

The Primary Benefits of Agile Development

You may have some skepticism about Agile. That's perfectly understandable. This book assumes that you take nothing for granted about Agile and want to know exactly how and why the various aspects of Agile are a good idea. Let's start by explicitly stating the exact benefits that you will get from implementing Agile as described in this book.

For the moment, let's assume that the claims are absolutely true and will be fully substantiated later. Whether they are true or not, if the claims have no relevance to your situation or you are not in need of any of the benefits, then there is no point in reading any further. On the other hand, if the benefits to you personally are compelling, then why stop now? If the benefits seem appealing, but something tells you that it goes against everything you believe in, consider that it is just possible that a change of perspective may be in order.

What is Agile Development?

Giving a concise definition of Agile is far from easy, probably because Agile is actually an umbrella for a wide variety of methodologies and because Agile is officially defined as the four values in the Agile Manifesto:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

But how would you translate this into a dictionary definition? It may be pointless to define Agile development. It may be better to just talk about the concepts and the practices and say "these things are worth learning about regardless of what you call it or how you define the overarching concept." I still think it is worth a shot so as to establish a starting point for discussion. So, let's discuss the definition of Agile a bit and then move on to the nitty-gritty.

Because there are so many practices associated with Agile development, a simpler way to define Agile is to do it in terms of the benefits. It is not a perfect way to define something, but it is currently the best way that I know. There would be no point in doing Agile development unless it was better, so I define the benefits in relation to traditional development. I consider waterfall to be part of traditional development, but I use the phrase "traditional development" instead of waterfall because there are many shops which would say "we're not doing Agile, but we're not waterfall either."

Defining Agile via Benefits

I define Agile development as that which, in comparison to traditional development, provides the benefits of more flexibility, higher ROI, faster realization of ROI, higher quality, higher visibility, and

sustainable pace. Let's learn more about these benefits and the differences between traditional development and Agile development by looking at an example development scenario.

Six Features in Six Months

Let's say you want to add new features to stay competitive. In a traditional project, you have a known timeframe for major releases. Too short and you'll spend too much time on overhead, too long and you'll miss opportunities. For the sake of argument, let's pick a timeframe of six months and say that allows you to provide six "big features" including time for all preparation and testing. Marketing says the six features with the highest ROI are a Facebook plug-in, a Second Life plug-in, an RSS feed plug-in, and three other features. During the six months, the business value of the planned features may change. If it goes up, that's great, but if it goes down, there's very little you can do about it.

Midway through the design process, marketing announces that the Second Life plug-in is not as marketable as they had hoped and iPhone support is showing signs of becoming very lucrative. You think, "oh well, nothing we can do about that now."

Just after you finish coding, marketing declares that the Second Life plug-in is going to be a complete flop and wants to know when can they get iPhone support?

Now that the functionality has settled down, QA finishes their test plan, and starts writing test cases and running them. Planning and development took longer than expected. Originally, there was a month reserved at the end for testing, but now the release deadline is looming with just two weeks left in the schedule. The time for testing is compressed, QA concentrates on the most critical stuff and gives the rest a spot check. Once the find/fix rate gets down to an acceptable level, you declare victory and deliver the new release. In the end, the functionality that was asked for at the beginning is delivered a month late and doesn't have the originally anticipated value.

Problems with Quality

In a traditional project, the elapsed time from start to finish from the perspective of any individual work item is very long. There is no consistent process applied to each and every work item. Instead, work items are fused together into "the release" and the quality of "the release" is measured. One consequence of this is that QA gets a big dump of functionality near the end of a release and has to figure out how to make the best use of the time remaining which is often less than originally planned. That means taking shortcuts. The "most important" new features get very thorough testing and the rest get a "spot check." Another problem with traditional projects is that since much of the test case writing, test automation, and test plan execution is left to the end, problems can hide out until just before the release and thus there is a long time between the introduction of a problem and the detection and fixing of that problem.

The Misperception of the Value of Traditional Testing

Here's a mystery. If running through your full test plan takes two days, why does testing take a month? The answer is that you aren't really doing a month's worth of testing, you are doing the same testing over and over while a month of time goes by. Problems have been creeping in all along the way that you

are just now finding out about and it takes many test/fix cycles including repeating some or all of that test plan over and over again to expose and fix the problems. It is only the final full run of your test plan that gives you the measure of the quality of the product, so in the end you've really only gotten the value of the test plan. If it takes two days for that final run, then you have two days of testing, not a month.

Low Visibility

Time after time, with traditional development, progress is measured based on progress against a plan. The problem with that is that customers don't buy Gantt charts, they buy shippable software and the progress that is measured by Gantt charts is not directly connected to shippable software. Just because you have finished 100% of the requirements which is 20% of your plan, that doesn't mean you are 20% done. In fact you are 0% done from the perspective of the customer because they can't benefit from that progress until you ship. In a traditional project, you only know how much ahead or behind plan you seem to be based on the progress that people claim, but you don't really know how much ahead or behind plan you actually are from an "is this shippable" perspective. It is almost impossible to see at a glance what the real project status and progress are. You know that you won't start getting information about where you really are until sometime after code freeze.

It is hard to know what to do next if you aren't sure where you are. Do you feel like you know exactly where you are and what to do next at every step of the process? Or do you feel pulled in a million different directions at once and lose track of what you've already done and what you need to do next? If you do feel like you've finally "got" how software is developed in your organization, how long did it take you to get to that point?

The Agile Approach

Now let's try Agile development using the same scenario. Marketing says the three features with the highest ROI are a Facebook plug-in, a Second Life plug-in, an RSS feed plug-in, and three other features. You start with the Facebook plug-in. You plan. You design. You create a test plan with test cases while the code is being written. You discover potential problems and deal with them. You automate the test case while the code is being written. As the code is written, it is integrated, built and tested continuously; problems are found and fixed immediately. At the end of development, the only problems that remain are the ones that could only be found at the end of development. If you wanted to, you could cut a new release with the Facebook plug-in with very little overhead.

Agile Allows for More Flexibility and Options

With Agile development, you develop your software in short iterations where "short" means a month or less. At the end of each iteration you have a new increment of functionality which is shippable. That means that at the end of each iteration you have the option to easily change plans before the beginning of the next iteration in order to take into account new business requirements or information gained during the previous iteration.

At this same point in time, the traditional project would just be finishing up their preparations for starting development and would not yet have anything to show for their progress other than plans. With traditional development the software looks like a construction site until just before the release: some parts are done, others are not, but the building is not currently usable at all. Changing plans during the development cycle means ripping out work in progress and possibly work that is done but depends on other work that is not done.

In the Agile scenario, since there is nothing in progress, the organization can now reevaluate which work will produce the most business value. Marketing determines that the Second Life plug-in is not as marketable as they had hoped and iPhone support is showing signs of becoming very lucrative, so they tell engineering that the new ranking of features is RSS feed support, iPhone support, the Second Life plug-in, and three other features. Since RSS feed support is ranked the highest, you start on the RSS feed support.

Now marketing says the Second Life plug-in is worthless but iPhone support is hot. That means that the new ranking of features is RSS feed support (which you are working on), iPhone support, three other features, and then the Second Life plug-in. So, you finish the RSS feed support and then work on providing iPhone support. You continue this process indefinitely.

Agile Provides Faster Realization of ROI

Since you have shippable software at the end of every iteration you can start realizing the ROI whenever you like instead of having to wait.

Agile Delivers the Highest Business Value and ROI

Because Agile gives you more flexibility you can change your plans to take into account the current market conditions instead of working on things which seemed like a good idea when you started working on the release but have now lost their appeal.

In the end, the traditional team produced a release which had less business value than originally expected, and they missed out on the opportunity to deliver iPhone support early which turned out to have very high business value. The Agile team on the other hand produced more business value than was originally expected.

Agile Produces Higher Quality

In an Agile project, the elapsed time from start to finish from the perspective of any individual work item is very short. Test case writing, test automation, and test plan execution are done throughout the iteration and each work item is given the amount of QA resources that is appropriate for it. Because problems are found and fixed faster, there is less chance of the quality of a project being both unknown and poor for long stretches of time. Code for each iteration is written on the stable base of the previous iteration and is more likely to be stable itself because there will be accurate and timely feedback on the results of the changes.

Agile Gives High Visibility

Short iterations solve the problem of misleading progress reports. With short iterations, you know at the end of each iteration exactly how much progress you have made. Instead of the traditional "we're still on target" all the way up to just before the end of the release, you know at the end of every iteration exactly where you are. Whatever work was done during that iteration is done and potentially shippable. You know exactly how many work items were fully completed, how many weren't started, how many test cases remain to be written and automated, how many test failures remain to be fixed, and how many defects were filed on work done during the iteration. You get the kind of information that is usually only available just prior to release on a monthly or even weekly basis.

In a release plan consisting of six iterations, each iteration is $1/6^{\text{th}}$ of your overall plan. When you've finished your first iteration, you know you are not only $1/6^{\text{th}}$ of the way through your plan, you also have $1/6^{\text{th}}$ of the work actually done and shippable. When you've finished three iterations you are now 50% of the way through your plan and you also have 50% of the work done.

It may well be that you encounter a problem. Let's say that in your fourth iteration you run into a major problem and you only get $1/2$ of the work done that you planned for that iteration. Well, at least that work is done and you know for sure that you are now both $7/12^{\text{ths}}$ of the way through your plan and you have $7/12^{\text{ths}}$ of the work done and shippable with a 2 week slip in plan.

You know immediately that you are behind schedule and you can respond immediately instead of finding out near the end and having a much shorter runway to respond.

Sustainable Pace: Supply and Demand

People are much more productive and much less prone to error when they work at a constant and sustainable pace. This also means that when there are unforeseen circumstances, people are more likely to be able to respond well.

In a traditional project, the demand for resources from the four major aspects of software development see-saws dramatically over the course of a project. These aspects are project management and planning, architecture and design, development, and QA. You need resources on hand to serve the peak demand level, but during periods of low demand those resources will either be idle or used for activities which have a lower ROI.

A common circumstance is that there are insufficient resources on hand for the peak demand level and so people end up working in "crunch mode." During crunch time, people tend to make more mistakes. Agile levels demand out over time and removes this see-saw effect which simplifies resource planning and removes the need for crunch time.

With traditional development, delays during development compress most of the testing to the end of the process which then requires taking shortcuts due to schedule pressure. I used to think that one way of compensating for insufficient QA resources was to delay the release until QA finishes. On the surface it seems to make sense. But only if the folks writing code sit on their hands while QA does their work.

Ok, so you have multiple projects and the developers work on another project. But then they finish that. Now QA starts on the second project and the developers move to the third. The problem is still there.

On the other hand, as a result of the need for increased QA resources during testing, you may have two other problems. If you have enough QA resources to handle the pressure of the endgame, you may have too many QA resources during the rest of your development cycle. Alternatively, you may bring on additional QA resources on a short-term basis to compensate. Both of these options are obviously undesirable.

There's a natural balance between the amount of effort required for developing something and the amount of effort required to QA it. No matter what you do, if you have the wrong ratio of development resources to QA resources, it will cause problems. If development creates more than QA can absorb, you will create a backlog of QA work that will always grow.

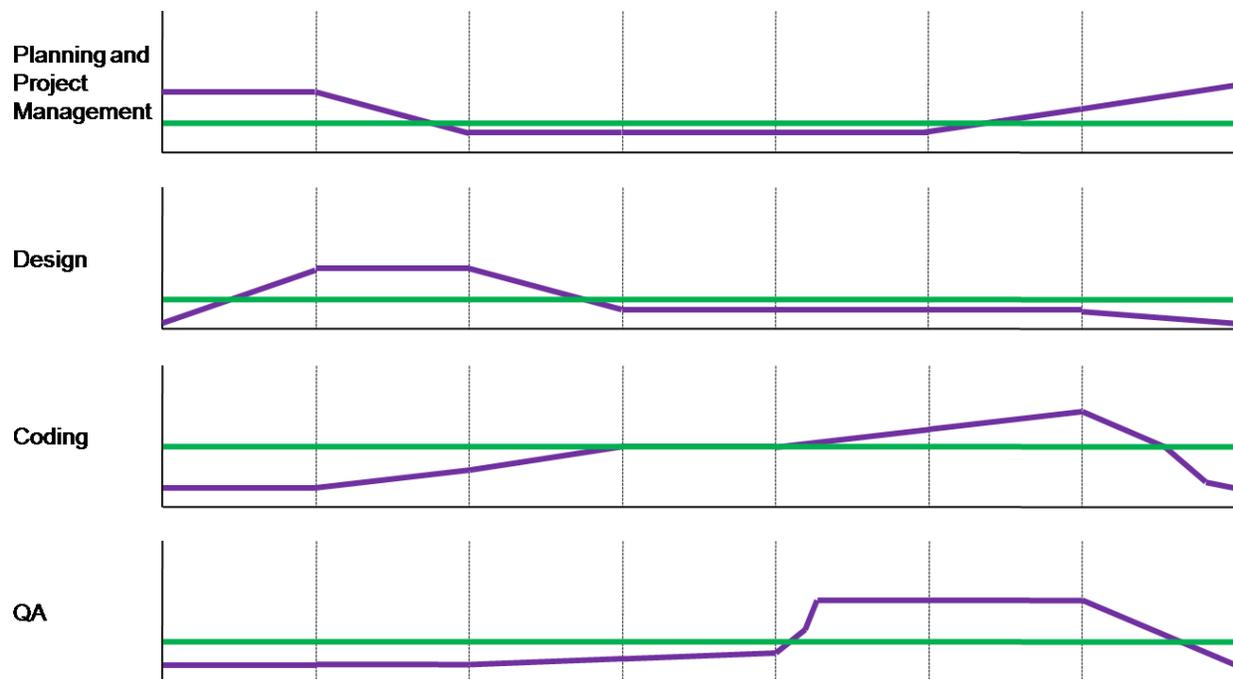


Figure: the straight green lines represent resource consumption on an Agile project. The purple lines represent a traditional project.

This natural balance holds between all four aspects of software development. Depending on your organization, there may be an imbalance between supply and demand at any stage in the pipeline.

When using short iterations, resource imbalances are easier to detect and correct. Having balanced resources means that all development activities are done consistently and on a regular basis and there is no need to take the shortcuts that are typical of traditional development.

Primary vs. Secondary Benefits of Agile

Agile is generally talked about as a single package: if you adhere to the principles, you get the benefits. There is another way to look at Agile. On the one hand, Agile introduces a whole new set of practices to the development toolkit. These practices include: product backlog, pair programming, on-site customer, continuous integration, refactoring, test driven development (TDD) and many others. While all of these practices have been either associated with or created as a result of Agile, their application and resulting benefits can be applied completely independent of any specific methodology.

Whether you use Scrum or "Waterfall" you still get the benefits of the Agile practices. For instance, refactoring is a standard feature in Eclipse. Eclipse is an IDE and is completely orthogonal to your methodology. The benefits from these practices are secondary benefits when practicing Agile.

The primary benefits of more flexibility, higher ROI, faster realization of ROI, higher quality, higher visibility, and sustainable pace come from the single practice of short iterations.

Many people object to some of the Agile practices without realizing that they don't need to adopt them to get the primary benefits. You don't have to do pair programming, use 3x5 cards, collocate, only use small teams, or have a customer on site to get to short iterations. Depending on your circumstances, these techniques can be used to accelerate and reinforce your transition to Agile, but they are not the only practices that will. For instance, test driven development is very easy to adopt, contributes greatly to enabling short iterations and does not require any experimentation with personal boundaries like pair programming does.

Once you have successfully transitioned to short iterations and are receiving the primary benefits, there are many things you can do to fine-tune your process and continually increase the benefits of Agile development. Depending on your circumstances that may mean incorporating pair programming, working at the same site as the customer, collocating, using self-organizing teams, etc.

Why Bother Changing?

Aside from personal preference, the only reason to make a change to the way you develop software is to realize a benefit. It could be to increase quality, customer satisfaction, employee satisfaction, productivity, or profits. In the end, these should all result in increased profits. If profits are not an important part of your organization, for instance you work at a not-for-profit, then another way to look at this is reducing expenses. For simplicity, I will focus on profits.

Why should you care about the profitability of your company? There are a number of reasons. If you are a stockholder, you benefit directly. The more profitable a company is, the more likely it is to be secure. The more profitable a company is, the more likely it is to embark on exciting new opportunities which means more opportunities for you.

Agile Profits

In 2006, Litle & Co. landed at No. 1 on the Inc. 500 list with \$34.8 million in 2005 revenue and three-year growth of 5,629.1 percent. In 2007 Inc. magazine cited Litle & Co.'s 2006 revenue as \$60.2 million, representing a three-year growth rate of 897.6% over its 2003 revenue of \$6.0 million. How has Litle achieved these impressive results? One factor that they site is their use of Agile development.

Another reason to care about the profitability of your company is because your company is more likely to invest in its development infrastructure. If you were smirking while reading the previous sentence because you know that extra profits will never be invested into the development organization, perhaps you are working for the wrong company. Or perhaps nobody ever thought of investing more into the development organization and a suggestion or two in the right place is all that is needed. In either case, you can still take to heart the idea of re-investing profits into the development engine and work on strategies to make it happen.

Focus on Value

At Litle & Co., each developer has a quad-core workstation with 2GB of memory and a fast disk. They know that the less time developers spend waiting for build and test results or switching back and forth between the work that provides high value and other work, the more productive they will be. Not only will they be more productive, but they will also spend more of their time working on the highest customer value activities.

Benefits of Adopting Agile

The benefits of moving to Agile development can be split into two categories: benefits to the organization, and benefits to you personally. As a result of the high level benefits that Agile provides - more flexibility, higher ROI, faster realization of ROI, higher quality, higher visibility, and sustainable pace - Agile can provide the following benefits to the organization:

- Increased revenues
- Reduced costs
- Increased market share
- Higher customer satisfaction

Each of these benefits leads to a stronger organization which is then in a better position to reward you for your efforts both directly and indirectly. Some of these benefits include:

- Getting a raise and/or bonus – more discretionary income to buy cool stuff

- Improving your working conditions
- Actually using all of your vacation time
- The opportunity to spend more time working on cool stuff

In addition, Agile can provide the following direct benefits:

- A less stressful environment
- Less cancelled or shelved work
- Career advancement due to learning new skills
- Having the resume that gets you your dream job

The Magic of Agile Development

From a business perspective, the main reasons I appreciate Agile development are the benefits that I've described above. But from a purely personal perspective, the reason I enjoy Agile development is because it made my job more fun.

Today, thanks to Agile development, I interact with customers more than ever before. As a product owner, I do more demos and am able to provide new features that hit closer to the mark faster and more frequently than ever before. This in turn means more oohs and ahs from customers which is more fun for me and more profitable for the business.

Reinvest in Your Development Engine by Improving Your Work Environment

There are really only five ways to increase the profitability of a business based on software development: reduce costs via outsourcing, reduce headcount, reduce other expenses, increase productivity or increase revenues. Reducing expenses can only go so far. The most expensive part of software development is the people. Thus, one of the most successful ways to increase profits is to increase the productivity of the software development team.

The Agile Workplace

At Litle & Co., developers like the fact that Agile provides the additional challenge of solving business problems instead of just technical problems which requires thinking at a higher level. Developers at Litle report that they have a higher level of job satisfaction than in previous companies that were not using Agile because they see the results of their software development efforts installed into production every month. Also, they like the fact that there is much less work which amounts to "implement this spec."

Your development infrastructure is really no different than the general company infrastructure which includes your cube or office, the carpet, the artwork on the walls, the company cafeteria, your phone, your computer, and the company location. These are all part of your work environment. If you have a

computer that is 5 years old, your work environment is not as good as if you have a computer that is only 2 years old. If you are writing in C rather than C++, C# or Java, your work environment is sub-optimal.

The closer that your development infrastructure is to the ideal environment for your circumstances, the more productive your team will be. This principal extends to all aspects of the development environment, from development language, to build system, to build farm, to issue tracking system, to the process that you follow.

Your Development Process is Part of Your Work Environment

Your development process (regardless of how it is implemented), is also part of your work environment. If as a result of your development process you regularly end up redoing work because problems weren't discovered until just before the release, or projects get cancelled or shelved, then this is also likely to reduce productivity and job satisfaction. As this process improves, so does your work environment. The smoother it operates, the more pleasant your working environment will be.

There are many problems which you may think of as being unrelated to your development process. For instance, broken builds. Broken builds are simply the result of somebody making an idiotic mistake, right? Perhaps that's true some of the time, but most of the time it is due to the complexity of integrating many changes made by many people for software that has many interdependencies.

To be sure, a "perfect" process does not guarantee happiness, success, or the absence of problems. You still have to debug complicated problems, port to new platforms, deal with unforeseen circumstances, etc. However, the state of your process impacts the efficiency with which your effort is applied. For instance, if your process is perfect and completely frictionless, then 100% of your effort will be applied to the work that creates value. If it is rife with problems, it may mean that only 50% (or less!) of your effort will be applied to work that creates value. If there are problems with the process, then you are already expending effort which is essentially wasted. You would be better off investing some of that effort in removing the problems permanently instead of losing it to friction on a regular basis.