlog with confidence that company resources were being invested in the best way. The model was well received by most of our group's stakeholders, including multiple business units, managers, and practitioners. A model like this could be useful for any organization confronted by a large number of projects, all of which are important. Identify your own success drivers, calibrate the model based on work already completed, and use the output from the model as a part—but only a part—of the decision-making process.

POST-PROJECT REVIEWS

The Kodak Web team had previously conducted post-project reviews, but we wanted to make them more structured. Post-project reviews provide an opportunity to look back at a completed project (or from a mid-project checkpoint) and capture the lessons learned.[6] We defined a post-project review process that included a summary of things that went well, a frank exploration of things that could have gone better, and a list of any experiences that surprised us.

Our post-project reviews took the form of facilitated workshops that included the project participants from both the software architecture and customer experience teams. The facilitator emphasized collaboration, steering the participants away from any blaming behaviors. The participants really contributed constructively to these post-project reviews. From the raw data collected during the workshop, we extracted perhaps a dozen lessons learned, which we organized into categories and collected on one of our process Web pages. We tried to write the lessons learned in a neutral tone, so the reader couldn't tell if we learned the lesson because the project went extremely well or because we made a mistake. Future project managers can review these lessons to remind them of practices they should incorporate into their plans, and risks they may need to control. The project team also uses the post-project review results to develop an action plan addressing the key issues revealed by the review.

The insights gained by reflecting on recently completed projects can be of great value in a rapidly evolving environment like Web development. The post-project reviews were well received because all project participants knew that things had not gone perfectly, and they were willing to put their issues on the table and learn from them. Issues raised during the post-project reviews meshed nicely with those generated by the other process improvement activities.

RISK MANAGEMENT

Although it has been identified as a software industry best practice,[7] formal risk management is not regularly practiced on many small development projects. We developed ways to manage the tactical risks facing individual projects, and also felt it was important to take a look at the strategic risks threatening the success of the group's activities as a whole.

The software architecture team identified more than 45 strategic risk items. Many of these pertained to the security and reliability of the Web delivery infrastructure, critical for a heavily trafficked site engaged in e-commerce. As an example, one risk factor was that there might be security shortcomings that could allow users to access off-limits directories on the Web server. Several risk factors related to the leading-edge Web technologies the group was using. Others had to do with organizational, business unit, and management issues, in part reflecting the growing pains of this young development organization.

We applied conventional risk management principles, estimating the probability that each risk could materialize into an actual problem and the impact if it did. This analysis helped us to prioritize the identified risks. Again, we wrote an action plan to begin mitigating risks selected from the top of the priority list. Unfortunately, monitoring the risk management actions did not float to the top of the busy group manager's priority list. To help make risk management succeed, assign a risk officer other than the project or group manager to coordinate risk management activities, and incorporate risk tracking into your routine project status tracking.

CHANGE CONTROL

Small software teams often use an informal approach for making changes to their products. The person who has an idea or who found a bug tells the programmer, who makes the change. This method does not scale up well. Change control and problem tracking constitute another software industry best practice.[7] Despite the malleability of Web site software and data content, discipline is needed to effectively manage changes to Web products. We needed to manage five kinds of changes:

- requests for new projects
- changes to the requirements for projects currently under development
- problem reports concerning current systems or the Web infrastructure
- enhancements to current systems
- content changes in current systems

Practitioners sometimes think that simply installing a problem-tracking tool constitutes a complete change control system. However, a change control system includes both a documented process and clear procedures for the activities to be performed, and tools to automate some of these activities.

We collected requirements for the change control system from several stakeholders who would use it or be affected by it. Requesting voice-of-the-customer input is a way to foster buy-in to the process changes you propose. Those affected become part of the solution, rather than being victims of the improvement initiative. This is particularly important with processes such as change control that redefine the interface through which your customers and other stakeholders interact with the software group.

Based on these requirements and on previous experience, we wrote a change control procedure, had several stakeholders review it, and published it on our internal process Web page. The principal initial benefit was that we established a formal mechanism for collecting and evaluating the requests for new projects that came in from business units every week. For a few weeks, the two people who managed the resulting project queue also piloted the change control function using paper forms. During that time, we explored commercial problem-tracking tools that might meet our need for a Unix-based server and Web-based user interface. The feedback from this pilot helped us improve the process before we committed to selecting, installing, and customizing a specific tool.

The community was receptive to using this change control process to reduce the chaos of the change backlog, with its uncertain change request status and unclear decision making. Any development organization of any size can apply formal change control in this way. One success factor for implementing change control is to minimize the process overhead of submitting and evaluating change requests. Another is to make sure the change process, and the people who practice it, are responsive to submitted requests. If a process doesn't work, people will work around it.

REQUIREMENTS ENGINEERING

While the group had built a series of successful Web sites based on informal requirements collected from business units, such a casual approach breaks down with larger and more complex projects. Although some people hold that documenting the project requirements imposes excessive process overhead on small projects, the costs associated with reworking a software product because of poorly understood requirements can be substantial. We took several actions to improve our requirements engineering processes, specifically requirements elicitation and specification. We began by adapting the IEEE software requirements specification template (IEEE Std 830)3 to meet the nature and scale of our projects. For example, we added a section on internationalization requirements, an important aspect of many Web projects.

The entire Web development team attended a one-day training class on requirements development and management. This helped the participants reach a common understanding about requirements concepts and practices, such as the application of use cases for eliciting requirements, and dialog maps for modeling user interfaces.

Several project leaders began applying the use case method. An initial attempt floundered because too many people attended the workshops that captured and defined the use cases. When we reduced the workshop size from 12 people to six, progress accelerated nicely. Two business unit participants stated that the use case approach helped them to clarify the scope of their project and to understand the actions a prospective Web surfer would be able to perform at their new sitelet. Months later, the Web group manager indicated that the increased emphasis on gathering requirements was a major factor in the group's successful completion of several new projects.

DEVELOPMENT LIFE CYCLE

Millions of people are now amateur Web site developers. While a code-and-fix life cycle can work for the individual assembling a few simple pages, building complex Web sites involves an intimate collaboration among programmers, database experts, visual design specialists, and content providers. A well-defined but flexible development life cycle can help.

Our group's existing Web site creation process was thinly documented, lacked a supporting infrastructure, did not relate to how projects really were performed, and was not practiced in a consistent way. Several post-project reviews revealed problems that an improved development life cycle process could solve. This is another example where traditional software process approaches can benefit a Web development team when the scope of the team's projects exceeds its ability to deliver through informal collaboration.

First, we surveyed the team about the shortcomings of the current process. Based on the responses, several team members then outlined a rational sequence of phases through which our projects should progress. For each phase, we allocated activities, identified the major deliverables, and defined entry and exit criteria. We generated lists of activities and deliverables by combining the contents of the current life cycle, elements from similar models found within Kodak, and the actual experiences of what people really did on their projects.

We recognized that some activities bridge two life-cycle phases. Web projects include development of both content and the enabling software. The site design has to take place before the software architecture design can be completed. We incorporated this time offset in allocating activities and deliverables to each phase. For example, phase 2 includes development of user interface navigation design and the preliminary software functional specification. The functional specification is completed and baselined in phase 3, after the interface design is completed. Life-cycle models that artificially constrain multiple-phase activities to a single phase do not reflect the way people actually work.

We documented the improved Web site creation life cycle on our intranet, with descriptions of each activity and deliverable. Both the customer experience and software architecture teams responded favorably to the improved life-cycle process, because it represented a more realistic approach to executing their

projects. We are now supporting this framework by developing procedures and checklists to facilitate performance of the key activities and by collecting templates and examples of the deliverables.

Early experience suggests that project leaders who feel their management-imposed schedule deadlines are unrealistic may be tempted to hide behind the more extensive documentation demands of this new life-cycle process as a justification for not meeting deadlines. Not surprisingly, this approach does not sit well with upper managers. We need to evolve the culture to a state in which new processes are seen as enablers, not obstacles, and as structures, not straitjackets. This will take time.

PEER REVIEWS

The group wished to begin holding peer reviews and inspections of software work products, another industry best practice,[7] both for the quality benefits of finding defects early and for the increase in information exchange among team members. We began by sending the software architecture team to a half-day class on software technical reviews. Two members of the team then worked with me to develop a suitable peer review process, which included both formal (inspection) and informal review procedures. We published the process documents on our intranet, along with forms for capturing review results and checklists of common defect types found in various work products.

Although the team members were receptive, peer reviews were not one of our great successes. The consensus was that reviews would not become a regular practice until they were incorporated into project plans and schedules. Therefore, we included reviews as key activities to be performed at various checkpoints in our new Web site creation life cycle.

I think it is quite difficult to establish a culture of technical peer reviews. Asking someone else to tell you what's wrong with your work is a learned behavior, not an instinct. To help make a review program succeed, train the whole team, focus your limited review time on high-risk items, and publicly recognize those team members who routinely solicit a little help from their friends.

WHAT WE LEARNED

Five months after beginning our process improvement activities, we surveyed the members of the customer experience and software architecture teams to gauge their reaction to the program. The responses were strongly favorable. We could not quantify the results, but the improvement areas of change control, life-cycle definition, project prioritization, requirements engineering, and project planning were perceived to have yielded the most benefit. Many respondents mentioned the value of reducing the chaos level though improved project management and resource planning. No one indicated that any of the improvement efforts were a waste of time.

The respondents agreed that having a dedicated process improvement leader on staff was highly beneficial. Several participants felt our change initiative had proceeded a little too quickly for comfort, even though new processes were phased in gradually over five months. This points to the limits that any group, even one with highly capable and receptive members like this one, has in accepting and internalizing new ways of working.

Several factors converged to make this process improvement activity successful:

- Accumulated pain from the current state of affairs motivated team members to pursue better processes.
- Management demonstrated initial commitment by bringing a process improvement leader into the group and by clearly stating expectations about the importance of making appropriate changes.
- The team leaders got involved hands-on in the improvement activities.

- Many practitioners participated in devising, reviewing, piloting, and critiquing suggested new procedures and document templates.

- We had access to a corporate repository of software engineering "good practices" that included sample procedures, templates, and work product examples across the entire range of software engineering and management practice areas.[8] If you don't have such resources available, start with the IEEE Software Engineering Standards Collection.[3,4] Another useful product is EssentialSET from the Software Productivity Centre (www.spc.ca), which contains more than 50 sample documents covering the gamut of software project development and management activities.

- Perhaps most significantly, practitioners from both the software architecture team and the customer experience team were willing to try new ways of working, despite their tremendous workloads and schedule pressures.

If I were to undertake a similar enterprise again, I would engage middle management earlier to provide more "pull" for the improvement initiative. This lack of early engagement did not pose a problem in our case, but management must continue to set strong expectations if the practices we launched are to be institutionalized into routine application. This may also help turn action plans into useful actions.

One of the leaders of our group succinctly expressed an ideal attitude toward process improvement. She said, "Our processes give us the ability to select the right projects and complete them on schedule." She recognized that successful software groups of any kind prosper because of intelligently chosen processes, not in spite of them. Structured software process improvement is a valuable means to improve the results from project teams working at the frantic pace of Internet time, as well as for more traditional software development projects. This group's experience demonstrates that even a team dealing with leading-edge technologies, rapid projects, and heavy business pressures can—indeed must—improve the methods it uses to manage and implement software projects.

ACKNOWLEDGMENTS

None of the improvement efforts we undertook would have borne fruit without the commitment of the Web development group's leadership team, LuAnne Cenci, Lee Corkran, Alan Dray, and Deborah Patrie.

REFERENCES

1. Software Eng. Inst., *The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison Wesley Longman, Reading, Mass., 1995.
2. K. Wiegers, "Software Process Improvement: Ten Traps to Avoid." *Software Development*, May 1996, pp. 51-58.
3. *IEEE Software Eng. Standards Collection*, 1997 ed., IEEE Computer Soc. Press, Los Alamitos, Calif., 1997.
4. *IEEE Software Eng. Standards Collection*, 1999 ed., IEEE Computer Soc. Press, Los Alamitos, Calif., 1999.
5. W.J. Pardee, *To Satisfy & Delight Your Customer*, Dorset House, New York, 1996.
6. B. Collier, T. DeMarco, and P. Feary, "A Defined Process for Project Postmortem Review," *IEEE Software*, July 1996, pp. 65-72.
7. N. Brown, "Industrial-Strength Management Strategies," *IEEE Software*, July 1996, pp. 94-103.
8. K. Wiegers, "Improve Your Process with Online 'Good Practices,'" *Software Development*, Dec. 1998, pp. 45-50.