

# Agile Methods: The Bottom Line

Robert C. Martin  
29 May, 2003

For the last four years the software industry has been chasing it's tail trying to figure out what to do about Agile Methods like Extreme Programming. Are they any good? Do they work? Should we believe the hype? Should we start a pilot project? Will we believe the results?

Articles have been written touting the incredible benefits of Agile Methods. Other articles blast Agile Methods as regressions into the dark ages of software development. We hear people telling us of remarkable successes, and we hear others warning us of impending disasters.

(Sigh.)

Here's the bottom line. Agile Methods are about managing software projects. Nothing more. Nothing less. Agile methods are about providing managers with the data they need to make management decisions.

How do you manage a software project? Most project managers know that managing a software project involves more prayer than science. Why? Because getting accurate data about the project is very difficult.

The first piece of data we have about a project -- before we have a name for the project -- before we have any requirements for the project -- before we have selected the team -- is the due date. There are very good business reasons for this. Perhaps there is a shareholders meeting on a particular date. Perhaps there is a trade show. Perhaps our money runs out by a particular date. Whatever the reason for it, the date is chosen for *business* reasons, not technical reasons.

What's more, once the data is chosen, it is frozen. The only effective way to move the date is to miss it. Short of that, the date holds. The team may make estimates that show the date can't be met; but estimates are slippery things. Decades of data have shown us that we are *terrible* at estimating. Those same decades have also shown us that hope springs eternal. Most development teams, one way or another, will find a way to create an estimate that meets the required due date.

Now the managers have a date, and a set of estimates that meet that date. Next they might make a plan. They might draw a PERT chart, or a GHANT chart, that shows all the estimated tasks, and who is assigned to complete them. The chart clearly shows that the team will complete the project by the due date. Everyone is happy.

I don't need to tell you what comes next. The tasks take longer than estimated. More tasks are discovered that were never considered. Within weeks the chart is trash. It may stay on the wall for some time, growing ever less relevant with each passing day.

To make matters worse, the requirements are changing. We lost customer X, so we don't need feature F anymore. We might be able to sell to customer K, but we'll need feature G to do it.

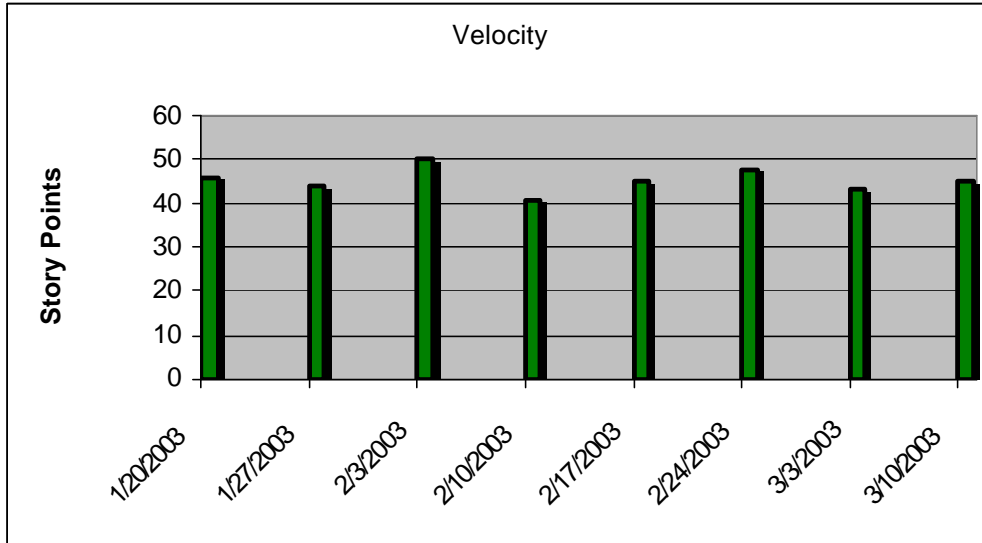
The managers may want to create a new version of the plan to hang on the wall, but they are stuck with the original due date. They know the business is depending on them to get the project done by that date. A new plan would certainly show the project slipping that date. So perhaps it would be better not to make the new plan.

Often managers will resort to pressure and motivation as their sole management techniques. They warn the developers of all the horrible things that will happen if they miss the date; and then they hang pictures of seagulls and people climbing rocks on the wall to motivate the developers to get the project done. They'll wander through the development area asking the developers how things are going. The developers will respond: "Pretty good!" And then, the managers go back to their offices to pray.

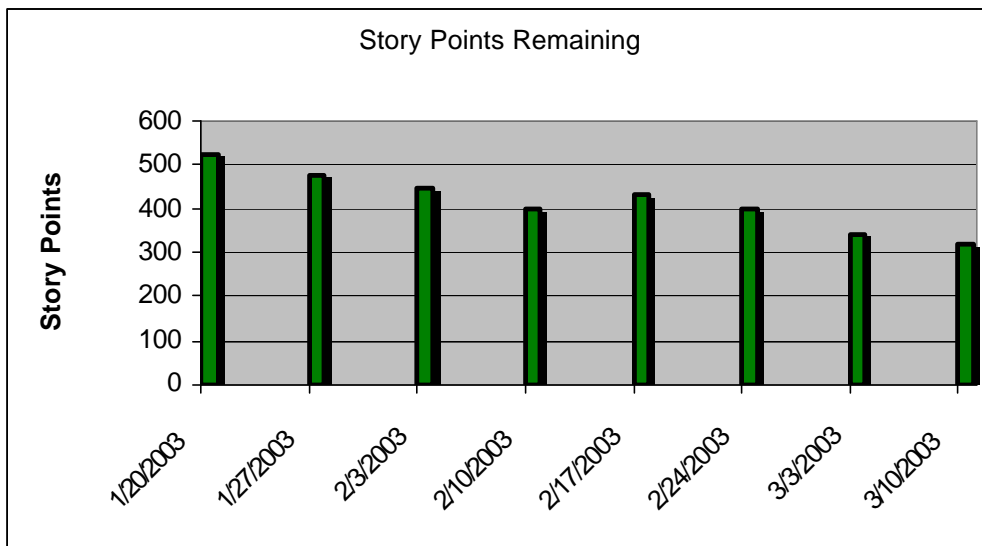
Yes, I'm exaggerating for effect. But I'm sure most of you have seen elements of this in the projects you have worked on. I certainly have.

## Managing with Data

What would happen if managers had real data to manage the project by? What if this chart were on the wall?



This chart shows, week by week, how much work is getting done. Anybody can look at it and see that the team is getting about 45 points per week done. That's powerful data, but we need more. So what if we also had this chart on the wall?



This chart shows, week by week, how many points remain in the project. You can see how the chart slopes down. You can also see hiccoughs in the trend where perhaps a new requirement was added, or the developers re-estimated something.

If we had both these charts on the wall, then anybody could look them over and see that the project was going to need about eight or nine more weeks to complete.

With data like this, you can manage the project. You can *see* how fast the team is working, and what the delivery date will likely be. And you can see this *early* in the project. Using this data you can make

decisions about adding staff, trimming scope, or even moving the deadline.

Without data, you can pray, you can hope, and you will almost certainly wait until it is too late to do anything. But with real data about the project, you can make the hard decisions in time to make a difference. You can manage the project to a good outcome.

## Generating the Data.

How do we get this data? That's what Agile Methods are all about. Agile Methods give us the means to collect this data and put it on the wall.

A project conducted under an Agile Method like Extreme Programming is broken up into a set of very small deliverables called stories. Each story is perhaps a man-week of effort or less. We know that our estimates are terrible, and we know that the requirements are going to change, but we have to start somewhere. So we start with lots of small stories.

We estimate each story as best we can, usually in arbitrary units of some kind. Some people use perfect engineering days. Other people just use points. We might estimate the login story at 3 points, and the logout story at 2. Perhaps the deposit story is 5 points, and the withdrawal story is 6. We don't care what the points represent, we just want a story with a 3 to be about half as hard as a story with a 6.

Next the managers and developers agree on a goal for the next week. Let's say this goal is 66. The team will try to get 66 points done in the next week. The managers and stakeholders select stories that add up to 66 and the developers implement them.

Of course we are terrible at estimating. So at the end of the week we've only gotten 40 points done. The other 26 get pushed back into the pile of stories that aren't done yet. This is disappointing, but now we know something we didn't know before. Now we know we can get about 40 points done in a week. So for the next week, the managers and stakeholders select stories that add up to 40.

This time things go better. By Thursday we've got 38 points done and some of the developer have run out of work. So we select another few stories and add them to the mix. At the end of the week we've gotten 45 points done. So the next week we select 45 new points, and so on.

It is easy to see how this scheme translates to the charts on the wall. Each week we draw a bar on the chart that shows how many points we got done that week. Each week we measure the size of the pile of remaining stories, and plot that bar on its respective chart. We also make a pass through the pile of remaining stories and re-estimate them based upon our new experiences. Thus, we keep the data on the wall as accurate and timely as possible.

One common objection to this scheme is that there are elements of software development that cannot be broken down into a small story. The claim is that things like infrastructure and architecture must come first, and will take longer than a week. This objection is simply incorrect. Team after team has shown that entire complex projects *can* be broken down into tiny stories. Team after team has shown that infrastructure and architecture can be grown from week to week instead of being created up front.

## Test Driven Development

Another objection is that the data are too easy to fudge. It is too easy for developers to claim that their stories are "done", when in fact they are not. This is similar to the old trick of checking in empty, or minimally implemented, functions so you can check them off the schedule. The fact that the function doesn't work becomes a bug report, instead of a schedule violation. (Yes, Virginia, there are teams that really do that!)

Test Driven Development -- specifically the Extreme Programming practice of Acceptance Tests -- provides the solution to this problem. When managers and stakeholders select stories to be developed, they must also work with QA to create *automated* acceptance tests. No story can be said to be complete until that story passes its acceptance tests. (See <http://fitnesse.org> for a free acceptance testing tool.). This practice allows managers and stakeholders to control what "done" means.

Controlling the definition of "done" means that managers also control the validity of the data on the bar charts. When the chart says that the team got 45 points done last week, it means that those 45 points were proven to work because the stories passed the acceptance tests that the managers and stakeholders specified. Therefore the managers can trust the data on the charts, and can depend upon that data to help

them make rational management decisions.

At first it can be hard to see how to write automated acceptance tests for all stories. Objections range from the difficulty of testing GUIs or device drivers, to the extra time it takes to write the tests. However, the benefit of *knowing* that the data on the bar charts is valid often provides sufficient motive to solve these difficulties. Managers are often willing to pay a high price for certainty and predictability. Moreover, the payback can be significant. Once a test is automated, it never has to be run manually again. Manual testing can be horrifically expensive.

## **Conclusion**

Agile Methods like Extreme Programming produce reliable data every week or two. Managers can use this data to make management decisions about the project. They can use it to predict when the project will be complete, or when the next release will be ready. They can use it to justify staff increases, or reduction of scope. They can use it to push back deadlines and reset customer expectations.

It is the production of this data that is the bottom line for Agile Methods. The data is their *raison d'atre*. If you are using an Agile Method, and you aren't producing this data, then you aren't really using an Agile Method.