



Building rock solid *relationships* and optimising technology every time.

## Better Tracking of Agile Projects

*By tracking both cost and progress, burn charts can be effectively extended to help everyone – from the geeks to the CEO – understand project status clearly and quickly.*

Before Agile came along, virtually everyone managed projects with Microsoft Project – for better or worse. Agile introduced an alternative in the form of “burn charts”, which were initially seen as a less rigorous technique. While the industry is becoming more comfortable with them, there are problems with burn charts. In particular, they don’t show actual costs very well. Your burn chart shows how much of the software has been built – but it doesn’t tell you how much the work has cost you to date.

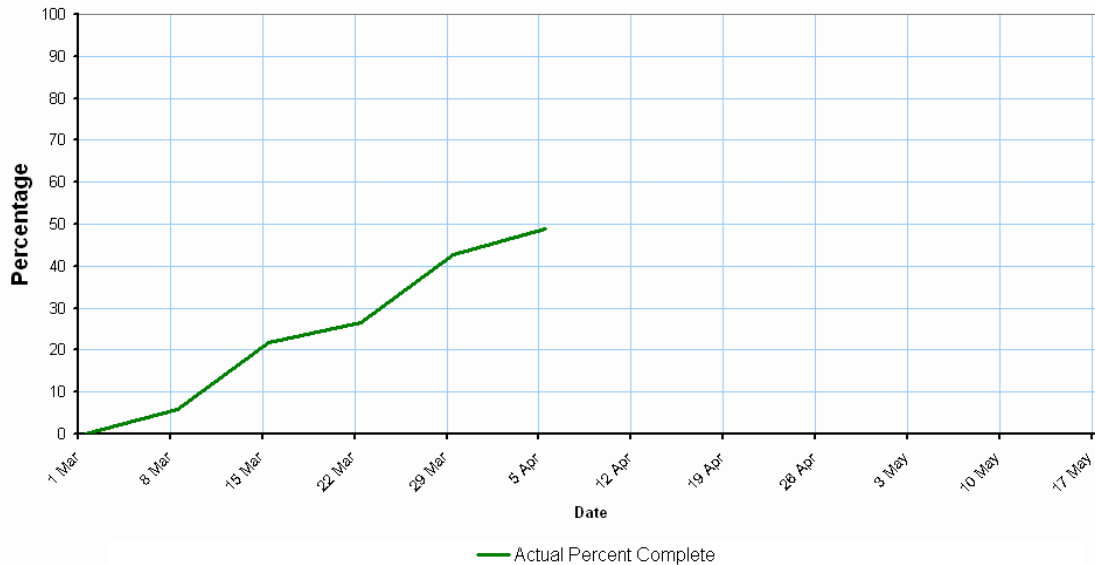
In this article, I’ll outline a simple extension to burn charts, to show costs as well as progress. By relating cost to progress *on the same chart*, you obtain a very powerful result. In fact, it amounts to a simplified version of Earned Value Management (EVM) – a robust technique which has been used for decades in the US Department of Defense, NASA and other agencies to objectively measure cost, scope and schedule on large projects.

The technique in this article is relevant to both agile practitioners and users of traditional EVM. In fact, I recently wrote about it from the EVM perspective, in a magazine published by the US Department of Defense. Here, I’ll describe it from the agile viewpoint.



Building rock solid *relationships* and optimising technology every time.

A standard agile burn chart looks like this:

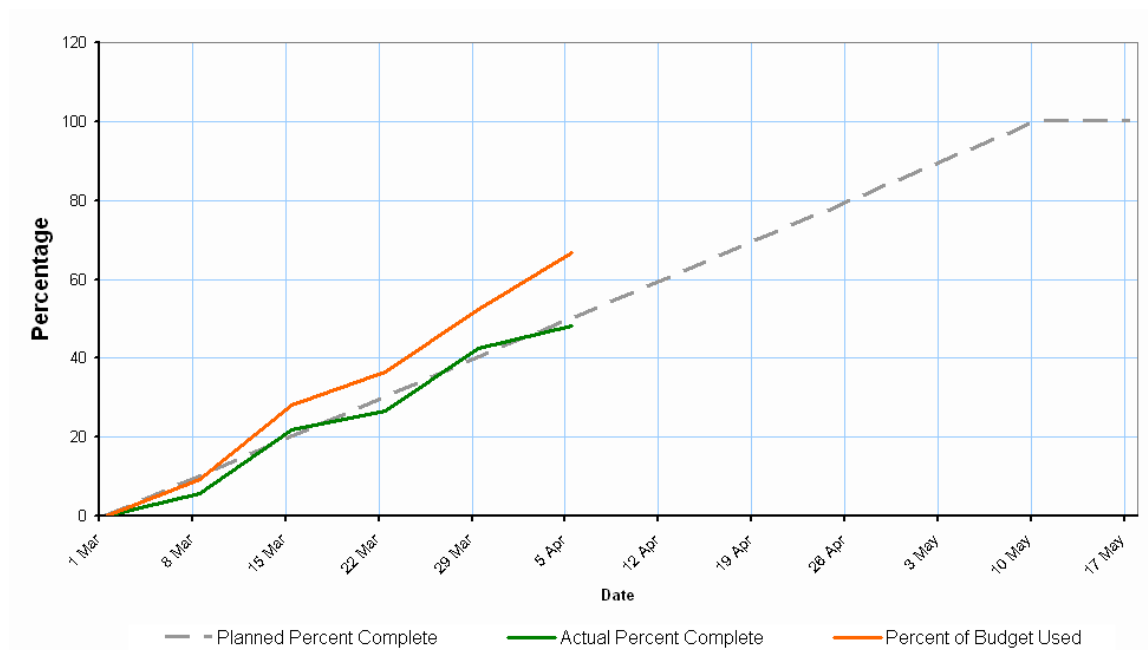


It tracks the completion of work, starting from zero percent complete and moving up to 100%. (Some agile teams draw their burn charts up the other way, charting work *remaining* instead of work *done*. I'll stick with the upward-sloping style because it matches traditional EVM charts.)

To track cost, simply add another line to track how fast you are "burning" the budget. I also like to add a straight line going from 0% at the start to 100% at the end, to represent the target.



Building rock solid *relationships* and optimising technology every time.

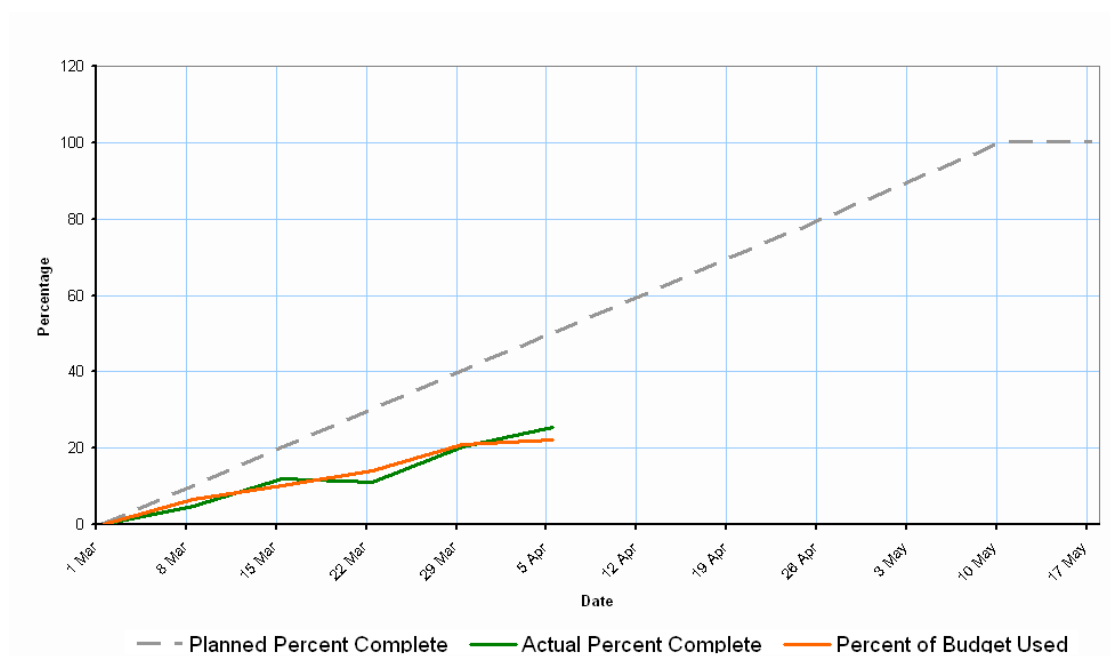


In the example above, the team is building the software at the scheduled rate. You can see that because the green line is following the grey target. However, costs are higher than planned, clearly shown by the red line tracking higher than the green one. If things carry on like this, the project will finish on time but about 30 percent over budget.



Building rock solid *relationships* and optimising technology every time.

Here's another example:



You can see this project is running very late. It should have been 50 percent completed by now, but only 25 percent of the work has actually been done. The good news is that cost is proportional to work completed – so while the progress is low, so are the costs. The typical cause of this situation is under-resourcing. People who were earmarked for the project have been called away to do other work, so cost and progress are both lower than planned. If the project carries on like this, it will be late but it won't blow the budget.

When you use an extended burn chart, you can immediately see the relationship between cost and progress. As the project proceeds, the red and green lines snake upwards. If the green falls below the red, you know you have a budget problem. If the green falls below the grey, you know you have a schedule problem. The chart will generally show these problems early, while you still have time to fix them.

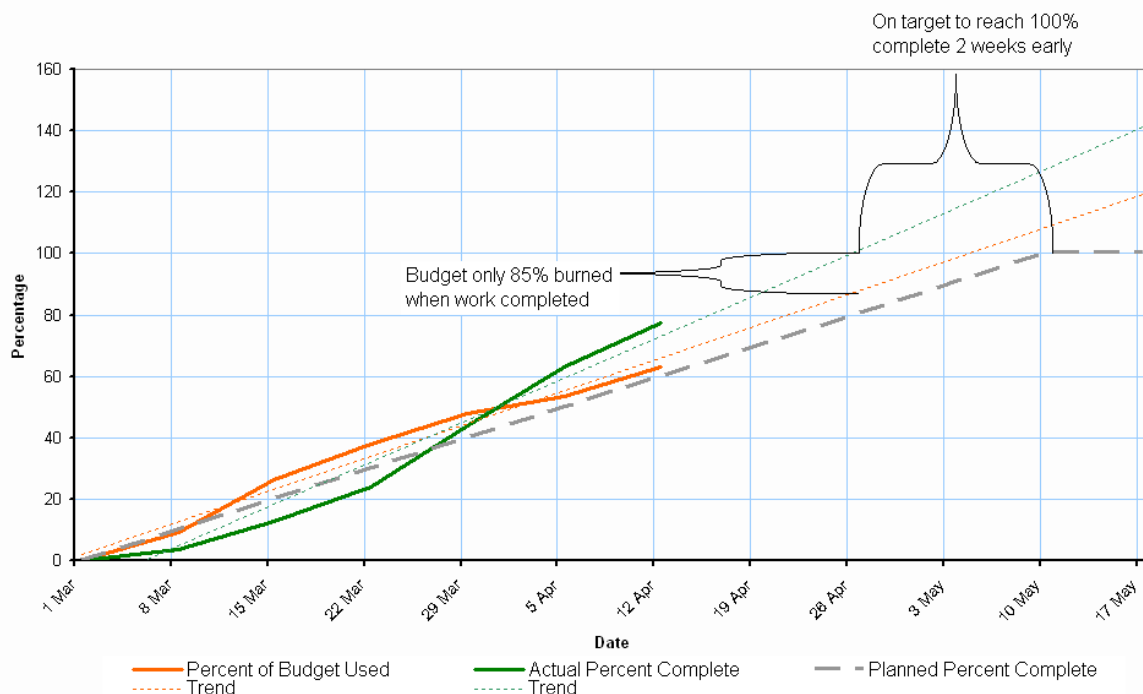
In our development teams at Optimization, we've also found the chart to be a useful motivational tool. If programmers can see it updated quickly – either as soon as they complete each piece of functionality or weekly at worst – it helps everyone to focus on making progress, avoiding scope creep, and finishing on time.

The technique works particularly well on agile projects because they're designed to progress at a "linear" rate, with roughly the same amount of work completed in each



Building rock solid *relationships* and optimising technology every time.

period of time. This means that the grey target line can be drawn as a straight line on agile projects, in contrast to traditional projects where it's usually an S-curve. The linear approach makes it easy to use the graph for estimation and forecasting, as in the following example. Here, by linearly extrapolating the red and green lines, the team can see they are on track to complete the project two weeks early, saving 15 percent on the budget.



At a very high level, this approach is exactly equivalent to traditional EVM. The main difference is that traditional EVM has the vertical axis in dollars. If you want to follow this approach, converting between percentages and dollars is simply a matter of multiplying or dividing by the project budget.

When you use this technique, it's important to consider the time period that your chart will cover. It needs to be long enough to show the big picture, and in my experience, a timeframe between three and 12 months works best. That might be one release of the product or even the entire project. You can also use the technique at iteration level – spanning time periods of a few weeks – but if you do, make sure you also chart the longer-term view because that's where this technique really shines.

Naturally, charting the longer-term view requires that you actually *have* a longer-term view. Contrary to popular opinion, a long-term view of scope is perfectly



Building rock solid *relationships* and optimising technology every time.

acceptable on an agile project. Amongst the various flavours of agile development, some have always required up-front scoping (e.g. the feature list in Feature Driven Development), while others allow it as an option (e.g. in Scrum you can create the product backlog either “up-front” or “just-in-time”). The “up-front” scope definitions are usually created after a few weeks of initial work. They don’t have to be detailed or perfect. A normal backlog or feature list is fine, as long as you create it early in the project and attach a rough estimate of relative size to each item on the list.

Finally, one common question with burn charts is whether you also need a Gantt chart. Generally, you don’t. Agile teams put the emphasis on *minimising* dependencies between tasks, rather than *modeling* them. For the few dependencies that can’t be eliminated, a Gantt chart is usually overkill. Perhaps the most telling comment on this subject comes from Chris Peters, who spent 18 years at Microsoft and rose to be Vice President in charge of Office. In the book *Microsoft Secrets*, he said that Microsoft Project is more appropriate for managing the design of airplanes and buildings, rather than software.

Instead of using Microsoft Project, at Optimization we’re building these extended burn charts on top of our existing agile project management tools. We’ve integrated closely with RallyDev and Microsoft Team Foundation Server, and developed basic integration for other tools. Having used this approach for several years, we’re now releasing our charting components to help other local companies get started with the technique.

#### **About the author:**

John Rusk is a software architect at Optimization and has worked in the Wellington software industry since 1995 creating bespoke applications for major government and private sector organisations. He is an active member of the agile development community and the author of two open source software projects. He has a B.Sc. in Computer Science from Massey University, New Zealand.

He can be contacted at [john.rusk@optimization.co.nz](mailto:john.rusk@optimization.co.nz)