

Agile Development Doesn't Have to Mean Fragile Enterprise Processes

The Move to Agile

Agile software development methodologies are garnering a lot of interest these days. While Agile methodologies started out focused on supporting relatively small teams, more and more large enterprises are looking to adopt Agile approaches for major projects or their entire organization. Application of traditional Agile methodologies as-is does not address enterprise concerns such as scalability and regulatory compliance. This paper will show that Agile practices and approaches must be blended with proven enterprise practices to achieve the right mix of agility and discipline. Scalability is achieved by leveraging a robust and unified application lifecycle management solution to automate processes and activities where possible, improve visibility and transparency across the organization, and automatically provide status reporting as a byproduct of the activities the team performs using the solution.

The reasons for the interest in Agile approaches are straightforward; traditional methodologies such as the Waterfall or V-model have a poor track record. The original CHAOS report produced by the Standish Group in 1995 identified that on average only 16% of software projects were successful; 53% were "challenged", with significant cost and schedule overruns, and 31% failed and were never completed. The average "challenged" project was 189% over budget and took 222% longer than scheduled. The majority of the projects considered for that study were developed using traditional methodologies, or no defined methodology whatsoever. Lack of user input, incomplete and changing requirements were cited as the main reasons why projects were challenged or failed. Traditional methodologies such as Waterfall and even Barry Boehm's Spiral model were predicated on the assumption of well defined requirements and little change to those requirements over the life of the project. While those assumptions are sometimes true for military systems projects, they are not typically the case for most commercial systems.

Agile methodologies came about in response to the failure of traditional methodologies to handle projects with poorly defined requirements or rapidly changing requirements. Some Agile methodologies, such as Extreme Programming (XP) or Dynamic Systems Development Method (DSDM) have been around for over a decade, and have been successfully implemented at a number of organizations. Unlike traditional methodologies, which are focused on following a plan, Agile methodologies are focused on adapting to change. Additionally, Agile is geared towards minimizing risk and wasted effort in environments where uncertainty or ambiguity exists. Thus Agile methodologies are better aligned with the conditions found in typical software development projects.

What Distinguishes Agile Methodologies?

In addition to its focus on adapting to change, Agile is characterized by having close (ideally continuous) involvement of the customer, and an emphasis on empowered, self-organizing teams. Agile methodologies are geared to producing working software with minimal overhead. Agile methodologies have been strongly influenced by the concepts of Lean Manufacturing. The focus on reducing waste (non-value adding activities), continuously improving, and adapting to change are all based on or influenced by Lean Manufacturing concepts to some extent.

The move to Agile methodologies came out of a perception that many traditionally run projects became slaves to bureaucratic processes and practices that didn't necessarily increase the quality of the final software product, or increase the chance of timely delivery. Thus Agile methodologies focus on working software as the primary measure of value to the customer.

Taking a page from Lean Manufacturing principles, the Agile approach attempts to eliminate activities and development artifacts that do not add significant value, such as overly detailed documentation.

Agile methodologies advocate a much more decentralized approach to project management than do traditional approaches, replacing a centralized “command and control” approach to project management with an emphasis on facilitation and mentoring. Rather than assigning work to the team based on outsider estimates of how much they should be able to accomplish, an Agile team has the people who will do the actual work perform the detailed estimation, and work done in an iteration is defined by what the team has committed to, not by what a manager would like accomplished. This emphasis on delegating to empowered individuals, coaching rather than managing, is one of the biggest cultural changes that organizations have to deal with when transitioning to Agile.

Another defining aspect of Agile methodologies is the desire for close, ongoing interaction with the customer. It reflects that fact that for many projects where Agile is appropriate, neither the customer nor the development team fully understands what is required at the start of the project and that even what they do understand is likely to change over the course of the project. It also reflects the desire to minimize the number of translations between customer and those who will implement what the customer wants.

The main practice supporting Agile methodologies is the use of an iterative and incremental development framework. Unlike traditional approaches such as the Waterfall model, which do the majority of planning, requirements definition and design up front, then implementation followed by testing, an iterative and incremental approach partitions a development project up into a number of iterations, each of which is strictly held to a fixed, relatively short time interval. This constraint on the duration of iterations is known as time-boxing, and is a significant part of the incremental and iterative framework.

Traditional approaches attempt to fix the scope of the project up front, and adjust resources and schedule to meet delivery requirements. Scope creep results in all three factors being variable, which plays havoc with the project plan. Agile methodologies start with the concept of an iteration which is short enough in duration that changes can be deferred until the next iteration; the schedule and team resources are fixed for

the iteration, and only the scope is available for adjustment. By maintaining a consistent duration for iterations, metrics can be collected and compared across iterations. A common metric is the throughput per iteration; in the Scrum methodology this is known as velocity and is measured as a weighted measure of implemented user stories per iteration. Velocity provides a good long-term estimate of how much can be delivered over a given set of iterations.

The iterative nature of the development framework means that as new user stories are implemented, the overall software product should be continually analyzed and refactored as needed to ensure that the quality of the code does not degrade over time and the implementation remains clean and maintainable. Software design patterns are also used to maintain the quality of the code and to increase the chance that the emergent architecture of the software product is a good one. This emphasis on maintaining cleanliness and quality of the software product as it is incrementally developed tends to require more discipline from individual developers than traditional methodologies demand.

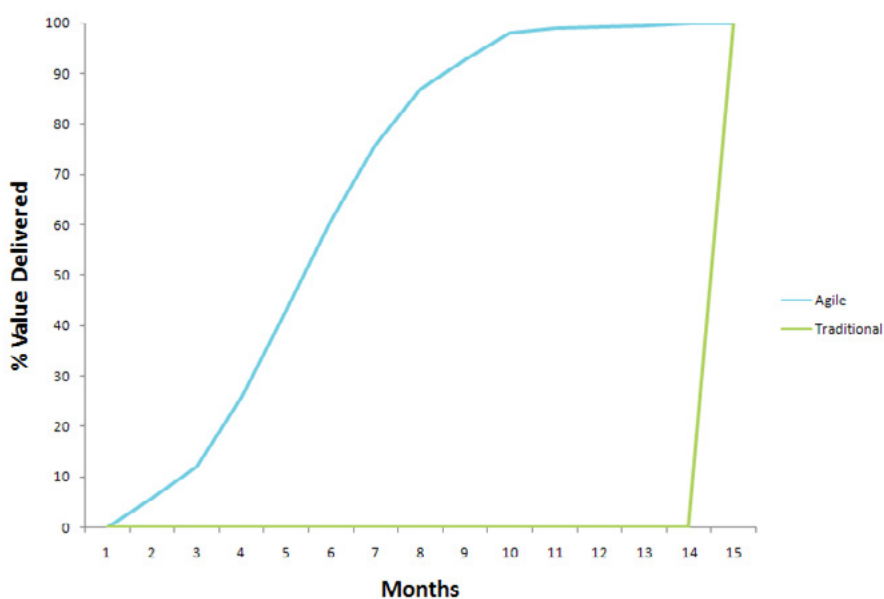
The iterative and incremental development framework is combined with a “just enough” and “just in time” approach to planning and requirements analysis. At the product level, very high level requirements are identified and defined in customer context; Scrum calls these “user stories”. User stories are defined just enough that a rough order of magnitude scoping and risk analysis can be performed. The customer or product owner (customer representative in Scrum terminology) then prioritizes this set of user stories, and the resulting queue of user stories is known as the product backlog. For a given project, referred to as a Release in Scrum, a subset of the product backlog is identified as candidates for that Release.

For a given iteration, the highest priority user stories from the backlog are analyzed and estimated in more detail. The development team then reviews each of these stories and adds them into the queue for the iteration until they feel they cannot commit to deliver any more that iteration. Work then begins on the iteration, with daily feedback across the team on how everyone is doing and if there are any issues that need to be addressed.

If the team manages to implement their committed set of user stories before the iteration ends, they can pull more from the product backlog; if they run into problems and do not complete all the stories committed to at the beginning of the iteration, the uncompleted stories go back onto the product backlog. This “pull” approach to planning the work to be done along with the “just in time” nature of analysis and scope estimation is very much based on Lean Manufacturing concepts.

At the end of the iteration there is a review where issues that impacted the iteration are identified and opportunities to improve are identified. With most Agile methodologies, such as Scrum, the goal is to have “potentially shippable” software at the end of each release. This means that integration of new work is an ongoing activity and not left as something to be done at the end of the Release. In fact another common practice in Agile methodologies is that of Continuous Integration, where every committed change triggers a build and test cycle to validate that change as soon as possible. At the end of the Release there is usually an additional review, referred to as a retrospective in Scrum, which is an overall post-mortem of the project. The successes and problems are both identified, and action items raised to resolve problems or implement suggested improvements.

The series of short, fixed length iterations also encourages regular feedback between the customer or product owner and the development team. At the end of each iteration, ideally there is a stable build of working software for the product owner to evaluate. Based on that evaluation, the product backlog can be reprioritized, the direction of product development can be redirected, or the product owner might decide the release has met its goals already and proceed with the final release rather than continue development in the next iteration. As the figure to the right shows, iterative and incremental development with working software at the end of each iteration has the potential to deliver significant value at various points over the course of a Release. Traditional development approaches, on the other hand, tend not to produce working software until the very end of the project.



Issues with Enterprise Implementations of Agile Methodologies

Agile methodologies have been shown to work well with small teams and new software product development. Agile purists recommend teams of no more than ten people who are physically co-located in order to maximize communication and minimize the overhead associated with planning and management. Agile purists are also almost totally focused on the delivery of working software, and are less concerned with demonstrating compliance to processes. Unfortunately, these conditions are rarely available or suitable in the enterprise environment.

The first issue that most enterprises have to deal with is that they have existing teams and projects that are following existing methodologies. Making radical changes mid-project to critical enterprise applications is not generally recommended. In general, most organizations want to pilot Agile methodologies – try them out on a few new projects that better fit the criteria of the Agile purists. Once the organization has proven the worth of the new approach and gained experience in its implementation, they can consider introducing it to larger teams and legacy applications. In order to make this transition as smooth as possible, organizations should consider using flexible application lifecycle management (ALM) solutions which can support both traditional and Agile methodologies

on different projects within the same organization. ALM solutions like Integrity, a PTC product, can also aid in the transition from an existing methodology to a more Agile approach by managing the change of processes and procedures.

The next issue is the matter of scaling Agile approaches to handle enterprise organizations. The typical enterprise development organization consists of far more than 10 developers, which is the recommended upper limit of an Agile development team according to the purists. Organizing such groups into teams of teams can scale the Agile approach somewhat, but even that won't handle most enterprise organizations. As the number of teams that interact grows, the coordination issues and the number of communication interfaces grow exponentially. Further adding to the problem is that most enterprise organizations are distributed across multiple sites, which rules out the physical co-location of everyone. Again, robust application lifecycle management solutions can play a significant role in addressing these scalability issues. A common repository which enables real-time access to pertinent data regarding task and defect status as well as real-time visibility into what code changes are under development can address the issues raised by having larger groups that are not co-located.

Software change and configuration management (SCCM) support for parallel development, such as first class support for branches of configurations, can reduce the disruption that would occur if everyone performed continuous integration into a single release branch. By allocating branches to separate teams or even separate feature development, continuous integration can be performed in a staged manner, so that no one branch is overwhelmed by change, which could result in excessively frequent broken builds. The SCCM functionality should also provide dedicated support for distributed development, so that latency and bandwidth limitations between sites do not affect the performance of any given team or the visibility across the organization. Automating processes where possible, particularly the collection of data and generation of status metrics, is another way that application lifecycle management solutions such as Integrity can assist with scaling of Agile methodologies.

Regulatory compliance and senior management oversight are other enterprise aspects on which Agile purists typically do not focus. Enterprise organizations often have to demonstrate that processes are defined and consistently followed. The process automation capabilities of a solution such as Integrity make it easy to define processes and enforce them, with automatic audit trails generated as to that compliance. Integrity takes it a step further with the ability to enforce and automate aspects of change management for process changes through its Admin Staging capability. Given the continuous improvement aspects of Agile approaches, plus the need to gradually migrate teams from traditional methodologies to Agile, the ability to control and automate the roll-out of changes to processes the teams follow can be very helpful.

Another aspect that Agile methodologies are traditionally weak at is requirements management. Agile purists often treat the customer requirements (user stories) as transient objects, which are disposed of once the iteration that implements them is complete. These purists claim the source code and test cases written as part of the implementation is all an organization needs to track requirements, or that they can just consult the product owner or customer for a ruling if in doubt. However, when working with complex, long-lived applications, these approaches are not adequate. New functionality must integrate with existing functionality, and the dependencies and impacts of changes need to be clearly analyzed. For complex applications with a long release history, even the customer or product owner is unlikely to clearly remember all the details of already implementing functionality. As George Santayana once said, "those who do not learn from history are doomed to repeat it". If requirements are not properly captured and managed, there is a greater chance that existing functionality becomes reimplemented with subtle and unintentional differences and the ability to determine the impact and dependencies of new changes on existing functionality will be limited. Using an application lifecycle management solution that provides strong support to capture, manage and analyze the impact of changing requirements can go a long way to incorporating Agile approaches into the development of complex applications with significant history.

Finally, the last enterprise issue to address is the need to support families of related products, or implement a Software Product Line (SPL) strategy. In such situations, certain components are common across the family of products, while others vary depending on the specific product. Often in such cases each product in the family or product line has a different customer who may not really care about reuse or commonality with other customers' products; the desire to maximize the identification and reuse of common components is driven by the enterprise to reduce their cost and time to market, not the customers. Blending a component reuse or SPL strategy with Agile methodologies requires a platform that provides first class support for managing variants of the same product line as well as sharing of common components between those variants. Without active support for managing such commonalities, the Agile approach would tend to produce unrelated products, given the focus on close collaboration with the customer for a specific product.

In Summary

There are both significant cultural challenges and technical challenges involved in implementing Agile approaches within an enterprise. Some of the cultural challenges are the de-emphasis on centrally managed development, and the delegation of significant autonomy to self-organizing teams. Examples of the technical challenges are providing visibility and transparency across multiple sites and large development organizations; supporting both traditional and Agile methodologies within the same organization; providing support for complex applications with significant history and families of related products. The indisputable benefits of Agile approaches ensures that enterprise adoption of such methodologies will continue despite these challenges.

Many of these technical challenges can be resolved or minimized through the use of a flexible application lifecycle management solution. Integrity is a powerful, flexible application lifecycle management solution that is well suited to implementing Agile as well as traditional methodologies since it provides out of the box transparency across all artifacts and activities, and real time visibility into the activities being performed and the changes being made to the software.

Integrity Business Unit Locations

North America
1 800 613 7535

United Kingdom
+44 (0) 1252 453 400

Germany
+49 (0) 711 3517 75 0

Asia Pacific
+65 6830 8338

Japan
+81 3 5422 9503

integrityinfo@ptc.com

For more information visit: PTC.com/products/integrity

© 2011, Parametric Technology Corporation (PTC). All rights reserved. Information described herein is furnished for informational use only and is subject to change without notice. The only warranties for PTC products and services are set forth in the express warranty statements accompanying such products and services and nothing herein should be construed as constituting an additional warranty. References to customer successes are based upon a single user experience and such customer's testimonial. Analyst or other forward-looking statements about PTC products and services or the markets in which PTC participates are those of the analysts themselves and PTC makes no representations as to the basis or accuracy thereof. PTC, the PTC Logo, Mathcad, Creo, Elements/Pro, and all PTC product names and logos are trademarks or registered trademarks of PTC and/or its subsidiaries in the United States and in other countries. All other product or company names are property of their respective owners. The timing of any product release, including any features or functionality, is subject to change at PTC's discretion.