Product Thinking in Process Improvement

Terry Leip, Intel Corporation

Abstract. We are often told that process improvement activities should be managed as a project, but seldom do we hear that they should also be managed as a product. Key decisions ranging from high level strategies to the deployment of improvements can become much simpler when we view the approach of our process improvement work in the same way we would for the development of more conventional software products. This article discusses six examples of this concept that will help you not only simplify process improvement decisions, but improve the odds of success in your process improvement activities.

Background

More than six years ago, Intel Corporation’s 5,000-person IT organization embarked on a journey to improve its internal development processes in an effort to increase development efficiency and address customer satisfaction issues. One of the key approaches learned by the author during this time was to shift his thinking from “Building Processes” to “Building a Product,” in essence, to apply the lessons learned in his software development to process improvement activities.

1. Know Your Market

Software products typically have marketing plans that identify target markets, the size and composition of the segments within those markets, and a description of the customers within those segments and their needs. These plans allow products to be focused on meeting the most important needs of the most valuable markets. In addition, this enables products to be created that are matched to the skill and experience level of users in those markets.

Our early attempts at process improvement work were lacking this type of information and as a result, our process improvement team assumed our market was large, high-risk, and long-duration projects that were led by very experienced project managers. In reality, the majority of the projects were smaller, shorter, and lower risk than had been assumed, and many were led by individuals who were inexperienced in the role (Figure 1).

The inevitable result was that the process improvements did not meet the needs of the user base; both informal feedback and audit data showed that processes were neither well received nor widely used.

Through this and other similar experiences, we learned that we could not make assumptions about our customer base and expect to be successful. Even though our understanding of our markets continues to develop, we now know a great deal more about how the typical project looks in terms of size, experience, duration and other key attributes. The most recent releases of processes and training have been rewritten with these characteristics in mind so that we are not only better meeting the needs of the projects, but we are also no longer supporting...
material which isn’t being used. In a “before and after” survey from our users (Figure 2), we found that customers feel that our product is much easier to use as well as more applicable to the size and complexity of their projects.

**Figure 2: 2007 vs. 2009 Project Manager Survey Results**

### 2. Product Architecture is Important

In software development, a good architecture helps ensure that a product not only meets the requirements, but that the product can be more easily maintained and extended over time.

Four years of documentation changes combined with the lack of defined process architecture left us in the situation where process resided in templates, training had found its way into our processes, and policy had become intermixed with processes. The resulting patchwork of documents had become extremely time consuming for us to maintain and difficult for users to find needed information quickly. In addition, this lack of structure also resulted in users being required to enter the same information in multiple locations.

Faced with increasing maintenance effort and a chorus of customer feedback, we performed an evaluation of the required data for all of our templates and tools. We attempted to identify where the same information was required to be entered in more than one location. This simple analysis yielded some surprising results: For a cascading waterfall lifecycle project (about 70% of our projects), there were a total of 150 duplications of required data over the life of the project. The single largest offender was the problem statement, which was duplicated in a total of six different locations (Figure 3).

In addition to the reductions of required data, we defined a process architecture that included a clear definition of what type of information resided in which type of documents. We then kicked off a project to rewrite our process materials following that architecture, as well as targeting the aforementioned duplications. The results from this effort were very rewarding (a reduction of redundant data entry by more than 70% and positive survey feedback from our customers), however the rework has been costly and could have likely been avoided if an architecture had been clearly defined before our processes were initially developed.

### 3. Beware of “Free Features”

When building software products, it has been often said that there are no free features; everything must be developed, tested, supported, and maintained. In software there is often the temptation to toss in a few seemingly simple features or to add a minor last-minute request from a key customer. The problem, of course, is that maintenance activities are often 60-80% of the total costs of software and every one of these free features must be maintained regardless of their value to the customer.

We have found the same is true with our product; every extra process step, guideline, template, or checklist requires development, testing (i.e., reviews), support (training, coaching, and auditing), and maintenance. We had fallen into a habit of saying “yes” to almost every customer request in an attempt to please our customers, regardless of the scope of use or cost to maintain. As a result, our suite of documentation had grown to more than 1,400 pages spread between 200 or more documents, and even minor changes required updates to a multitude of areas: Process, templates, guidelines, checklists, examples, training, etc. At one point, we had two types of requirements templates and three types of test plan templates with detailed examples to accompany each one. Based on often isolated requests, we had created detailed process steps for seldom used processes and built templates which included far more data items than the majority of the users ever needed.

Our 2008 process release focused on removing these free features that had crept into our product. We condensed the number of similar templates, dropped the examples, and took a bold step to reduce our process documentation and templates to the absolute bare minimum possible (typically one to two pages per document, just focusing on the essentials). The results were dramatic; we shrank our collection of processes, templates and checklists by more than half (Figure 4), and we are projecting savings of more than $25,000 per year in reduced internal maintenance costs, and users actually found that the resulting materials were easier to use.
We also recognized that we needed to address this stream of low-value features and change our current behavior to prevent our product from winding up with the same problems in the future. To do this, we beefed up our process for handling feature requests, including creating a simple tool for scoring change requests as high, medium, or low against four value drivers (Figures 5 and 6).

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value (user)</td>
<td>Low</td>
</tr>
<tr>
<td>Scope of Use</td>
<td>Low</td>
</tr>
<tr>
<td>Cost (user)</td>
<td>Low</td>
</tr>
<tr>
<td>Cost (PMO)</td>
<td>Med</td>
</tr>
<tr>
<td>Score:</td>
<td>2.8</td>
</tr>
</tbody>
</table>

Using this approach and tool, we have been able to reduce the number of low value changes (our overall rejection rate went from 10% to 33%) and most surprisingly, users have been very understanding when we say “no,” in large part due to being made aware that a standard set of criteria is being used in making the decision.

4. **Change your User Interface Cautiously and Infrequently**

In software projects, development teams are careful about radically changing the "look and feel" of an established application because they know that once users are accustomed to using their product, they will need to spend valuable time to relearn things that have changed. In process improvement, the interfaces are the templates, process documents, and web sites where they are contained.

We learned this lesson when we performed our first round of process improvements in 2005 [1] to reduce the complexity of key areas of our processes and templates. When the updated materials were deployed in the organization, we were surprised to discover that many users were upset by our improvements because they had grown accustomed to the look, feel, and the steps of the previous processes. Even though the new versions had fewer and simpler steps, they still required the project teams to spend the time and effort to learn the new interface.

The key learning was that our users want to spend their time and energy doing their work rather than relearning new interfaces to our product. In the intervening years, we have become much more cautious about changing interfaces and have learned a great deal regarding how to minimize the impact of changes. One major improvement was to simply reduce the frequency of large changes and we now work on a cadence of two releases per year, and any substantial changes to interfaces that must happen are grouped into one of those releases rather than dribbling them out over time. This more predictable approach has greatly reduced the complaints and anxiety that we experience with past changes.

5. **You Cannot Just Ask Users What They Want**

One of the most commonly encountered issues when discussing software requirements is that you can not simply ask users what they want and expect their answers to be correct and complete. It is not that users do not want to provide good requirements, but that they are often too close to the issues and often have many unstated assumptions about how they really work. People developing software requirements know that multiple elicitation techniques should be used to properly understand the real problems to be solved and uncover the true requirements.

In the early days of our process work, we routinely collected our process improvement feedback from customers during quality assurance audits and via change requests from any users that cared to submit them. The bulk of these suggestions were for “less process” or minor changes to templates or processes, but seldom addressed issues that would improve overall project execution. Last year, we analyzed the results from more than 60 project postmortems and discovered that virtually none of the issues or solutions identified by the postmortems related directly to our user feedback (see Figure 7).
Given that one of our key business objectives was to meet delivery commitments, we recently started performing formal root cause analysis of projects that missed their committed release date. This data has shown that issues with the performance of vendors and sufficient resources are the drivers for 65% of late project deliveries. In addition, we recently engaged in more sophisticated business-problem focused interviews with randomly selected project managers and test leads to uncover issues impacting key outcomes such as product quality. The results of this activity identified issues such as management dictating scope, schedule and resources, and an IT mandated 26-week project duration limit as large drivers of quality issues (see Figure 8).

While we are still struggling to gather our requirements using more sophisticated methods, our understanding of our customer requirements has improved substantially since we have moved beyond simply asking our customer, “How can we improve our processes?”

6. Provide Strong Customer Support

Many of us have the experience of calling a customer support hotline for a product and after navigating the phone menu waiting for an extended period of time. When we finally speak with a real person, we then discover that they have insufficient knowledge to help us. The key is that without solid support, many customers may become frustrated and simply give up on a product and not bother to use it.

One of our earlier and better decisions was to provide process coaches and quality assurance auditors who not only knew the processes, but also had real-world experience using them on projects. This allowed them to help our customers to understand both the value of the processes and how to appropriately apply them to projects. This support was instrumental in ensuring that project teams did not give up in frustration when they did not know how to use a process or tool. Feedback from project teams has been overwhelmingly positive towards this effort, with many project teams indicating that they would not have been able to adopt the processes without the support. The bottom line is that if it is too hard to use a product, customers will not use it fully or will not use it at all; the same is true for any process improvement “product” as well.
Conclusion
I have provided only a few examples of applying “product thinking” to process improvement, but I continue to find new applications nearly every day. The key is to step back from decisions and ask the question, “How would we address this process improvement issue if this was a software product?” I think you will discover that many formerly perplexing process improvement issues become clearer and solutions become obvious when you make this key shift in your thinking and approach.

Disclaimer:
Other names and brands may be claimed as the property of others Intel, the Intel logo, Intel. leap ahead., and Intel. Leap ahead. logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

© 2010 Intel Corporation. All rights reserved. This article is for informational purposes only. INTEL MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS ARTICLE

REFERENCES

ABOUT THE AUTHOR
Terry Leip is a Process Engineer in the Operational Excellence group at Intel Corporation’s IT Flex Services. He has more than 20 years of software development and quality experience with Intel and Lockheed Martin. He holds a bachelor of science degree from Grand Canyon University, is a Six Sigma Green belt and an SEI authorized CMMI SCAMPI B/C Team Lead.

E-mail: Terry.Leip@Intel.com