Earned Value for Agile Development

by John Rusk, Optimation Ltd, © 2009, The DACS

"It's life, Jim, but not as we know it." [attributed to Mr Spock] Just as the Star Trek crew found life in very different forms, you can find earned value management in different forms too. But, like the crew of the Enterprise, you may have to travel a long way to find it. I am writing this article from Wellington, New Zealand, and I'm going to describe an approach to earned value management which is very different from the usual ANSI/EIA-748 approach. It's earned value, but (possibly) not as you know it.

There are three key points in this article:

- Agile and earned value management (EVM) are a natural fit for each other
- EVM implementations can be radically simplified for agile projects
- The ideas here are relevant to all EVM users, even if you have to use ANSI/EIA-748 and/or you don't use agile. In particular, they offer a way to share EVM results with stakeholders who don't know any EVM jargon.

Scope Management in Agile Projects

We need to begin by busting a few myths about agile:

Myth 1: Agile development is "Extreme Programming (XP) + Scrum".

In fact, there are many different varieties of agile development. The agile movement (formally) began in 2001 when 17 industry leaders met to identify the common ground that united their various viewpoints. All had their own approaches to building software. Some used Extreme Programming (XP), some used Scrum, some used Dynamic Systems Development Method (DSDM), some used Crystal Clear, some used Feature Driven Development (FDD) and the others used a variety of other approaches. Their intention was not to standardize their efforts on a single approach, but rather to document some common values and principles which they all agreed with. They documented those values in the Agile Manifesto, which became the "founding document" of the agile software movement. [1] The group of 17 fully intended to continue practising their own individual approaches, under the agile "umbrella". However in the years that followed the IT industry focused its attention on just two varieties of agile, XP and Scrum, to the extent that most people began to equate agile *only* with XP and Scrum.

Myth 2: Agile projects can't define scope up front.

They can. For instance, the FDD variety of agile has always defined the full scope up front. The Crystal Clear variety has always given the *option* of defining scope up front. Scrum and XP are generally seen as the least supportive of up-front scoping, but even Scrum does record the functionality to be developed in an

evolving list called the "Product Backlog". Some Scrum teams create the list up front; others create it as they go.

In summary, agile processes give you the *option* of producing a reasonably complete definition of scope at the start of the project. That's exactly what we need for EVM – we need overall targets for scope, time and cost so that we can track our progress towards them. You don't *have* to set these targets if you *just* want to do agile; but you do have to set them if you want to do agile *with EVM*.

Agile Burn Charts

Once you have set a target for the overall scope of the project, you can track your progress towards it. Most agile teams do so with a simple chart like this:

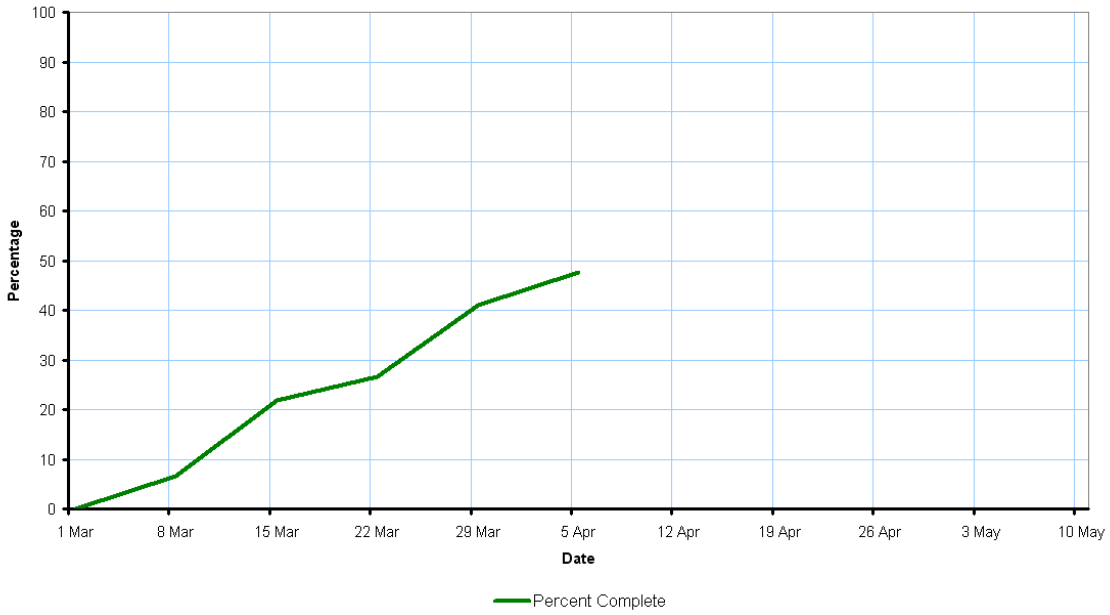

Figure 1 Burn Chart

Agile teams typically call these "burn charts", and they use the word "velocity" to refer to the slope of the line. Some teams chart the work that has been completed, as I have done here. Others chart the work *remaining*, in which case the line slopes downwards and the chart is called a burn*down* chart. I will stick with upward-sloping charts in this article, because they correspond more closely to the typical EVM chart.

As I will show below, the line of an agile burn chart is equivalent to EVM's Budgeted Cost of Work Performed (BCWP) also known as the "Earned Value" (EV). Therefore, most agile teams are already tracking earned value – although they may not use that word for it[1].

[1] In fact, when agile teams say "value" they usually mean "business value", which is the *usefulness* of proposed features. But in EVM "value" is not about usefulness; it is simply about *size*. It is the amount of budgeted cost which is considered "done" (i.e. earned) when the feature is complete. If you are an agilist reading this, then all you need to remember is that EVM "value" is the same thing as your "points".

The Agile Earned Value Technique

Where I work, at Optimation Ltd, we found that the simple agile burn chart was not enough for our agile projects. We wanted to understand the financial context of the team's progress. Was the project on track? Were we making enough progress, but only by blowing the budget with overtime? If we were running late, was it due to problems in the project, or was it due to other projects "borrowing" our people?

We answered these questions by enhancing our burn charts to include the cost of development. We soon discovered that our technique closely corresponded to traditional Earned Value Management, albeit in a much simpler form.

This is how we describe it to people for the first time:

We keep track of our projects by drawing a chart with three lines on it.

The grey line shows the progress that we expect to make, as a percentage of the total project. It starts at zero and slopes up to 100% at the end of the project.

The green line shows how much of the product we have built, *as a percentage of the total product size*. Ideally, it should follow the same path as the grey line – starting at zero and reaching 100% completed on the project's scheduled end date. If it falls below the grey line, that means we're running late; if it's above the grey line it means we're early.

The red line shows the cost we have incurred so far, *as a percentage of the total project budget*. In a perfect world, it would also follow the same path as the grey line - starting at zero and finishing at 100% of budget, on the project's scheduled completion date.

If the red line is above the green line, that means we're spending the budget faster than we're building the software – which is bad. If we carry on like that, we'll have a cost overrun. Conversely, if the green line is above the red line, that means we're building the software faster than we're spending the money – which is good. If we carry on like that, we'll finish under budget.

We don't use EVM acronyms. This simple approach makes it easy to communicate the ideas to absolutely everyone involved in the project – from the most junior worker up to our most senior executives. Everyone understands the idea after just a few moments of explanation.

We don't tell them that the grey line is Planned Value (aka PV or BCWS); the green line is Earned Value (aka EV or BCWP) and the red line is Actual Cost (aka AC or ACWP). Everyone simply refers to the lines by the names of their colors. For instance:

"The green line is still too low" = an unfavorable schedule variance (i.e. SPI < 1)
"The red line is above the green" = an unfavorable cost variance (i.e. CPI < 1)

As you may have noticed, we measure everything as percentages, rather than the usual EVM practice of measuring everything in dollars. I believe the percentage-based approach is easier for newcomers to understand: most business and technical people understand percentages with ease, but are not necessarily familiar with EVM's notion of measuring progress as the sum-of-the-budgeted-costs-of-completed-tasks. You can convert between percentage and dollar measures simply by multiplying or dividing by the total project budget[2].

How can you use this?

Since you'll be reading this in Software Tech News, I assume you're working in a DoD environment, and therefore have to follow the ANSI/EIA-748 standard. The simple approach in this article can *summarize the results of your ANSI/EIA-748 work*, in a form that can be easily understood by all your project team, and all your external stakeholders, without requiring training in EVM. The graphical form, free of acronyms, requires no explanation other than the brief paragraphs above.

You might also find this technique useful on projects which are too small for ANSI/EIA-748. For projects worth less than $20 million, the standards-based EVM approach is not mandatory and is often omitted, due to its cost of implementation. The simpler technique in this article is cost effective on even the smallest projects. We have used it on projects worth less than $100 *thousand*, and here's how we do it:

Implementing the Simplified EVM Approach

1. We simplify the distribution of Planned Value over time. We can do this because agile projects are designed to have a linear output over time – producing roughly the same amount of output in each iteration; therefore, we don't need to compute an s-curve to draw our grey (PV) line. We just draw it with a ruler (or the computerized equivalent). This technique is not perfect, but it is very easy to implement, fits well with agile projects, and gives a result that is adequate to our needs.

2. The chart is our sole format for reporting earned value; so we do not provide numerical tables. The chart covers all current work in the project; so we don't prepare breakdowns by WBS areas. When the chart identifies a problem, such as a dip in the green line, we use other techniques to identify the cause. (In fact, thanks to the openness and feedback of agile projects, we're often aware of the issue before the chart confirms its adverse impact.)

3. We simplify the gathering of actual costs. In particular, we don't break actual costs down by task. (We only graph the aggregate anyway, so we wouldn't use

[2] This is exactly the same conversion as that shown in the Defense Acquisition University's "Gold Card" of EVM acronyms, under the heading "Overall Status". We just use the formula round the other way – charting the percentages and deriving the dollars.[2]

the breakdown even if we captured it.) Since we're building software, our costs are almost entirely composed of labor costs. Each week we simply record the number of hours worked on the project, during that week, by each team member. Then we sum the hours and multiply by our hourly rate, to get the cost for the week[3].

4. We don't interface to our financial or timesheet system. To save on integration and setup costs, each team member simply reads their total hours for the week out of the timesheet system, and emails(!) the number to the project manager. This sounds crude, but is has been very effective and much cheaper than actually hooking into the timesheet system programmatically. It works because we don't need task-level costs, only the hours per week.

5. We have one very simple "earning rule": the earned value for a task accrues when the task is completed. You get no points for partial completion of a task. This practice is consistent with that normally used on agile burn charts and is referred to as the 0/100 rule in EVM. Furthermore, as in most agile teams, only working software features earn value. There is no earned value associated simply with designing something – you only score points when it is designed, built *and* tested.

6. The chart works best when the green (EV) line is "live". It may be in a web-based or desktop tool. The key point is that changes happen straight away; when developers complete tasks, they immediately see the green line move up a little. This instant positive feedback boosts motivation and ensures that everyone cares about "moving the green line up".

7. The chart works best when it covers a period of time that is big enough to feel like the "big picture", but still small enough so that the team can seen the line move up during the course of a (successful) day. For instance, I prefer not to use it on one-or-two week iterations since it doesn't show enough of the big picture at that timescale. Instead, I like it to cover roughly 3 to 6 months – either the whole set of iterations that make up one "release", or even the whole project. If your project is so big that daily changes are imperceptible, you can use two charts – one covering a period that is short enough for live/daily updates to be noticeable; and the other covering the whole project.

If your environment does not support these ideas, you can still consider making percentage-based graphs for easy communication with stakeholders. Simply feed your EVM data, including non-linear Planned Value, into one simple chart, color-code it, and then discuss it by color instead of by acronym.

Deriving other EVM Metrics – Visually

If you *can* make your Planned Value approximately linear, you will reap some additional benefits: linear PV makes it easy for stakeholders to visually "reason" about the graph.

[3] Multiply by the rate first, then sum, if different people are charged at different rates.

People can visually extrapolate the red and green lines, to intuitively grasp many of the concepts that are difficult to learn in classic EVM[4]. For instance, we can see this agile project is likely to overrun its budget by about 30%. (In classic EVM terms, the Estimate at Completion (EAC) is approximately 1.3 times the project budget.)

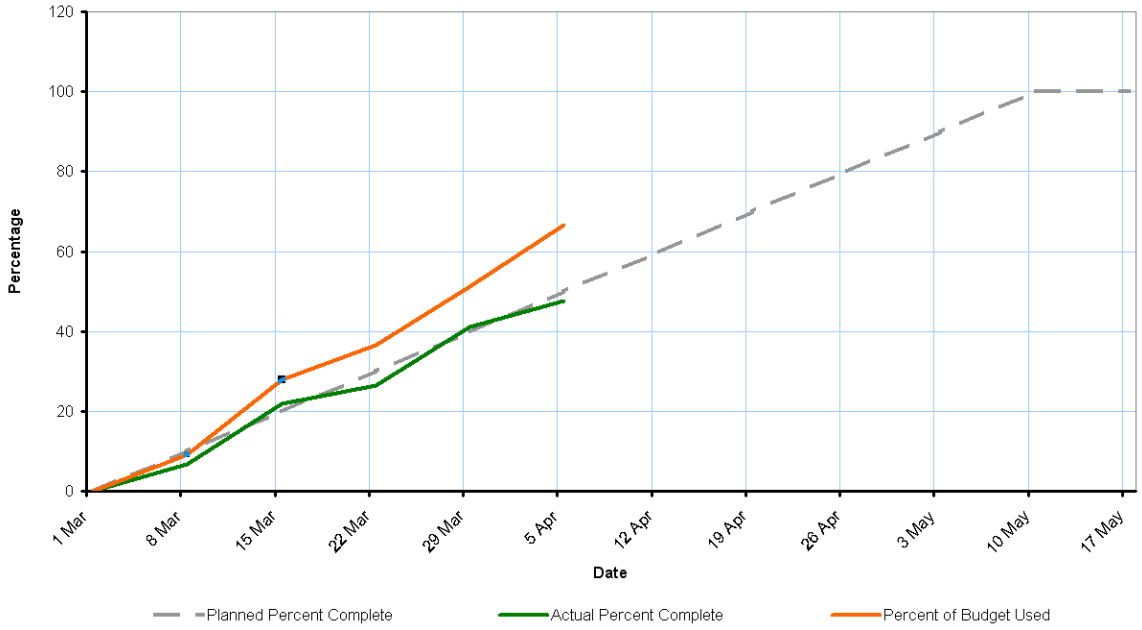

Figure 2 Heading for Overrun

In the next example, we can see that the project is well behind. If it is to complete on time and budget, future progress will need to be much faster than past progress (without spending more on resources). We can see this intuitively without getting into discussions of To-Complete Performance Index (TCPI).

[4] Such mental/visual extrapolation is easier with a linear project than with the traditional s-curve.

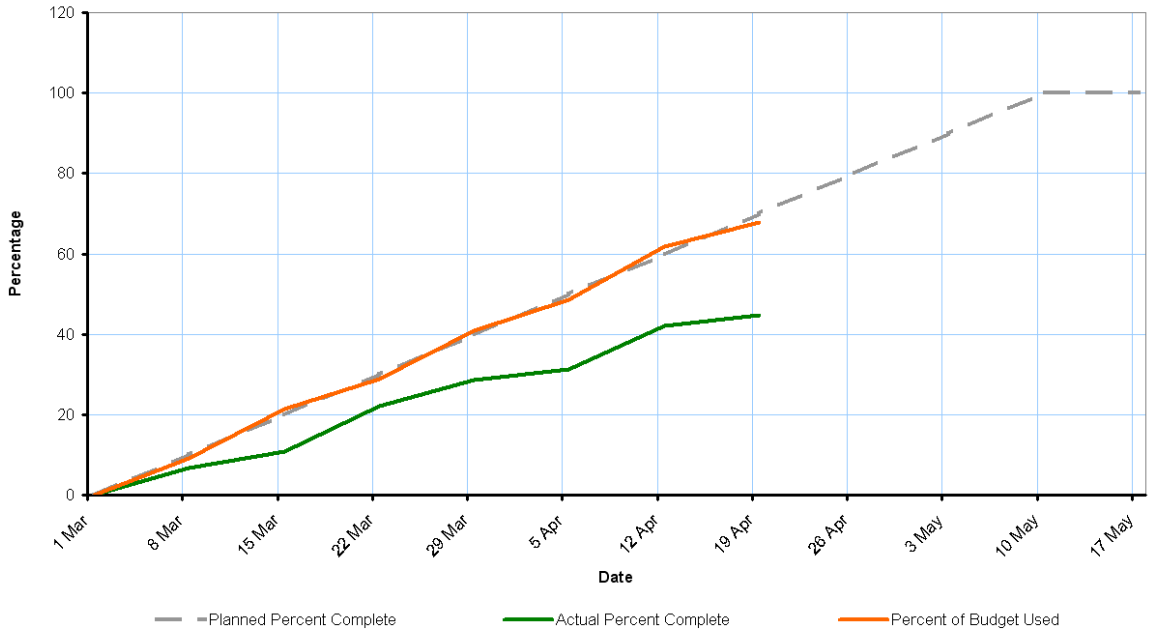

Figure 3 Needing a Miracle

Finally, in this instance, we can see that the project is on track to complete 2 weeks early, and that only about 85% of the total budget will be used when we get there.

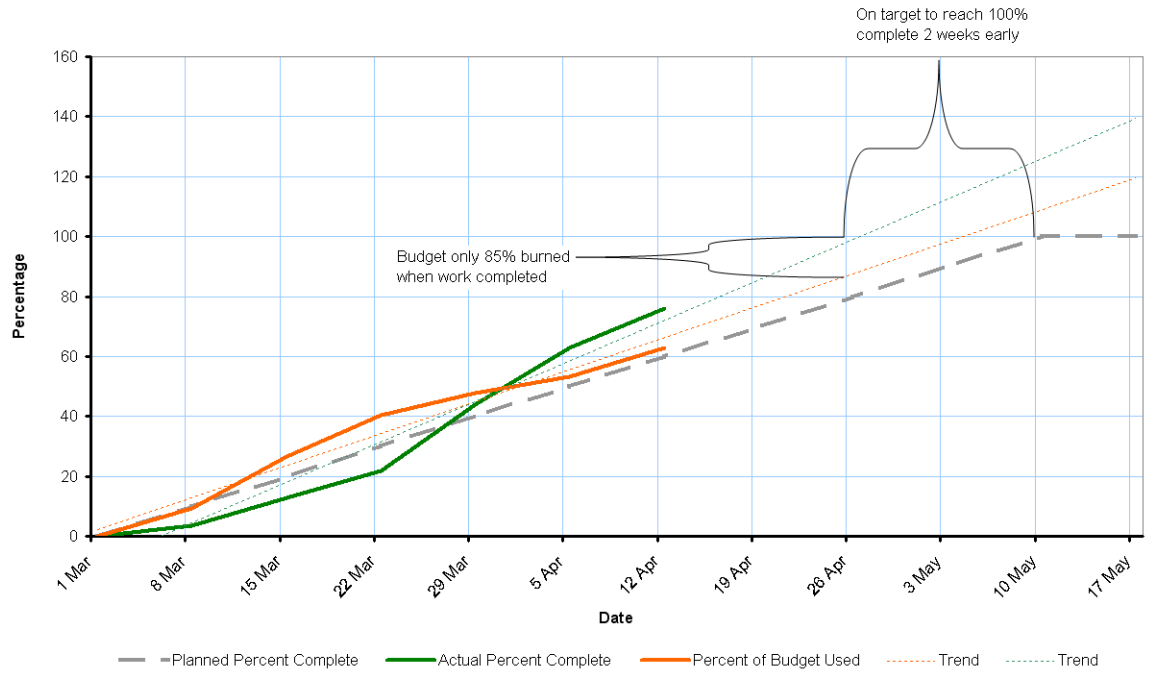

Figure 4 Completing Early

Note the users of the chart are able to intuitively reason about schedule variance *in calendar time*[5]. In the chart above we are asking ourselves, "What will the date be when the green line hits 100%?" I am told this approach is similar to the emerging practice of Earned Schedule.

How Detailed Does the Scope Definition Need to Be?

The scope definition doesn't need to be any more detailed than it would be on any other agile project. For instance, a Scrum Product Backlog is fine, as is an FDD Feature List. Details can (and should) be fleshed out on an iteration-by-iteration basis in the usual agile manner. This is remarkably similar to the "Rolling Wave" approach which is described in the DoD's Earned Value Management Implementation Guide [3], as an option available to all contractors:

> **2.5.2.4.2 Rolling Wave Planning.** The contractor may also elect to plan the PMB in detail WPs for near term activities and hold the future budget in higher level PPs. The contractor should periodically plan the next increment of work into detailed WPs. … Government approval of or interference with the process of rolling wave planning is not appropriate; however, the CMO and PMO should be aware of the contractor's schedule for rolling wave planning.

What about Changes in the Scope?

It helps to distinguish between two kinds of "changes". The first is simply fleshing out of detail, consistent with rolling wave planning and normal agile practices. As you do so, you will find unforeseen nuances of the previously-identified requirements. These are what author Robert Glass calls "derived requirements" – essential to the implementation but not explicit in the original user-facing requirements [4]. My preferred approach is to simply implement these as they arise[6], without reflecting them *at all* in the EVM system. This works because, by promptly implementing the derived requirements, you distribute them roughly evenly throughout the project. So, for example, if you have built half of the initially-known requirements (and you've implemented all their derived requirements as you worked); then you can be reasonably confident that you have done half of the *total* project. This approach minimizes the cost overhead of managing derived requirements. It also takes the pressure off the initial scoping phase, because it means the results don't have to be perfect. They only need to be as good as the typical Scrum Product Backlog or FDD Feature List. Adding simplified EVM to an agile project does not require a higher standard of initial scoping.

The second kind of change is when the customer comes up with completely new ideas or completely new areas of functionality. These are indeed additions to the scope. The

[5] Classic EVM measures schedule variance in dollars, which is counter-intuitive to the uninitiated.
[6] Make sure you only implement them if they really *are* essential to the success of the planned scope. Also, make sure you leave some spare capacity when you plan each iteration, because you'll need it to implement the derived requirements.

changes can be charted right on the simplified Earned Value chart. Details are beyond the scope of this article, but I have posted several examples on my web site. [5]

Applicability

The applicability of this technique depends on two things:

1. **Can you define scope up front?** Your definition doesn't have to be any better than you would normally do on an agile project; but you do have to do it at the start[7]. If you have a very exploratory project, in which you have no plans at all beyond the next few weeks, then you can't have specific targets for the end of the project, and so you have no context to assess your earned value. You still can (and probably should) use an agile process, but you can't use EVM. [8]

2. **Can you achieve (roughly) linear delivery of value?** Most agile processes are designed to do exactly this. But difficulties can arise if, for instance, you lack adequate automated test coverage, so you need a large manual test phase at the end. Big changes in team size can also complicate matters. Even so, you can still tolerate some non-linearity while charting your grey (PV) line as a straight line – as long as all stakeholders understand that the grey line is a linear approximation to the plan, rather than an exact representation of it.

You might be wondering how you can apply these ideas in a regulatory environment that is governed by ANSI/EIA-748. From where I sit (8000 miles away on the far side of the Pacific) I can't answer that – but *you can*. You know what's happening in your organization, so you're in the best position to identify improvements. Pick something which you think will be beneficial, which is achievable, and which fits within the rules. (Don't make the mistake of limiting yourself to the most common *interpretation* of the rules; there may be better ways to comply.) Try the idea out on a small scale, perhaps on one of your smaller projects. If it works, use its success to justify wider usage; if it fails, test your next idea. This approach is inspired by the emerging field of Evidence Based Management. I hear that the DoD already applies Evidence Based Management very successfully in some quarters. I wish you equal success and encourage you to seek advice from within the DoD, from agile experts, and particularly from the Evidence Based Management community.

---

[7] And you have to estimate the *size* of each element of scope up front (using "points", hours of effort or whatever your chosen sizing measure may be). Percentage complete (EV) is based on these sizes.

[8] There is a new practice emerging in some parts of the agile community in which features are built at a minimalist level first, and then fleshed out in later iterations as/when time and budget allow. One variation is called "feature thinning" and another is called the "A-B" split [6]. They are very powerful and useful techniques, but they do make it much more difficult to do burn charts and EVM. I will not discuss them further in this article.

Other Alternatives

This technique is by no means the only way to apply EVM to agile projects. For instance, Tamara Sulaiman and a team at SolutionsIQ implemented a version of "Agile EVM" which I see as being somewhat closer to the classical EVM approach[7]. Garry Booker of Project Frontier is developing a scalable model of Earned Value Management [8]. It begins with the simplest approaches of all – such as agile burn charts which track only the completion of features. From there it can scale up to simple "3 line" methods such as the one presented in this article; then scale up a little further to the Sulaiman/SolutionsIQ approach; and finally right up to ANSI/EIA-748.

Evaluation of the Simplified Approach

Let's compare the simplified implementation to the goals listed in section 1.2 of the DoD's Earned Value Management Implementation Guide [3].

DoD Goal	**Simplified Implementation**
Relates time-phased budgets to specific contract tasks and/or statements of work (SOW)	The "specific contract tasks" relate to the up-front scope definition. Actually scheduling those tasks, by slotting them into particular iterations, should follow your chosen agile methodology. With most agile approaches, the result will be an approximately linear flow of value.
Objectively measures work progress	The simplified earning rule encourages objectivity: the only thing that counts is completion of running, tested features. The insistence on the feature being tested before accruing EV is particularly important – EV is only earned when testing proves that the feature really is finished.
Properly relates cost, schedule, and technical accomplishment	The simplified approach relates cost, schedule and technical accomplishment in the same way as classical EVM – albeit with a linear schedule rather than the classic s-curve.
Allows for informed decision making and corrective action	This has definitely been our experience when using this technique – it identifies problems and allows us to respond promptly. The ease of understanding is also important. We circulate the chart to a wide audience, including decision makers who wouldn't have the time (or training) to interpret the numbers and acronyms of classic EVM. They can, and do, correctly interpret the color-coded chart.
Is valid, timely, and able to be audited	Timeliness is achieved by live updates to the green (EV) line and by weekly updates to the red (AC) line. Validity is ensured by driving the green line directly out of the project's task-tracking system, and by basing the red line on

	timesheet data (admittedly with a manual step to collect it).
Allows for statistical estimation of future costs	Future costs are estimated by linear extrapolation from the recent past.
Supplies managers at all levels with status information at the appropriate level	The format works well for high-level managers. For day-to-day management, project managers will naturally want more detail than what is shown in the chart. They don't obtain that information from the simplified EVM system. Instead, they obtain it from the project's chosen tool for agile project management / task tracking (just as they would have done if they were doing agile without simplified EVM).
Is derived from the same EVM system used by the contractor to manage the contract	In our case, we *are* the contractor. (We give the chart to the customer in our weekly reports.)

Conclusion

Simplified EVM approaches are relevant to both the agile world and the classic EVM world. In fact, agile thought-leader Alistair Cockburn has described such techniques as a bridge between the two worlds.

Key points for agile teams:

- You can, and often should, examine scope up front. For instance, you may populate your "Product Backlog" with an initial approximation of the full feature set.
- You can, and often should, add "budget burn" (AC) to your burn chart. Doing so gives a simple, but effective, earned value chart that conveys considerably more information than a standard burn chart.

Key points for users of classic EVM:

- When presenting EVM to stakeholders who aren't project managers, consider presenting it graphically, without acronyms.
- Simplified implementations don't need per-task actual costs.
- Simplified implementations can use linear Planned Value *if* they use agile techniques to achieve approximately linear output.
- Showing "live" changes to EV is good. It boosts motivation if the team can see their day's work having a direct impact on the green line.
- Use Evidence Based Management to improve your EVM implementation: what is working well? What is working poorly? Which changes will you try?

Acknowledgements: Thanks to Garry Booker, Alistair Cockburn and Bob Sutton.

References:

[1] The *Agile Manifesto* is on-line here: http://www.agilemanifesto.org/ Various descriptions of its origins can be found, both on-line and in print. Here's one: http://pragdave.pragprog.com/pragdave/2007/02/some_agile_hist.html

[2] The DAU *Gold Card*, Jan 2009, is available here: http://acc.dau.mil/CommunityBrowser.aspx?id=19577

[3] DoD *Earned Value Management Implementation Guide*, October 2006. Available from http://guidebook.dcma.mil/79/EVMIG.doc

[4] Robert L Glass, *Facts and Fallacies of Software Engineering*, 2003. pp 76 – 79.

[5] John Rusk, *Charting Change*, 2006, http://www.agilekiwi.com/charting_change.htm

[6] For an overview of Feature Thinning and related techniques, see http://alistair.cockburn.us/The+A-B+work+split,+feature+thinning+and+fractal+walking+skeletons

[7] Tamara Sulaiman, Brent Barton and Thomas Blackburn, *AgileEVM – Earned Value Management in Scrum Projects*, http://www.solutionsiq.com/PDF/Sulaiman-AgileEVM.pdf

[8] Garry L Booker, "Performance Management for All Projects" in *Project Landscape*, Feb 2009, vol. 3 no. 2, pp 3-5, available from http://www.pmitulsa.org/PMI2/images/stories/newsletters/pl2009.02.tulsa.pdf

Additional reading: *(added since original publication)*

Pages by Alistair Cockburn and Glen Alleman were my introduction to the link between Agile and Earned Value. The pages are:

http://alistair.cockburn.us/Earned-value+and+burn+charts
http://www.niwotridge.com/Resources/DomainLinks/EarnedValue.htm

Glen Alleman and Michael Henderson wrote this informative paper on how they successfully combined Extreme Programming (XP) with ANSI/EIA-748: http://www.niwotridge.com/PDFs/ADC%20Final.pdf

Mike Griffiths writes on Agile Earned Value here: http://leadinganswers.typepad.com/leading_answers/2008/06/a-better-s-curve-and-simplified-evm.html

This article was originally printed in the DoD Software Tech News, Vol. 12, No. 1. Requests for copies of the referenced newsletter may be submitted to the following address:

Ellen Walker, Editor
Data & Analysis Center for Software
P.O. Box 1400
Rome, NY 13442-1400

Phone: 800-214-7921
Fax: 315-334-4964
E-mail: news-editor@dacs.dtic.mil

An archive of past newsletters is available at www.softwaretechnews.com