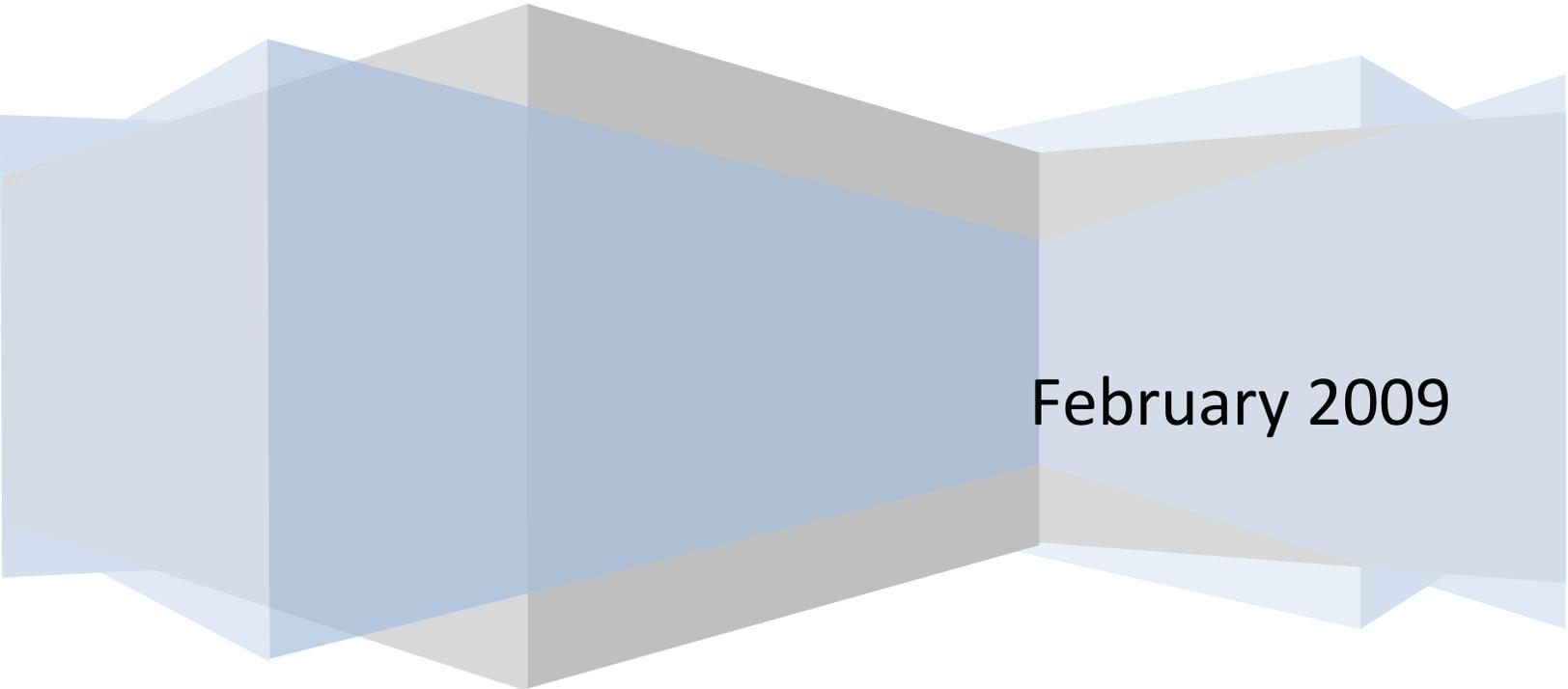
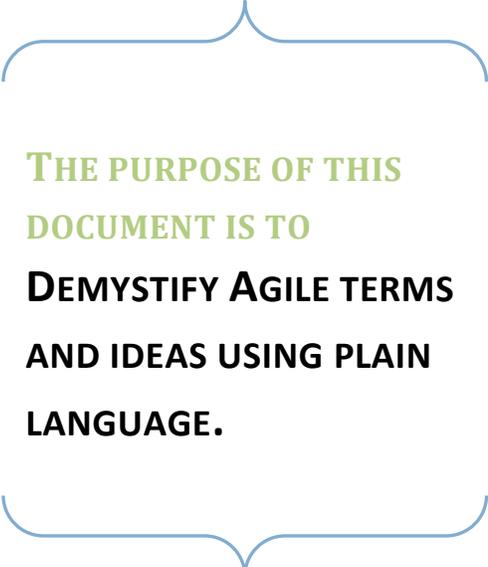


# What Every Business Person Needs To Know About Agile Software Development

Audrey R. Troutt



February 2009



# IS THIS GUIDE FOR YOU?

**THE PURPOSE OF THIS DOCUMENT IS TO DEMYSTIFY AGILE TERMS AND IDEAS USING PLAIN LANGUAGE.**

If you are reading this, then your business needs software to succeed.

Maybe you are wondering if you need to try something new to grow your business and stay competitive in your industry.

You know your market. You know your business. You respect the competition. You understand what your customers want. I bet you even know what your software needs to do to succeed. So why is it so hard? A lot seems to do with *how*.

How you approach software development—from listening to the first wish list to launching the final product—differs greatly between practices. This is especially true for Agile.

You've tripped over the jargon, the blogs and the books. Do you want to know what everyone is *really* talking about? Do you want to know the terms in plain language? If so, then this guide is for you!

# WHAT IS YOUR GOAL?

Let's start with the most important question: what is the business problem your project is trying to solve? What needs to be different about the way you usually work for this next project to be more successful? Here are some common answers:

- ✓ Technical excellence/increased quality
- ✓ Control/visibility into the process
- ✓ Accelerated time to market and flexibility
- ✓ Reduced costs
- ✓ Increased end-user satisfaction/adoption
- ✓ Repeatable/scalable process

All of the Agile practices we are going to talk about are frequently prescribed to gain one or more of these benefits. Not all Agile practices are going to be equally helpful in your context and none of them are magic bullets.

If you let me help you clear the technological fog, I promise you can better understand Agile practices and if they might work for you.

For each of the benefits above I will break down the most common Agile practices advertised for achieving them. Along the way I will introduce you to the major Agile movements, call out some helpful tips, point out some common speed bumps and give you some direct questions to ask your development team or custom software providers.

## TECHNICAL EXCELLENCE/INCREASED QUALITY

Quality and technical excellence require talented and dedicated teams. Yet we all know about projects that went astray even with a great group of highly skilled developers, and we all know of great successes pulled off by otherwise average folks. Who is on the team matters. How the team approaches its work may matter even more. I'm not going to sugarcoat this—process is important.

Quality is one of the areas in which some worry Agile can hurt more than help. It is wise to get to know your development team and the processes they use now to ensure that your quality and technical precision requirements are met. Then, you can better see if Agile practices can take you to the next level. How, you ask? Here are two commonly prescribed methods:

**Pair programming** - Pair programming is the practice of putting two developers at a computer together, working on the same task. The idea is that at any time one is typing and the other is watching for mistakes and thinking about the next step. I know what you are thinking “yeah, better quality, maybe, but at twice the cost, definitely! Why bother?” Well, all but the most hardcore pair programming devotees will tell you that not all development tasks lend themselves to pairing or greatly benefit from two pairs of eyes instead of one. On the other hand, there are many programming tasks, particularly more complex assignments, that definitely do benefit from pairing.

The developers at the Menlo Institute wrote about why they use pair programming.<sup>1</sup> They found that pair programming can reduce the number of bugs and increases compliance with code quality standards among other benefits that I'll mention below under reducing costs.

**Test-driven development** - Test-driven (often referred to as TDD) or test-first development is the practice of writing tests before writing any other code. This practice requires a serious amount of discipline on the part of the developer. The idea is that writing your tests first makes you write more testable, readable and robust code—that means it will be easier to build on it and add more tests in the future and there will be fewer bugs. Plus, you build up a large suite of tests that can be run every time a new feature is added so that you can be confident that what works today will work tomorrow. If your software has to be very precise then you will want to have a lot of tests around it, but be aware that not all tests are good tests.

**Ask your vendor!** How do you measure code quality and ensure technical precision? What is your testing strategy? It is not unreasonable to expect your vendor to have a well-practiced and documented process for quality control.

---

<sup>1</sup> Check out the Menlo Institute paper, “Paired Programming in the Software Factory,” by Thomas Meloche, James Goebel and Richard Sheridan at [http://www.menloinnovations.com/freestuff/whitepapers/paired\\_programming\\_q\\_and\\_\\_a.pdf](http://www.menloinnovations.com/freestuff/whitepapers/paired_programming_q_and__a.pdf)

## CONTROL/VISIBILITY INTO THE PROCESS

Visibility and control are important so that you can ensure that everybody is rowing together toward success. This is a hot topic for Agile because on one side Agile is all about visibility and dynamic steering, and on the other side managers sometimes feel like Agile takes away their control over the scope of the project. What kind of Agile terms do you come across when you talk about control and visibility? Here they are:

**Iterative development** – Iterative development is working on software in short cycles (one to four weeks, usually) at the end of which the software is in a potentially-shippable state, that means fully tested with all of the completed features in place. Iterative development usually entails completing vertical slices of functionality, in other words, all the code that is needed for the back-end processing and storage as well as everything for the front-end user interface so that you, the business owner, can touch and try the new feature for yourself at the end of the iteration. How does this give you control and visibility? You get to see and touch the software that your team has been working on and you get a chance to modify the plans for the next iteration if needed.

**Iterative development** is common in the design phase of product development in a manufacturing setting where designers come up with a design, make a mock-up, gather feedback (and criticism) and then do it again until they get to the design that they believe is going to succeed in the market.

Here's a construction metaphor that may help: You start with a

sketch of what you want your house to look like and then iterative development would mean completing the kitchen first (roof, walls, foundation and plumbing, paint, plus all the fixtures) in the first two weeks, then a complete living room built off of the kitchen by the end of the next two weeks, and so on until you have a whole house.

The idea is that you can start living in the house from the day the kitchen is finished; you can see if you like it, change the plans if needed and start using it for your business. Later on you can decide you want more cabinet space and add on to the kitchen if you like---this is your house! At the start of each two week cycle you can change the plan if needed based on what you have seen to better suit your needs.

## FLAVORS OF AGILE:

**Scrum**, maybe the most popular flavor of Agile, includes the idea of an iteration of work, also known as a sprint (check out the definition of iterative development on this page). At the beginning of the iteration the team commits to completing a number of tasks from the top of the backlog. At the end of each sprint, the software is shippable, meaning it is fully tested and includes all of the functionality completed to date. Scrum has emerged as the most popular kind of Agile software development, but some criticize it for being incomplete. For this reason a lot of teams use Scrum and XP together.

That is all about control, but how about visibility? Here are some tools from Scrum that help to make information about your project visible:

**Ask your vendor!** *How much design do you do ahead of development?* Iterative development makes a lot of business folks nervous because the architects of your software won't work out all the details ahead of time—how do you know the whole thing won't come crashing down as soon as it's done? Iterative development does not mean there is no architecture and design. An experienced iterative development team will be able to balance design with getting things done.

**Scrum board (information radiators)** - A scrum board is a task board on which all planned, active and completed tasks and stories for a sprint are visible. For co-located teams the board is often a cork board or whiteboard and individual tasks are written out on index cards. For distributed teams there are a lot of tools out there that provide virtual scrum boards and facilitate the same visibility into the progress of the team. When a developer starts or finishes a task they walk up to the scrum board and move the index card to the “work in progress” or “pending stand up” column. (What is a stand up? See below). If something needs to get done in order for the team to meet their goal for the iteration (cycle of work, see above) then that too is put up on the board. At Ternary we are small enough that we all know what is going on so we have a “blocked” token that gets pinned to work that is blocked by someone or something else. For a bigger team it might be helpful to post more information about the block. Anytime you walk by the scrum board you should be able to tell at a glance who is working on what and how much work has been completed so far in the iteration. With this information you should be able to tell if the team is on track to meet their goals.

Scrum boards are sometimes called information radiators because they are always there giving out information. Scrum calls for the use of information radiators to ensure that everyone who wants or needs to know what is going on, what is done and what is waiting and why.

**Daily stand-up meetings** - A 15-minute meeting for the development team in which each person says what they have done since the last meeting, what they plan to do before the next meeting and what if anything is keeping them from getting their work done. The developers typically stand to keep the meeting at 15 minutes or less. There is always someone at the meeting that is responsible for keeping track of the team's progress and removing whatever obstacles the team mentions during the meeting. In Scrum this person is called a Scrum Master. The Scrum Master is sort of like a project manager, although the debate over differences and similarities between the two is fodder enough for a whole new paper.

**Retrospectives** – A retrospective is a meeting at the end of each iteration (short cycle of work) in which the team reflects back on the iteration. The goal is to identify emerging patterns (good or bad) and come up with ideas to improve teamwork and progress for the next week. This is the teams chance to bring up (make visible!) problems with *how* things are getting done, giving the team a chance to tweak the process.

## ACCELERATED TIME TO MARKET AND FLEXIBILITY

When your success depends on being the first to market, you need your software team to be able to deliver. Pushing people to work faster will only get you so far. Agile is all about rapid delivery without running your team into the ground. Many Agile practices specifically address flexibility, which is important when your market's demands are constantly changing. The key agile practice that is most often tied to flexibility and accelerating time to market is iterative development; go back to page 4 if you missed my explanation of that one. There is one other agile tool that helps you stay flexible and figure out how to get the product you need when you need it: the product backlog.

**Backlog** - at its simplest a backlog is a prioritized list of remaining work to be done. This is different from a requirements list in that the features are described from a user's point of view in practical, results-oriented terms. They can be effective with much less depth/detail than a formal requirements document. At the beginning of a project the team works with the business to generate a list of high-level user stories. They are "high-level" because they do not include detailed requirements, but instead a description and a summary of the business value the feature will add. The backlog contains features that the user will see as well as technical tasks like "set up the test server." Over the course of the project the backlog is re-prioritized so that the team always works on the most valuable feature next. Along with iterative development this means you can get the most valuable parts of your solution out to your customers as soon as possible.

## FLAVORS OF AGILE:

### Lean Software Development

Lean is yet another common take on Agile. It was inspired by lean manufacturing, which Toyota made famous, making quality products and responding rapidly to market demands. Lean's seven mantras are:

- Eliminate waste
- Amplify learning
- Decide as late as possible
- Deliver as fast as possible
- Empower the team
- Build integrity in
- See the whole

The term Lean Software Development originated in a book by the same name, written by Mary Poppendieck and Tom Poppendieck. It's a good read if you are interested in learning more.

## REDUCED COSTS

There is no getting around it: custom software is expensive. Off-the-shelf software is relatively cheap to buy, but maybe a million other people help offset that cost and you don't get to decide what it does. So you need custom software, but how can you keep costs down?

If you agree that excess costs come from writing code that isn't used and fixing bugs that should never have gotten into the code in the first place, then there are a few Agile practices that claim to help:

**Backlog** – I know I already defined this one, but it is also often mentioned as a way to avoid waste. As I mentioned above, the backlog is a prioritized list of work to be done in the form of user-centered descriptions. Over the course of the project the backlog is re-prioritized so that the team always works on the most valuable feature next, this means that features you might not even need become apparent after you put them at the bottom of the list again and again. If something costs you a lot of money, but it doesn't gain you anything, then why do it, right? That is how the prioritized backlog can reduce project costs.

***Here's a tip!** Always put defects at the top of the backlog so they get worked on sooner rather than later. This will save you money because the fresher the code is in the programmers mind the easier it is for them to fix it.*

**Pair programming** - Again, I know what you're thinking: "Putting two expensive developers on the same task at the same time instead of just one is supposed to save me money?" It sounds crazy and as I mentioned, pair programming is not always the most cost effective way to work, like when you are tweaking the size of buttons. But sometimes it is cost effective. Remember I mentioned that Menlo article before? They found that pairing reduces the number of bugs, increases compliance with code standards, keeps programmers from wasting time on distractions (email, etc.), reduces the amount of "stuck time," produces more robust solutions, increases the "bus number" (Jargon: The bus number is the number of team members that would have to get hit by a bus before work on your project could no longer continue because vital project-specific knowledge would be lost forever), and makes the team more flexible and easier to add to. Bugs are expensive. They annoy your users and can cost dearly in terms of lost credibility with customers.

The fewer bugs that make it to the final version of your product the more money you save in lost business, re-work and reputation. It takes developers longer to fix bugs in code that they have not looked at for a long time—catch those bugs when the code is fresh in their minds and save money! Pair programming is a way to catch more bugs. And what about distractions and stuck time? You can lay down as many rules as you like, but programmers are human beings: they get distracted and they get stuck. Speaking from personal experience, you are not going to check your email ten times an hour if you have a pair partner sitting next to you. Think about how much that costs you. It isn't like your pair partner is there to babysit you, but you don't even think about checking your email. Same thing with "stuck time" sometimes your brain gets stuck on something very silly, and all it takes is another pair of eyes to come along and point out that you have the answer sitting right under your nose.

**Ask your vendor!** *How do you decide when to use pair programming?* Pair programming is very useful, but we have found that some tasks make more sense for one person than for two. For example, want to rename some fields in your web application? One person. Adding a credit card payment module to your web app? Better make it two. Just as wasteful as pairing when it isn't needed is not pairing when it is needed. Sometimes programmers don't want to pair, but if they are letting too many defects through or wasting time on distractions that could cost you a lot of money. A hard rule for or against pairing is a sure sign that your vendor does not have your success as a top priority.

**Continuous Integration** – Continuous integration is the frequent submission of newly written code into the common code base from all team members working on a project. Here I am using the word **submission** to mean the submitting of code into source control. **Source control** is software that is responsible for maintaining the one official copy of your software's source code. The common code base is that official copy that is in source control. Agile teams usually use continuous integration along with automated testing and building so that not only are your programmers constantly ensuring that their changes are pushed out to everyone else, but that they don't break any existing parts of the software. Not having to deal with merging code together saves you a lot of time (and money!).

**Ask your vendor!** *Do you use continuous integration and automated tests? How do you manage source control?* Any good vendor will have well-established practices for managing source code and integrating changes so that programmers can spend their time doing what they do best—creating value for your business!

**Automated testing and automated building** - Basically, the entire software product is tested and compiled (if the tests pass!) every time a programmer checks in changes to source control. Related to continuous integration, this means there is always a working, tested copy of your software available. This also means that the team often finds out right away if they introduce a bug, which makes it cheaper to fix.

## INCREASED END-USER SATISFACTION

Your business needs happy customers to succeed. Which common Agile practices are used to increase end-user satisfaction? There are Agile practices that can help in this arena:

**User Story** - This is a description of a feature of your software from the perspective of an end-user. User stories are what make up the backlog. There is a very specific format used for user stories: "As [name of end-user] I want [simple description of functionality] so that I have [business value]". Here is an example of a user story for this whitepaper: "As a business leader I want definitions of all these Agile terms I am hearing so that I can converse confidently with software development vendors about my options".

**Product owner** – The product owner is a member of the software development team who is responsible for making decisions about what the software should do and look like. This is someone who understands your business and knows your customers and your market. The product owner is the one that prioritizes the backlog. As the voice of the end-user, the product owner is a resource for the team when they need to make a design decision or understand a requirement. In traditional software development methodologies a lot of the information the product owner gives on a day-to-day basis might have been captured up front in the product requirements documentation.

### FLAVORS OF AGILE:

#### **XP or Extreme Programming**

XP was designed for dealing with vague or rapidly changing requirements using techniques like pair programming, test-driven development, user stories and continuous integration. They call it extreme because it was originally so radically different from how software was developed and because of the discipline required to carry it out. The key book on the topic is *Extreme Programming Explained* by Kent Beck.

## REPEATABLE/SCALABLE PROCESS

You need your development team to be able to deliver consistently. Who wants to be embarrassed by late/broken/wrong delivery when you have a client in for a demo? All of the Agile practices that are frequently linked to repeatable processes have already been defined: continuous integration, automated builds and source control. Once the groundwork is laid for those three practices together then you know you can always get a working version of your software with all the latest features.

And sure, you only need two developers now, but what happens when your new software brings in 50% more business? 200% more? 1000% more? Then you might need 20 developers or 200. How is your process going to scale? If your goal is to create a scalable solution, then these Agile practices might help:

**Continuous Integration, Automated Testing and Building and Source control** - You've heard about these before, but how will they help you scale? The idea is that having an official version of your code and an easy way to add to it (and to make sure you haven't broken it) together makes it easy to add developers to your team. It also makes it easy to scale back for a time when necessary: you always know where to find the latest code when you ramp up again. At the core these are just good development practices whether you work iteratively or not, and anything is better than ad hoc source control and testing.

**Pair programming** - I'll mention this again too because pair programming is often championed for how it makes it easy to add new programmers to your team; once your team is used to working together effectively you can take a new programmer and pair them with an old one and dramatically decrease the time until they become productive members of the team. This is not a guaranteed, but that is the idea.

# WHAT NOW?

That is the end of the tour. Hopefully, with this guide in hand you feel a little more at ease navigating the jungle of Agile jargon. I've given you some ideas and questions to take to your development team or custom software vendors to talk about your next steps.

Anything else about custom software development or Agile that you are curious about? To talk about your business challenges with the best, most jargon-free COO in the industry, send an email to Joanne McCool: [joanne@ternarysoftware.com](mailto:joanne@ternarysoftware.com)



This work is licensed under the Creative Commons Attribution-Noncommercial 3.0 Unported License. For any reuse or distribution you must attribute the content of this document to Ternary Software, Inc. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/3.0/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.