# More honest earned value management of software projects

*Earned value should focus more on the progress and quality of the product, not the process.*

**Why is earned value management so hard for software systems?**
Earned value management (EVM) is a means of determining the financial health of a project by measuring whether the work completed to date is in line with the budget and schedule planning. The basic parameters of an earned value system, usually expressed in units of dollars, are listed below and illustrated in Figure 1.

- Expenditure plan: The planned spending profile for a project over its planned schedule.
- Actual cost: The actual spending profile for a project over its actual schedule.
- Actual progress: The technical accomplishment relative to the planned progress underlying the spending profile.
- Earned value: The earned value represents the planned cost of the actual progress.
- Cost variance: The difference between the actual cost and the earned value. Positive values show over-budget situations. Negative values show under-budget situations.
- Schedule variance: The difference between the planned cost and the earned value. Positive values correspond to behind-schedule situations. Negative values correspond to ahead-of-schedule situations.

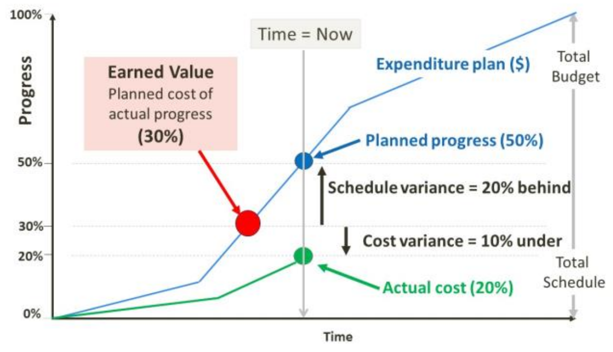

*Figure 1: The basic parameters of an earned value system*

For development programs, especially innovative efforts, *actual progress* is by far the most subjective and challenging assessment of all the parameters in an earned value system. Most projects know exactly how much cost they have incurred and how much schedule they have used. The variability in making accurate assessments of financial health is therefore centered in the fidelity of the actual progress assessment. Being subjective, it is easy to game.

Conventional earned value methods are based on a few assumptions:

- There is a linear relationship between resources expended and actual progress.
- The effort needed to achieve progress is predictable.
- Projects can plan the levels of effort and time in advance.
- Actual progress toward completion is straightforwardly assessable and measurable.
- The expected scope of the project captured in its requirements and baselined in budget and schedule targets is well understood and relatively static.

These assumptions hold for many routine civil engineering efforts, such as paving a road. If you have expended 5000 worker-hours to pave a 10-mile road, it will probably take another 20,000 hours to finish the full 50 miles planned. Software-intensive development projects require innovation, and like pharmaceutical research, start-up businesses and movie production, they start with incomplete information and significant uncertainties. The most critical progress is achieved primarily by reducing uncertainties. This is best enabled through executable demonstrations whose performance, usability, and function can be judged more objectively.

The realities of software-intensive systems violate the following assumptions, which form the basis for earned value management:

- The relationship between effort and results is nonlinear and typically follows more of a Pareto curve, where 80% of the value is achieved from 20% of the effort. The breakthrough validated learning occurs in relatively few critical activities, which are usually in design and integration testing.
- Early estimates of the planned work have high uncertainty. The variance in the initial project estimate (of effort or duration) is a measure of the missing project information. As the team gains knowledge and reduces uncertainty, the variance reduces, and estimation accuracy improves.
- No laws of physics or properties of materials constrain the problems or solutions of most software professionals. They are bound only by human imagination, economic constraints, and platform performance.
- In most software projects, you can change almost anything at any time: plans, people, requirements, milestones, designs, tests, and funding. Requirements — probably the most misused word in our industry — rarely describe anything that is truly required. Nearly everything is negotiable. So even if you could measure against an expectation, the expectation is constantly evolving.

As system design and development have become more and more dependent on software, experience with earned value management has become more problematic, and even unintentionally dishonest. The standard usage models do not accurately reflect the true health and status of software development and maintenance projects, as shown in Figure 2. The DoD and commercial industries have frequently experienced projects where 90% of the resources (time or cost) were expended after 90% of the earned value was claimed. With honest earned value, such reporting mistakes should not occur.

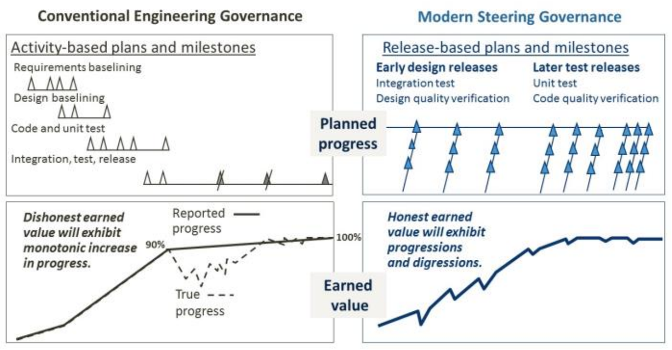
Conventional Engineering Governance
Modern Steering Governance
Activity-based plans and milestones
Requirements baselining
Design baselining
Code and unit test
Integration, test, release
Release-based plans and milestones
Early design releases
Integration test
Design quality verification
Later test releases
Unit test
Code quality verification
Planned progress
Dishonest earned value will exhibit monotonic increase in progress.
Reported progress
True progress
90%
100%
Earned value
Honest earned value will exhibit progressions and digressions.

Figure 2: Honest earned value shows progressions and digressions

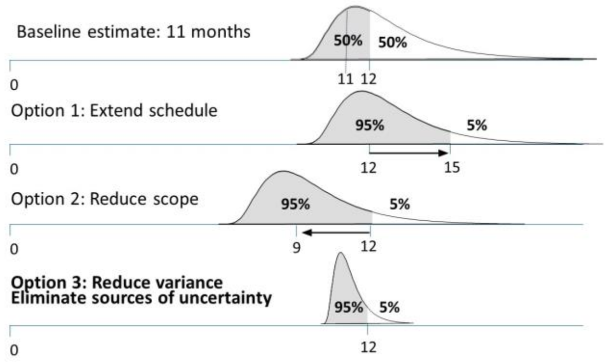

Figure 3: Transforming to probabilistic targets for steering

There are successful software projects that have used earned-value management effectively; they are rare. One precedent is the Command Center Processing and Display System-Replacement (CCPDS-R) project, a large USAF project delivered in the 1990s and well documented in a thorough case study. The modern steering profile on the right side of Figure 2 is a rough representation of the honest earned value approach used on CCPDS-R. TRW delivered the system on-budget and on-schedule, and the user got more than they expected. CCPDS-R was highly innovative and pioneered many new technologies, including the Ada programming language, a service-oriented architecture, reusable middleware, iterative development, employee profit sharing, and meaningful design quality metrics derived directly from the code base. CCPDS-R used technical metrics extracted directly from the evolving design, code, and test baselines for assessing actual progress and quality trends with more honest correlation to ground truth. This established credibility and trust among stakeholders and substantially reduced overhead.

**How can we transform to more honest earned value?**
To apply earned-value management to software programs more effectively, we must account for the nonlinearities of development progress and the uncertainties of innovation. A few key principles that can resolve this problem correlate well with accepted best practices in modern software development like lean, agile, and DevOps.

*Transparently express honest, validated learning*. Rather than a linear progression of monotonically increasing earned value, a healthy project will exhibit an honest sequence of progressions and digressions. Validated learning, which usually results in a near term regression, is the true measure that uncertainty is being reduced and estimates to complete are higher fidelity. Progress is measured through evolving demonstrable capability and refactoring (periodic digressions). Quality is measured through performance and usability, as well as scrap, rework, and defects extracted from release baselines and supporting artifacts.

*Communicate with distributions, not expected values*. Software designs and plans must be treated as a sequence of predictions with explicit uncertainty, as shown in Figure 3. Needs and designs emerge over time from a coarse vision with a wide variance (more uncertainty), to more precise, testable specifications with narrowing variance (less uncertainty). Nearly everything is negotiable early in the life cycle, and the feature set and operational characteristics remain negotiable as tradeoffs evolve from speculative debates to objective decisions.

Forecasted measures of actual progress, earned value, and planned estimates to complete should be captured as distributions of probable outcomes. It is fine to communicate an expected value, but a more honest exchange would include a discussion of the variance. Why are you confident (narrow variance)? Why are you uncertain (wide variance)?

*Pay attention to precision*. Measurement precision and fidelity should improve along the life cycle. As validated learning increases and uncertainties are resolved, more precision is appropriate. Precision in the life-cycle artifacts should be added as uncertainties are resolved. Unjustified precision, especially in early requirements, designs, forecasts, or plans, has proved to be a substantial source of future scrap, rework, and waste. The most common failure pattern in the software industry is to develop a five-digits-of-precision version of a requirement specification (or design, or plan) when you have only a one-digit-of-precision understanding of the problem. Unfortunately, many stakeholders demand early precision because it gives them (false) comfort in the progress achieved. Emphasis on accurate yet imprecise estimates with a discussion of the uncertainties remaining, namely the reasons for imprecision, builds trust.

*Measure the product as well as the process.* Assess earned value from the design, code, and test base evolving through successive life-cycle checkpoints. Direct measures of the code and test base are objective facts and mostly signal. Measures of the process and other supporting artifacts are indirect indicators and more subjective guesses. They are noisier and easier to game. Traditional project management measures focus on the efficiency and agility of the process. But these process measures provide only half of the insight you need. Design quality and code quality measures are the other half, the more important product measures that teams need to steer software outcomes more predictably. *Business agility is as much a function of product design as it is of process*.

## Contact Us

Silverthreads's mission is to advance the state of software measurement practice by quantifying complexity and design quality. Our measurement know-how can establish a more trustworthy foundation for improving software economics.
http://silverthreadinc.com