Tale of Two Implementations

Ken Schwaber

Abstract: *Many organizations have struggled to successfully spread agile processes beyond the initial beachheads. Two approaches that simplify the implementation and mitigate the pain of the change are presented.*

Overview: Implementing an agile process requires all of the change management skills available within an organization. Even though the benefits are easily understood and even experienced through pilot projects, the width and breadth of change required is daunting. Engineering practices need tightening, project management changes, customer relationships are shaken, collaboration rather than contracts and isolation becomes the norm, and most other assumptions and comforts that underlie an organization's operations are changed. This article presents two approaches for easing the pain of the change and enhancing the probability of success. These are the "two step" and the "wrap."

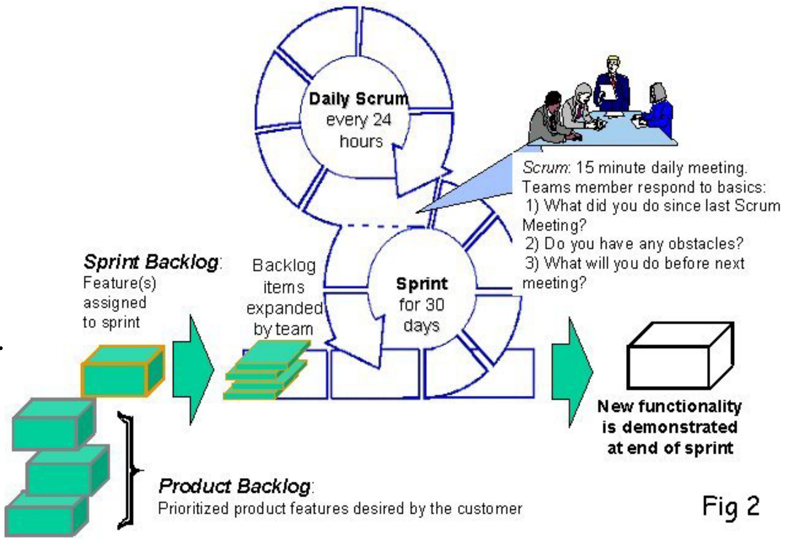
Keywords: iteration, increment, emergence, self-organization, change, product management, development team

Article: Two recent organization-wide implementations of the Scrum agile process may offer some guidelines, even patterns, that may be successfully employed by others. Each implementation employs indirect, even agile, tactics that led the organizations to incrementally benefit from agile processes without undergoing the pain that often results from a direct approach. These two implementations offer an alternative to the resistance to change that often sets in during radical changes to cultures. Implementations are akin to changing the wheels on a locomotive hurtling forward on its tracks; how to change the wheels without untracking the entire train?

The *"Two Step"* implementation was in a small biotechnology company. We'll name this company "AbleX." AbleX filled a niche in the biotechnology and pharmaceutical world. AbleX acquired raw experimental data, repacked it with many vectors of data that uniquely identified it, and resold the repackaged product to companies that required FDA licensing precision. The current development approach was a mix of hacking and waterfall.

Scrum and agile processes use the techniques of iterative and incremental development to create a regular flow of timely products. The techniques of inspection and adaptation ensure the timely presence and certifiable correctness of these products. These techniques increase the control of complex development projects. They reduce the risk of the projects never getting off the ground, not producing anything worthwhile, and diverging into uselessness. A flow diagram of the Scrum process that implements iterations through thirty day "Sprints" is shown in Figure 2.

AbleX was in the middle of upgrading its operational systems, those systems that provided the value-added to the raw data.


Daily Scrum every 24 hours
Scrum: 15 minute daily meeting.
Teams member respond to basics:
1) What did you do since last Scrum Meeting?
2) Do you have any obstacles?
3) What will you do before next meeting?
Sprint Backlog: Feature(s) assigned to sprint
Backlog items expanded by team
Sprint for 30 days
New functionality is demonstrated at end of sprint
Product Backlog: Prioritized product features desired by the customer
Fig 2

Management was very interested in the increased control and reduced risk of agile processes. It was able to achieve both by simply implementing the thirty-day Sprints and Daily Scrums. Inspection and adaptation happen during both Scrum practices. They were easy to implemented with minimal change. Management was then able to better track and control development, converging on increments of functionality every Sprint.

More difficult for AbleX were the concepts of emergence and self-organization. These are the practices wherein Scrum and agile processes deliver breakthrough productivity, creativity, and employee satisfaction. However, these require the most subtle and far reaching changes. For instance, these practices require product management to use brief product backlog, or stories, instead of detailed requirements and design documents. They wondered, how could these brief one-line descriptions make up for the detailed knowledge implemented into these extensive documents? How could a team of individuals with less business domain expertise possible come up with a better, much less an adequate solution?

In many regards, product management anxiety and apprehension were correct. The development teams had always been at a remove from the users and the business, acting as a software factory that relied on the precise instructions of the product managers. We needed a way to transition this awareness and knowledge to the development teams. To do so, we devised the second of the "two steps!" The product managers started skimping by only sketching out areas of less complex requirements and design. Although the product backlog was still complete, some of the backlog referred to very detailed requirements and design, whereas other backlog referred to this sketchy, "stubbed out" information. The team had to figure out the details of the sketchy parts on its own, relying on self-organization and emergence.

The second step facilitated the growing knowledge of the business by the development teams and the increasing trust of the product managers in emergence and self-organization.

The *"Wrap"* implementation was at a large software company that we'll call ActiveX for the purposes of this article. ActiveX came into being through the acquisition of independent companies. Each company provided software products that complemented each other. As a whole, the products provided a development environment for complex, n-tier, web-based applications.

ActiveX was experiencing rapid growth and dominated its software niche. The demands of the growth precluded consolidating its acquisitions either geographically or through similar processes. The seven sites were worldwide, spanning the full set of time zones. Their engineering and management practices ranged from almost CMM-level compliance to those more commonly found in the .com companies of the 90's. Systems architectures and good-will at an individual engineering level held product releases together. However, under the pressure of competition and new functionality, friction was increasing between employees and locations and product quality was suffering.

Object-oriented development relies on such practices as encapsultion and information-hiding. The results are simplified, standardized clean interfaces and a structure that is easily understood. We decided to use a similar approach to implementing Scrum. Rather than implementing Scrum in full, we decided to implement Scrum by wrapping all of the management and engineering practices at each of the seven locations with Scrum practices. Each site would use iterative, incremental development. During each iteration, each site could do whatever they wanted; at the end of the iteration, each site had to demonstrate working product functionality. During each iteration, each team had daily Scrum, or status, meeting. Hierarchies of these meeting were used to coordinate the multiple teams.

Through this wrapping, common operating practices were implemented that allowed anyone and any site to inspect the progress, status, and problems of any development area within ActiveX. Through the delivery of increments of working functionality, everyone in the organization could rely on the regular presence of working code that could be inspected, interfaced to, and relied upon.

Another object-oriented practice is refactoring. As a system gets developed, the redundancies and inaccuracies are removed from the design and code. This happens incrementally rather than by relying on a static design developed at the beginning of the project. ActiveX relied on process refactoring once the various organizational sites were wrapped by Scrum practices. The wrapper provided common interfaces and external practices, similar to the idea of object methods. Once this stability was implemented, each organization began to refactor its internal practices to conform to each other. We taught the various development groups the details and philosophies of Scrum. This provided a vision for each process refactoring.

Both ActiveX and AbleX understood their business and development domains. We understood the theory, benefits, interdependencies, and agile practices. Working together, we were able to implement Scrum to deliver agile practices with minimal disruption.