AGILEACTIVIST.COM

December 21, 2006

This article was originally published on Confessions of an Agile Activist

To Be or Not to be Agile
how and why to choose agile

By Paul Klipp

For a decade now, agile processes, lead by eXtreme Programming, have been gaining wide acceptance among developers, but many customers are still in the dark. Agile sounds good, but what does it mean? This is a quick and dirty preview of what you can expect when you choose an agile process.

At it's core, an agile process is designed to address a long-standing problem with traditional development methods, scope-creep. Most traditional processes begin with a thorough description of the desired product and then code until it's done. The weakness of these approaches is that in the event of a change in the business need or a reevaluation of the plan, much work can be lost and deadlines can easily slip out of control, and costs with them. The traditional way to address this problem is with change documents. A change document basically is a way of telling the customer what it will cost to make a change to the plan after development is underway. Agile processes are designed to do away with the cost of change so that the client is free to evolve the system under construction toward the ideal end goal, even if it is a moving target.

That's a tall order and skepticism is understandable.

Here's how it works.

As the client the first thing you want to do is to verify that the project has a positive ROI. I assume that you know why you want the software, what need it will serve and what value it will provide. What you need is a cost estimate to decide whether it's worth moving forward or not. The first round of planning provides that, but keeps things very rough and flexible.

As the project manager I help customers define the top level requirements and then the developers make rough estimates of the time needed to build the core components of a release. A release in an agile process is the first planned deployable version of your product. It's the first version that provides enough value to justify using it. Here is the first major difference between agile and traditional development. Using a traditional approach, you get your software when it's all done. Using an agile

To Be or Not to Be Agile
how and why to choose agile

By Paul Klipp

approach, the team builds many working versions of the software, and you get to choose when a version offers enough value to use, so you start enjoying benefits sooner.

If, based on the rough estimates, the client decides that the ROI is positive and chooses to proceed, I begin planning the first release. A team consisting of developers, a project manager, testers, and a customer collaborate on every phase of the process. Developers provide estimates and solutions, the project manager makes sure that the project runs as efficiently as possible and everyone has the information they need to do their job, testers help to write acceptance tests and they run both acceptance tests and unit tests, but the customer has the most important job of all. The customer writes the user stories, acceptance tests, answers developer questions during the development process, and decides when an iteration (a working version) of the product provides enough value to deploy. To be clear, I should explain that I use the word client to refer to the company or individual who commissions the software product. I use customer to refer to the team member (who typically works for the client) who is responsible for these very specific duties described above.

The release plan begins with a system metaphor and user stories. The system metaphor is a simple analogy that describes how the system will work. For example, Lunar Logic Polska recently created a purchase request system for which the system metaphor was an online store. That metaphor served two purposes. It allowed the developers to make intelligent choices when confronted with vague details in user stories and it provided much of the vocabulary for the project. User stories are described in more detail in my other articles, but they are basically short statements of functionality, just a title and a sentence or two, that describes what the product does. The project manager works with the customer to create the user stories in plain language. The details are filled in as needed during development by dialogs between the developers and the customer.

Once all of the stories for a release are written (the customer can add or remove stories between any iterations), developers estimate the time required for each story. They use best case scenario estimates. By consistently using best case scenario estimates, the developers set a standard for estimating that is easy to understand and apply. If a story is too complex, developers will work with the customer to reduce it to smaller stories. For some user stories, the developers might break the function into tasks defined in development stories that are attached to the user story.

The customer then takes the stories with the estimates and prioritizes them. I like keeping the priorities simple: high, medium, and low. There might be dependencies that aren't captured yet, but that's why the customer participates in creating the iteration plan.

Using the user stories and developers' estimates, we create an release plan. We decide how many iterations to do and how long each iteration should be. An iteration is a concept that deserves more description. An iteration is an agile way to achieve two main purposes: to produce useable product fast and to keep the code simple. It also helps to adapt to change. An iteration can last from one week to three months depending on the scale of the project. I prefer to keep them to a month at most. Each iteration produces a useable product. The exciting thing about iterative development is that the customer retains the right to change direction, change teams, or pull the plug and still have value to show for their money. As I wrote in a previous article:

"The beauty of iterative development combined with continuous integration is that if we approach a piece of software with the intention of working until all features are done (traditional approach), then at no point in the project is there useable code except at the end. Whereas with iterative development, the client can pull the plug at any time and have a working product, even if not fully-featured. For example, halfway through a large, waterfall or RAD (Rapid Application Development) style project we might have had a working administrative interface but not working user interface, or no user authentication system; in other words, a fundamentally flawed and unusable product. Halfway through an agile Project (at the end of any iteration) we have a product which, though lacking some intended features, actually had all the essential components of a software tool finished, tested, and ready to deploy if desired. The customer is not married to the project and the developers can't hold the project hostage until the bitter end; If it concludes early, the investment is not a sunk cost."

Iterative development also means that there are logical places to stop, reevaluate, change stories, and change the direction of the entire project if necessary, without causing any significant disruptions to the process.

An iteration plan assigns stories to an iteration based on the resources dedicated to the

To Be or Not to Be Agile
how and why to choose agile

By Paul Klipp

task and the time limit set for the iteration in the release plan. During development, developers work through stories in order of priority, occasionally asking the customer for clarification. The first thing the developers do each day is to discuss the plan for the day and get feedback from other developers on how best to approach the day's stories. Then, each developer begins a story by first planning the work necessary to implement it, and then writing unit tests to test the desired functionality of the code they plan to write. Writing unit tests before writing code forces the developer to clearly think thorough what needs to be done and the best way to do it. It also gives him a distinct goal - write the simplest code that will pass the unit tests. As he finishes each story, he integrates the code on the integration server and runs unit tests to ensure that his update hasn't broken any other functionality. When a story is complete, the developer notes the time taken to complete the story (we use this to refine our velocity calculation) and announces to the team that the story is complete, attaching a screen shot or video that illustrates this new function. The customer is welcome to offer immediate feedback so that story functionality can be tweaked early on, while the code is still fresh in the developer's mind.

As work progresses, I track actual time against estimates and thereby arrive at the team's "velocity." Knowing the velocity of a team on a project allows me to evaluate their future capabilities rather than second-guessing the estimates as would happen in a traditional method.

Meanwhile, the tester is working with the customer to write acceptance tests. More about this component of the customer's job can be found in my articles regarding the role of the customer. The tester is responsible for leading the team of QA testers who run acceptance tests on closed stories and verify that unit tests are passing. They produce regular reports of test failures that the developers use to refactor as they progress.

At the end of the iteration, which takes place at precisely the predetermined time, the testers run unit tests and acceptance tests and when everything passes, the working product is tagged in cvs. The customer now can decide whether to deploy the product of this iteration or to wait.

The iteration planning cycle then repeats. The customer can add or reprioritize stories and then a new iteration plan emerges. It is not at all uncommon for customers to have a much clearer idea of what they want as they see a project evolve. Iterative development accounts for this fact and gets working software into users' hands as fast as possible so that course correction is not delayed.

This has been a very brief overview of an agile process, but hopefully it illustrates how by using agile techniques, companies like Lunar Logic Polska are able to fulfill for their customers the ideals enshrined in the "Customer Bill of Rights" (http://www.c2.com/cgi/Wiki?CustomerBillOfRights):

Customer Bill of Rights

- ☑ You have the right to a plan, and to know what can be accomplished, when, and at what cost.
- ☑ You have a right to get the most value for each programming week.
- ☑ You have the right to see the progress in a running system, proven to work by passing tests that you specify.
- ☑ You have a right to change your mind, to change functionality, and to change priorities without incurring exorbitant costs.
- ☑ You have a right to be informed of schedule changes in time to choose how to reduce scope to restore the original date. You can cancel at any time and be left with a useful working system that reflects your investment to date.

If you know exactly what you want, and you know your needs won't change, and you are prepared to work with a software vendor to create a comprehensive document describing every aspect of the software you want created, and you are willing to commit to not making any significant changes in plan, and you trust your vendor to deliver exactly what you expect on time, then an agile process may not be right for you. If any of these statements do not apply to your project, then nothing will mitigate your risk and improve your level of control over your software project like a well-implemented agile process.