How to Estimate, Manage, and Track Performance on Modern Federal Software Development Programs

Authors

Bob Hunt, President, Galorath Federal
Dan Galorath, CEO Galorath Incorporated
Ian Brown, Director of Operations and Systems Analysis, Galorath Incorporated
David DeWitt, Galorath Incorporated
Esteban Sanchez, Senior Cost Estimation Consultant, Galorath Incorporated
Joe Dean, Senior Cost Analyst, Galorath Federal

Abstract

The recent final report of the Defense Science Board (DSB) Task Force on Acquisition of Software for Defense Systems (February 2018) outlines key challenges facing Defense with regard to software acquisition. The summary report states, "Software is a crucial and growing part of weapons systems and the Department needs to be able to sustain immortal software indefinitely." The report identified issues ranging from poor initial estimates to the Federal Government's inability to utilize modern developmental tools, such as the "Software Factory." The world is aggressively moving toward Agile with the assumption that Agile is faster, cheaper, and more effective. This paper will discuss the DSB findings and outline how to effectively estimate, manage, and track modern federal software development programs.

Contents

1. Introduction

In this paper the term "Agile," includes not only all forms of Agile but also any type of iterative approach for software development. We consider stories, features, story points, and feature points to reflect the same concept, recognizing that a "feature" typically may be used in a different context than a "story." Specifically, in large federal programs, "features" generally represent a larger concept than "stories." Based on our experience with large federal IT and software programs, we believe it to be unlikely that they will ever achieve the speed and efficiencies that are often realized by Agile practices at commercial "tech" companies such as Google or Spotify. We do believe that the application of estimating, management, and tracking practices recommended in this paper can significantly and positively impact the success and cost of federal programs.

In addition, we see two classes of federal software development programs. In the first case, federal programs that are evolving on an incremental basis generally follow the commercial Agile practice of small user stories that are completed in a single sprint, or even multiple user stories being completed in a single sprint. These fast-evolving federal programs do not generally follow an Earned Value Management (EVM) process. An example of an incremental federal program might be the USMC Tactical Architecture. In this program, requirements came in from the warfighter, and system fixes/changes were pushed out on a six-month basis. Programs like this followed a more traditional Agile process and often do not consider program tracking.

There are, however, large "transformational" programs like the Customs and Border Protection Automated Commercial Environment program and the Army's Integrated Personnel and Pay System that are creating completely new capabilities. In these "transformational" programs a "Hybrid-Agile" approach is often applied with longer sprints and larger conceptual stories/features and a full EVM process.

This paper addresses the processes that can be applied to incremental and transformational federal software development programs.

2. Status of Software Development Programs – Cost and Schedule Overruns

There are many studies attempting to quantify the cost of software failures. They don't agree on percentages, but they generally agree that the number is around $50 to $80 billion annually. The Standish Chaos Report, which is probably the most well-known of these studies, defines success as projects delivered within budget, on schedule, and with expected functionality. The 2018 Chaos report shows:

Successful Projects: 30%

Challenged Projects: 52%

Failed Projects: 19%

The 2015 CHAOS Report released by the Standish Group showed similar outcomes.

Software development is a major element of federal programs. Major Automated Information Management Systems (MAIS) that achieve their original cost and schedule projections are rare. Many information technology projects have been declared a disaster area by commercial and government managers. These projects have been too costly, too late, and often don't work right.

Applying appropriate estimation, management, and performance tacking (earned value management (EVM)) techniques can significantly improve the current situation.

Figure 1 below, "Why IT Projects Fail," is taken from the *2012 Gartner Report*[1]

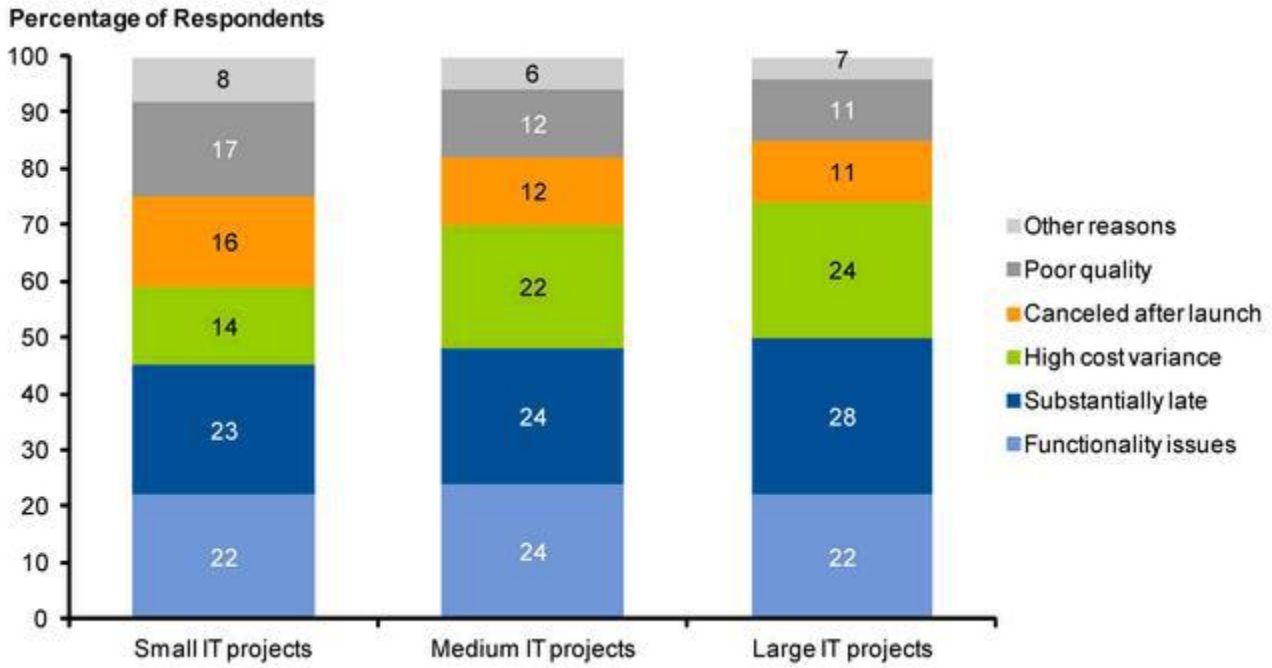

Figure 1. Why IT Projects Fail

Inaccurate estimates without bounding of risks can threaten project success by causing poor project implementations, shortcutting critical processes, and requiring emergency staffing to recover schedule. The lack of well-defined project requirements and specifications may result in significant growth in cost and schedule. Symptoms of this growth may include constantly changing project goals, volatile requirements and project scope, frustration, customer dissatisfaction, cost overruns, missed schedules, and the failure of a project to meet its objectives.

Additionally, recent studies like the Institute for Defense Analysis (IDA) paper, *Software Productivity Trends and Issues* (David M. Tate, March 2017), indicate this may get worse. The National Research Council (2010) wrote that "The extent of the DoD code in service has been increasing by more than an order of magnitude every decade, and a similar growth pattern has been exhibited within individual, long-lived military systems." One order of magnitude per decade is approximately 25 percent annual growth.

Figure 2, "Software Growth in Aircraft Systems," taken from the Defense Science Board (DSB) report, illustrates this trend for avionics software. Software continues to be a critical and growing component in defense programs. A study of code growth within aircraft systems was performed by the Defense Science Board. Figure 2 highlights their findings that software size continues to grow.

1 Gartner 2012

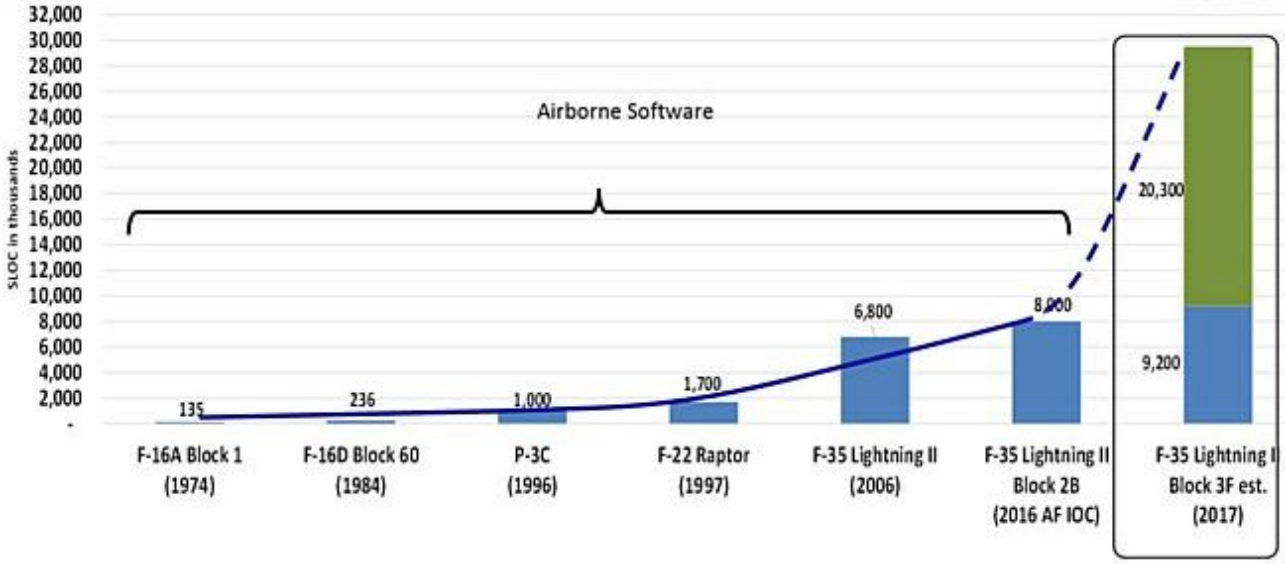

Figure 2. Software Growth in Aircraft Systems

3. DSB Report and Findings

Software continues to be a critical and growing component in defense programs. The DSB Task Force on the Design and Acquisition of Software for Defense Systems was chaired by Dr. William LaPlante and Dr. Robert Wisnieff. The key findings are no surprise: Software development in the commercial world has far outpaced that in the Department of Defense (DoD). The study recommends DoD adopt best practices on risk reduction and metrics in formal program acquisition strategies. Many of the findings are obvious and others are groundbreaking. The summary recommendations of the report are:

- **Recommendation 1:** Software Factory. A key evaluation criterion in the source selection process should be the efficacy of the offeror's software factory. The "software factory" is a low-cost, cloud-based computing system used to assemble a set of tools that enable the developers, users, and management to work together on a daily tempo.
- **Recommendation 2:** The DoD and its Defense Industrial Base partners need to adopt continuous iterative development best practices (continuing through sustainment) for software.
- **Recommendation 3:** For all new programs, the following best practices should be implemented:
    - Allow multiple vendors to begin work with down select after at least one vendor has proven they can do the work.
    - Service cost estimators should modernize cost/schedule estimation processes.
    - The project manager should build program-appropriate frameworks for status metrics; examples include: sprint burndown, epic and release burndown, velocity trending, control chart, line of balance and cumulative flow diagrams.

- **Recommendation 4:** Current and legacy programs in development, production, and sustainment should task current program managers to plan transition to a software factory and continuous iterative development.
- **Recommendation 5:** The government needs to develop a modern software development expertise in its program offices and a broader functional acquisition workforce.
- **Recommendation 6:** Software is Immortal. Therefore, RFPs should specify the basic elements of the software framework supporting the software factory including code and document repositories, test infrastructure, software tools, check-in notes, code provenance, and reference and working documents informing development, test and deployment.
- **Recommendation 7:** Implement Independent Validation and Verification (IV&V) for Machine Learning.

By applying the concepts recommended in this paper, DoD can leverage today's commercial development best practices to its advantage, including on its weapons systems.

4. No Estimates Movement (#NoEstimates)

The authors of this paper have over a century of combined experience doing software estimates. We seldom got it exactly right, but very often produced estimates that were of value to management. That should be the goal of a cost estimate.

The Agile framework has changed the way we approach software development while awakening a new dilemma about the need for project cost estimates. Many Agile enthusiasts are making the argument that estimates are difficult to produce and provide little value. On the other hand, organizations need to budget, and business leaders need estimates to make informed decisions. This is especially true for large federal programs.

With respect to Agile software development, there is significant chatter in the industry going on under the heading of #NoEstimates. The basic idea is that developing small chunks of software at a time can lead very rapidly to a desired shippable product increment, and when you take this approach there is no need to do estimation for the project as the "overhead" of doing so would only slow the project down. For Silicon Valley startups, or an entrenched company like Google who are continually evolving new capability on an hourly/daily basis, this may be true. These types of applications do not necessarily have an estimate and a fixed budget associated with them. If an application fails they do not have to worry about someone losing their life or bringing down and entire mission critical system. However, this is not the world of the federal acquisition manager.

Underlying truths are:

- Software estimation is challenging,
- Agile developers see estimates as constraining them to a schedule and therefore they are antithetical to the Agile Manifesto
- Software estimates drive decision making - they are not just for the developers, and

- Total ownership cost should be considered for the immortal systems and as a result, more emphasis and research should be and is being applied to the area of software maintenance.

In *Learning Agile: Understanding Scrum, XP, Lean, and Kanban* (Andrew Stellman and Jennifer Greene, 2014), the authors say "Plans don't commit us. Our commitments commit us: the plan is just a convenient place to write those commitments down. Commitments are made by people and documented in project plans. When someone points to a project plan to show that a commitment was made, it's not really the piece of paper that the person is pointing to. It's the promise that's written down on it that everyone is thinking about."

Good estimation practice requires that you break down the project into pieces that can be estimated. For Agile development programs, this is often code that can be completed in a "time-boxed" sprint period. We believe the real issue with estimation is that Agile software developers do not want to be constrained by their estimated schedule. However, the purpose of the estimate is not to constrain the developer. Rather, the purpose it is to allow the federal manager to appropriately resource the requirement.

During an "Agile in Defense Conference", Brent Barton, Past President of Agile Advantage Inc. outlined the importance of using EVM and described how to use it effectively in an Agile development. Once these EVM Databases are established within an organization, effective estimates can be developed to assist the federal manager to appropriately resource the requirement.

Recently, the USAF successfully completed a space system development on time and within cost using EVM in their Agile development process. It proved to be extremely effective in their management for controlling cost, schedule, and quality of the system.

5. Developing Reliable Estimates for Software Intensive Programs

Agile/Iterative Software Development Programs

The Agile/Iterative developmental framework has changed the way DoD approaches software development. It is making software development leaner. This approach has introduced new sizing metrics such as story points and new estimation techniques, e.g. velocity. However, many organizations still prefer function points or SLOC based methods.

Recently the authors of this paper have found success using Agile estimates built from the sprint team up. Then a function point analysis is developed to exercise a cost model, such as SEER for Software. This model based estimate is then used to cross-check and validate the bottoms-up estimate.

Similarly, a conversion to SLOC analogies and/or accomplishing a function point count has been successfully utilized to accomplish Agile/Iterative program developments. Cost estimates need to consider not only the Agile framework itself, but how the specific Agile methodology is adapted to the specific environment and organization. As Figure 3 shows, Agile/Iterative software development is a mindset and not a single method.

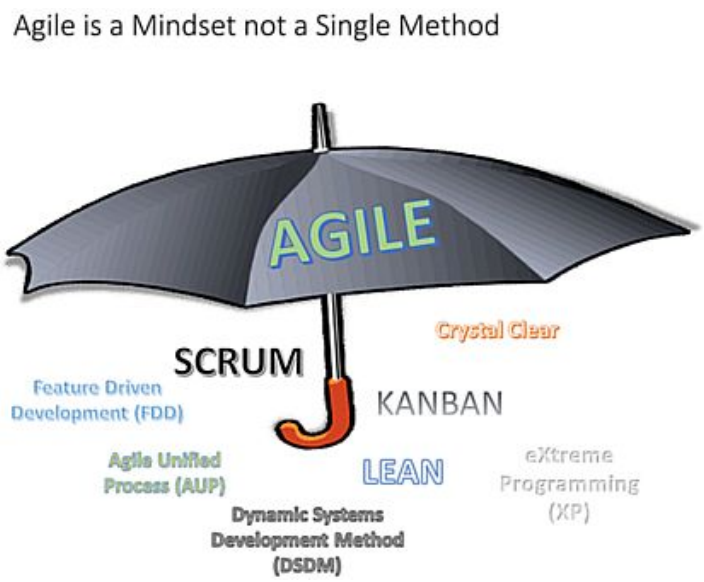

Figure 3. Agile is a Mindset, Not a Single Method

Estimates need to respond to the business reality and not just defer solely to the Agile Manifesto. Some of the common techniques used in Agile estimation include:

- **Story Points:** Arbitrary points are assigned to each user story (a.k.a requirement). Story points sizing is the most common Agile estimation technique. Our experience shows a consistency among project teams accomplished within a specific business unit. However, for a large corporation or between corporations, little objectivity and consistency is found. Story points are too arbitrary and not portable.
- **Planning Poker:** Participants use numbered playing cards and estimate the items in this new Agile version of the Delphi technique. Voting is transparent, and teams are encouraged to come to a consensus value for the "story" under consideration. Each participant brings a unique perspective to the complexity of the "story".
- **T-Shirt Sizing:** Create several "buckets" of size such as Small, Medium, Large, Huge, etc. The group estimates the user stories by placing them in one of these buckets.

Some other sizing methods typically applied to Agile estimation are shown in Figure 4, "Agile Sizing Methods."

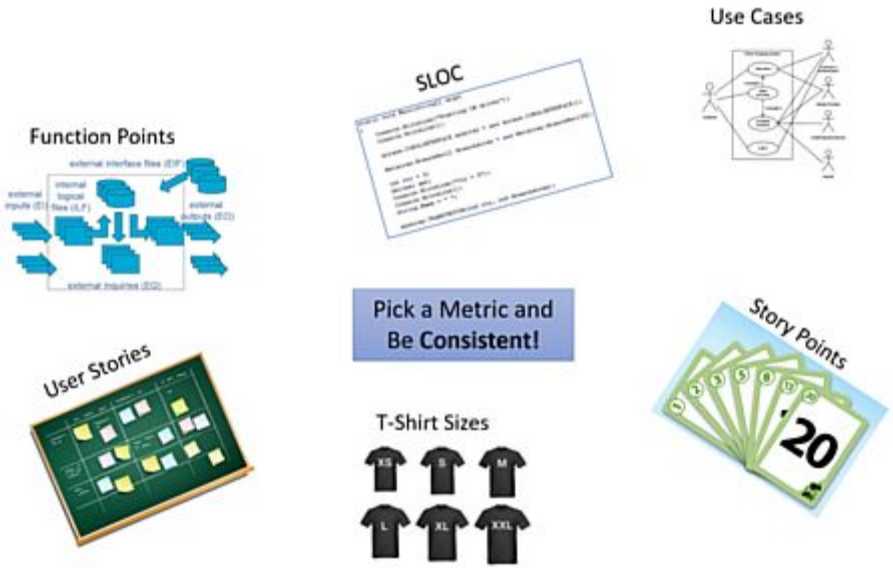

Figure 4. Agile Sizing Methods

Agile estimates can be done using any sizing method when the organization is consistent at adopting and applying a specific method and tool. Generally, this requires a cohesive team with small variances across the life of the project or onto new programs.

Defense cost agencies have invested a tremendous amount of time researching Agile estimation techniques and working with customers in the field to understand their needs and challenges. We have collected a series of industry best practices that have proven meaningful and beneficial for organizations that have adopted Agile practices. These best practices include: sizing methods, processes, training, level of detail, Agile drivers, velocity prediction, and more.

Galorath has developed an approach to help customers use known inputs, such as team size, backlog size and sprint duration, and simulate what they don't know - such as number of sprints, recommended team size, industry reference velocity, overall effort and cost, and more. As figure 5, The Cone of Uncertainty, below illustrates, estimating Agile software development is not more difficult; it is just a different software development methodology. The concept of the Cone of Uncertainty, originally introduced by Dr. Barry Boehm, is still applicable to Agile projects: Estimates gets more precise as more information becomes available.

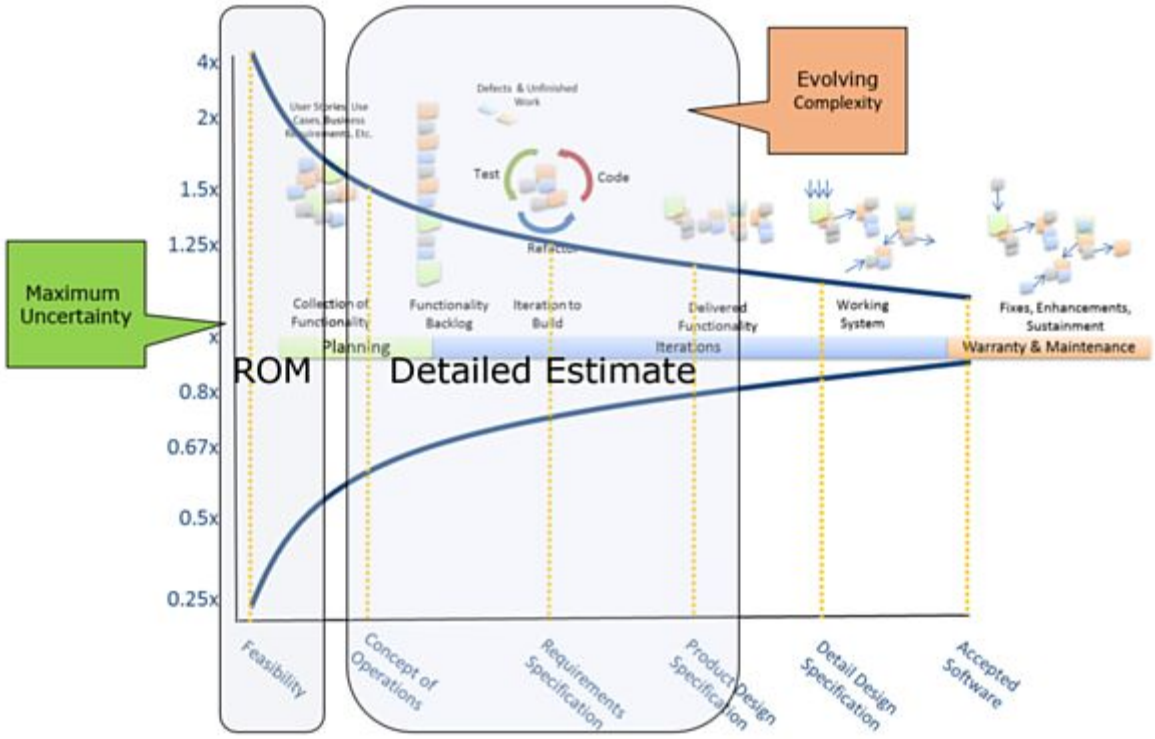

Figure 5. The Cone of Unceretainty

6. Ten Step Process

While Agile developers may not see a need for estimates, the managers need them for planning, budgeting and generally setting expectations of cost, schedule, and risks compared to the value the software generates. Software estimation must follow a consistent process that is integrated with the software development process to establish realistic and credible plans to implement the project requirements and satisfy commitments. The Ten Step Process outlined below is found in Dan Galorath's book, *Software Sizing, Estimation, and Risk Management* (2006), and is described in Figure 6, "The Ten Step Estimation Process."

Figure 6. The Ten Step Estimation Process

Step 1. Define and document estimate expectations.

When all participants understand the scope and purpose of the estimate, you'll not only have a baseline against which to gauge the effect of future changes; you'll also head off misunderstandings among the project group and clear up contradictory assumptions about what is expected. Agile practitioners are expected to embrace "change." This expectation of change needs to be incorporated into the estimate by providing an estimate range as opposed to a single-point estimate.

Documenting the application specifications, including technical details, external dependencies and business requirements, will provide valuable input for estimating the resources required to complete the project. The more detailed the specifications, the better. However, within an Agile project, the expectation of formal requirements is often minimized. Those engaged in estimating Agile projects need to be mindful of the degree of and formality of specifications.

The estimate should be considered a living document; as data changes or new information becomes available, it should be documented and factored into the estimate to maintain the project's integrity.

Step 2. Establish Technical Baseline, Ground Rules, and Assumptions

To establish a reasonable technical baseline, you must first identify the functionality included in the estimate. If detailed functionality is not known, ground rules and assumptions should clearly state what is and isn't included in the estimate. Issues of COTS, reuse, and other assumptions should be documented as well. During most Agile projects, the final delivered functionality is

notional when the first estimates are generated. Ultimately, the final delivery may have more, or less, functionality than originally envisioned. This uncertainty needs to be captured and documented within the estimate as it forms the basis of the estimate.

Ground rules and assumptions form the foundation of the estimate and, although in the early stages of the estimate they are preliminary and therefore rife with uncertainty, they must be credible and documented. Review and redefine these assumptions regularly as the estimate moves forward.

Step 3. Collect Data

Any estimate, by definition, encompasses a range of uncertainty, so you should express estimate inputs as least, likely and most rather than characterizing them as single data points. Using ranges for inputs permits the development of a viable initial estimate even before you have fully defined the scope of the system you are estimating.

Certain core information must be obtained to ensure a consistent estimate. Not all data will come from one source and it will not all be available at the same time, so a comprehensive data collection form will aid your efforts. As new information is collected, you will already have an organized and thorough system for documenting it.

Step 4. Software Sizing

Size is generally the most significant (but certainly not the only) cost and schedule driver. Overall scope of a software project is defined by identifying not only the amount of new software that must be developed, but also must include the amount of preexisting, COTS, and other software that will be integrated into the new system. In addition to estimating product size, you will need to estimate any rework that will be required to develop the product, which will generally be expressed as sprints, stories, features, source lines of code, or function points. There are also other possible units of measure, e.g. use cases. To help establish the overall uncertainty, the size estimate should be expressed as a least—likely—most range.

Whenever possible, start the process of size estimation using a formal description of the requirements such as the customer's request for proposal or a software requirements specification if available. For many Agile developments the initial requirements may be in the form of business requirements, high level user stories, or desired features (together, all these requirements are typically referred to as the product backlog). You should re-estimate the size as soon as more scope information is determined. The most widely used methods of estimating product size are:

- **Expert Opinion:** This is an estimate based on recollection of prior systems and assumptions regarding what will happen with this system, and the experts' prior experience.
- **Analogy:** A method by which you compare a proposed component to a known component it is thought to resemble, at the most fundamental level of detail possible. Most matches will be approximate, so for each closest match, make additional size adjustments as necessary.
- **Formalized Methodology:** Formally applying the guidelines of Function Points or any other standard sizing approach to the project specifications to derive the project size.

- **Indicative Sizing:** Use of automated tools and/or pre-defined algorithms such as counting the number of subsystems or classes and converting them to function points.
- **Statistical Sizing:** Provides a range of potential sizes that is characterized by least, likely, and most.
- **Planning Poker:** This methodology uses a blend of Expert opinion and Analogies identified above. This method is used when the existing requirements have been expressed in the form of User Stories or Features. A note of caution in that the size values derived from a team-centric planning poker session are only valid for the team performing the sizing exercise.

Step 5. Prepare Baseline Estimate

Budget and schedule are derived from estimates, so if an estimate is not complete or does not accommodate uncertainty, the resulting schedules and budgets are likely to be affected. Given the importance of the estimation task, developers who want to improve their software estimation skills should understand and embrace some basic practices. Firstly, trained, experienced, and skilled people should be assigned to size the software and prepare the estimates. Second, it is critically important that they be given the proper technology and tools. And third, the project manager must define and implement a mature, documented, and repeatable estimation process.

Many Agile practitioners eschew formal estimation practices as there is an assumption that the preliminary requirements are too abstract or ambiguous. However, this is not generally the case for mission-critical applications. Regardless the quality and availability of formal requirements, one can reasonably argue that a preliminary estimate can and should always be generated. The emphasis is upon the uncertainty expressed by the estimate range boundaries. This notion has long been understood in the community and was first expressed by Dr. Barry Boehm as the "Cone of Uncertainty" (*Software Engineering Economics,* 1990). While Agile provides development methodologies to counter uncertainty, formal estimation techniques are agnostic to software delivery models.

- **Bottom-Up Estimating:** Also referred to as "grassroots" or "engineering" estimating, this technique entails decomposing the software to its lowest levels by sprint, function or task and then summing the resulting data into work elements. This approach can be very effective for estimating the costs of smaller systems. It breaks down the required effort into traceable components that can be effectively sized, estimated, and tracked; the component estimates can then be rolled up to provide a traceable estimate that is comprised of individual components that are more easily managed. You thus end up with a detailed basis for your overall estimate.
- **Software Cost Models:** Different cost models have different information requirements. However, any cost model will require the user to provide at least a few — and sometimes many — project attributes or parameters. This information describes the project, its characteristics, the team's experience and training levels, and various other attributes the model requires to be effective, such as the processes, methods, and tools that will be used. There may be some concern that modern software estimation models do not reflect Agile development processes. Figure 7, "Parametric Estimation for Agile Projects," presents the way that Agile projects should be approached in parametric cost estimation. The main inputs to consider are:

- **Backlog Size:** The size of the backlog functionality to be implemented, which can be expressed in any of the sizing techniques previously described
- **Project Characteristics:** Attributes of the project describing the executing platform, type of application, development method and standards, among others
- **Team Dynamics:** These include development team size, sprint duration and velocity

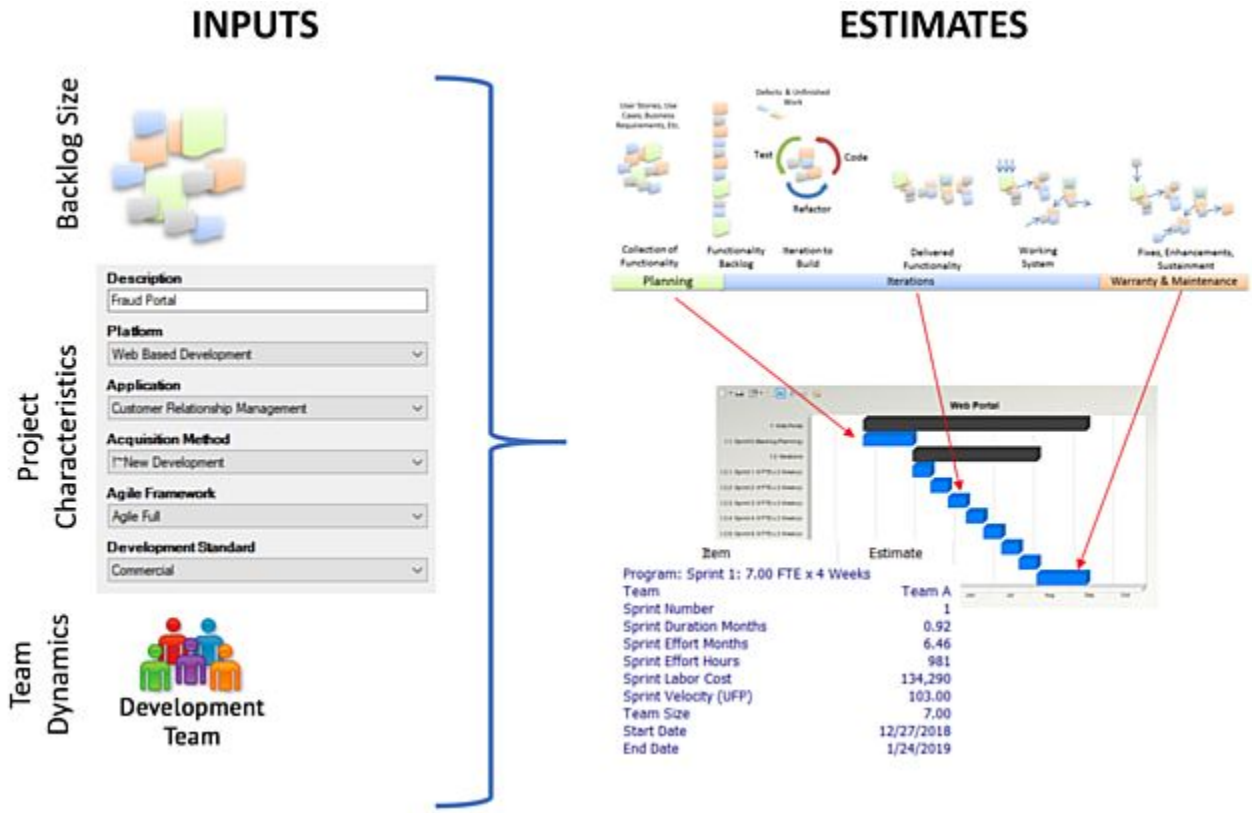

Figure 7. Parametric Estimation for Agile Projects

Many current models include an Agile Planner to specifically account for the aspects of Agile development.

Parametric cost models provide a means for applying a consistent method for subjecting uncertain situations to rigorous mathematical and statistical analysis. Thus, they are more comprehensive than other estimating techniques and help to reduce the amount of bias that goes into estimating software projects. They also provide a means for organizing the information that serves to describe the project, which facilitates the identification and analysis of risk.

A cost model uses various algorithms to project the schedule and cost of a product from specific inputs. Those who attempt to merely estimate size and divide it by a productivity factor are sorely missing the mark. The people, the development environment, and the process are all key components of a credible estimate and resultant successful software

project. Cost models range from simple, single formula models to complex models that involve thousands of calculations.

- **Delphi and Wideband Delphi:** You can further offset the effects of these biases by implementing the Delphi estimation method, in which several expert teams or individuals, each with an equal voice and an understanding up front that there are no correct answers, start with the same description of the task at hand and generate estimates anonymously, repeating the process until consensus is reached. Planning Poker can provide relative sizes that can be very useful for Agile programs.
- **Activity-Based Estimates:** Another way to estimate the various elements of a software project is to begin with the requirements of the project and the size of the application, and then, based on this information, define the required tasks, which will serve to identify the overall effort that will be required.

Step 7. Quantify Risks and Risk Analysis

It is important to understand what a risk is and that a risk, in itself, does not necessarily pose a threat to a software project if it is recognized and addressed before it becomes a problem. This is one area where many Agile developments miss the mark. Risk is characterized by an impact on mission, loss of time, quality, money, control or understanding, and so on. The loss associated with a risk is called the risk impact.

Management, stakeholders, and developers must have some idea of the probability that the event will occur. The likelihood of the risk measured from 0 (impossible) to 1 (certainty) is called the risk probability. When the risk probability is 1, then the risk is called a problem, since it is certain to happen. It can also be looked at as either "Very Low" to "Extremely High"

For each risk, we must determine what we can do to minimize or avoid the impact of the event. Risk control involves a set of actions taken to reduce or eliminate a risk.

Risk management enables you to identify and address potential threats to a project, whether they result from internal issues or conditions or from external factors that you may not be able to control. Problems associated with sizing and estimating software potentially can have dramatic negative effects. The key word here is potentially, which means that if problems can be foreseen and their causes acted upon in time, effects can be mitigated. The risk management process is the means of doing so.

The risk management process usually deals with opportunities as well. Opportunities are potential events that can benefit the project. The same techniques applied to risk management can also be applied to opportunity enhancement. For example, the probability and impact (in this case, positive impact) of an opportunity should be calculated and the organization must determine what can be done to maximize the probability of the opportunity and to exploit its benefits.

ISO 9001 identifies how to assess risks and opportunities for any program. Another excellent source for implementing effective risk management is "Software Engineering Risk Analysis and Management and Application Strategies for Risk Analysis" by Dr. Robert Charette.

Step 6. Estimate Validation and Review

At this point in the process, your estimate should already be reasonably good. It is still important to validate your methods and your results, which is simply a systematic confirmation of the integrity of an estimate. By validating the estimate, you can be more confident that your data is sound, your methods are effective, your results are accurate, and your focus is properly directed.

There are many ways to validate an estimate. Both the process used to build the estimate and the estimate itself must be evaluated. Ideally, the validation should be performed by someone who was not involved in generating the estimate itself, who can view it objectively. The analyst validating an estimate should employ different methods, tools and separately collected data than were used in the estimate under review.

When reviewing an estimate, you must assess the assumptions made during the estimation process. Make sure that the adopted ground rules are consistently applied throughout the estimate. Below-the-line costs and the risk associated with extraordinary requirements may have been underestimated or overlooked, while productivity estimates may have been overstated. The slippery slope of requirements creep may have created more uncertainty than was accounted for in the original estimate.

A rigorous validation process will expose faulty assumptions, unreliable data and estimator bias, providing a clearer understanding of the risks inherent in your projections. Having isolated problems at their source, you can take steps to contain the risks associated with them, and you will have a more realistic picture of what your project will actually require in order to succeed.

Despite the costs of performing one, a formal validation should be scheduled into every estimation project, before the estimate is used to establish budgets or constraints on your project process or product engineering. Failing to do so may result in much greater downstream costs, or even a failed project.

Step 8. Generate A Project Plan

The process of generating a project plan includes taking the estimate and allocating the cost and schedule to a function and task-oriented work breakdown structure. For Agile deliveries, the project plan will be a “backlog” of desired capabilities in the order of “importance” to the product owner. For “incremental” Federal programs this may not be practical. For “transformational” Federal programs, this is a requirement.

Some managers, mainly due to lack of experience, are not able to evaluate what effects their decisions will have over the long run. They either lack necessary information, or incorrectly believe that if they take the time to develop that information the project will suffer as a result. Other managers make decisions based on what they think higher management wants to hear. This is a significant mistake. A good software manager will understand what a project can realistically achieve, even if it is not what higher management wants. His job is to explain the reality in language his managers can understand.

Both types of “problem manager,” although they may mean well, either lead a project to an unintended conclusion or, worse, drift down the road to disaster. Within the Agile community this problem is mitigated by close collaboration with a Product Owner, as well as by incremental deliveries. However, even Agile deliveries experience the uncertainty and volatility that arise from technology and personnel capabilities.

Software management problems have been recognized for decades as the leading causes of software project failures. In addition to the types of management choices discussed above, three other issues contribute to project failure: bad management decisions, incorrect focus, and destructive politics. Models such as SEER-SEM handle these issues by guiding you in making appropriate changes in the environment related to people, process, and products.

Step 9. Document Estimate and Lessons Learned

Each time you complete an estimate and again at the end of the software development, you should document the pertinent information that constitutes the estimate and record the lessons you learned. By doing so, you will have evidence that your process was valid and that you generated the estimate in good faith, and you will have actual results with which to calibrate your estimation models.

Be sure to document any missing or incomplete information and the risks, issues, and problems that the process addressed and any complications that arose. Also document all the key decisions made during the conduct of the estimate and their results and the effects of the actions you took. Finally, describe and document the dynamics that occurred during the process, such as the interactions of your estimation team, the interfaces with your clients, and trade-offs you had to make to address issues identified during the process.

You should conduct a lessons-learned session as soon as possible after the completion of a project while the participants' memories are still fresh. Within the Agile community, this is typically done at the end of each sprint during a retrospective activity. These lessons-learned sessions should not be limited to only the development issues but expanded to include a review of the estimation activities.

Estimation reviews may range from two team members meeting to reach a consensus about the various issues that went into the estimation process, to highly structured meetings conducted by external facilitators who employ formal questionnaires. No matter what form it may take, it is always better to hold a lessons-learned meeting than not, even if the meeting is a burden on those involved. Every software project should be used as an opportunity to improve the estimating process for the next set of projects.

Step 10. Track Project Throughout Development

In-process information should be collected, and the project should be tracked and compared to the original plan. Most Agile methods provide a mechanism for monitoring development progress. Most use daily stand up sessions, task and story boards, burndown and burnup charts. If projects are varying far from their plans, revised estimates should be prepared. Ideally, the following attributes of a software project would be tracked:

- Cost, in terms of staff effort, phase effort and total effort
- Defects found or corrected, and the effort associated with them
- Process characteristics such as development language, process model and technology
- Project dynamics including changes or growth in requirements or code and schedule
- Project progress (measuring performance against schedule, budget, etc.)
- Software structure in terms of size and complexity

Traditional earned value, when combined with quality and growth factors, can be used to forecast completion very accurately and flag areas where managers should be spending time.

Software cost estimation can help ensure useful results by incorporating a process that is standardized and repeatable. Several of the steps we have discussed, particularly those that do not result directly in the production of the estimate (Steps 1, 6, and 7) are often deferred or, worse still, not performed at all, often for what appear to be good reasons, such as a lack of adequate time or resources or a reluctance to face the need to devise a plan if a problem is detected. Sometimes one may have more work than can be handled, and such steps don't seem absolutely necessary. Sometimes management is reluctant to take these steps, not because the resources are not available, but because managers do not want to really know what they may learn as a result of scoping their estimates, quantifying and analyzing risks, or validating their estimates. This can be a costly attitude, because in reality, every shortcut results in dramatic increases in project risks.

7. Significant Reasons for Software Cost Growth

The Department of Defense (DoD) can leverage best practices of iterative development even in mission critical software systems. The previously-cited study performed by the DSB described the impact of software size growth. However, there are other factors that contribute to the growth of software cost. Major reasons for software cost growth are identified below along with an approach to mitigation of these issues.

- **Scope Creep/Requirements Growth:** Scope creep /requirements growth is the result of not having a clearly defined scope and/or not tracking and declaring changes to scope. Sometimes this is due to starting the development too early in the project, not spending enough time on requirements analysis, and unrealistic project expectations. DoD programs can address this issue by an extensive review of the Cost Analysis Requirements Document (CARD) and making a comparison of the expectations against recent programs. Additionally, a robust change management process should enforce the appropriate level of change impact analysis to measure the impact of requirement changes on the cost and schedule baselines and the potential need to re-estimate.
- **Poor Input to Estimate:** This refers to poor quality requirements or objectives, or an incorrect measure of the software size, leaving the estimate open to risk and scope variation. The estimate is "built on sand" but the estimator does not factor for this or clearly express the uncertainty as a 3-point estimate. DoD programs can address this issue by employing a robust risk assessment methodology and a competent parametric cost estimation tool.
- **Failure to Clearly Define the Initial Scope:** Items may be omitted and/or there may be a failure to declare assumptions and ground rules. This will make the project vulnerable to scope creep or conflict later in the project as all parties argue over what is in scope and what is out of scope. DoD programs can address this issue by clearly defining ground rules and assumptions for the estimate.
- **Unrealistic Expectations and Assumption:** Optimistic assumptions from single source "experts," claiming benefit from improvements in software development processes often drive unrealistic expectations. DoD programs can address this issue by utilizing software estimating subject matter experts to fully vet all expectations and assumptions. In addition, benchmarking the estimate against relevant industry

data or internal historical projects can provide insight into the reasonableness and feasibility of the cost, schedule and quality expectations.

- **Failure to Declare, Track and Reduce Risk and Uncertainties:** The project may then be vulnerable to unnecessary risk. DoD programs can address this issue by utilizing a well defined and successful software estimating process.
- **Lack of Internal Peer Review:** Software estimates often fail to benefit from peers' experience. A review is a simple and cost-effective way to improve estimate "Quality." DoD programs can address this issue by holding peer reviews and utilizing several cross-check methodologies. Checklists and workflow templates should be used to drive the review process, ensuring nothing is missed and the review findings are documented and correctly fed back for implementation.
- **Lack of Estimation Experience:** Estimates generated by personnel who may not have sufficient experience in estimating or in the subject being estimated often lead to cost growth. DoD programs can address this issue by utilizing a team of tested software estimating experts.
- **Failure to Consider Environmental Factors:** Undeclared and untracked elements of the software development environment can change. These changes directly affect the cost estimate. Such changes include planned utilization of heritage code, changes in the developmental methodology, programming language or technologies, tools and personnel. Some changes, such as the unexpected and "dramatic" effects of the loss of key individuals, change of supplier, or an increase in organizational complexity can have major impacts on the cost estimate. DoD programs can address this issue by utilizing a team of tested software estimating experts and monitoring the progress of the program.
- **Failure in the Estimation Tool/Process:** The tools or processes may be unable to deliver the desired accuracy. It is essential to measure and declare the estimate tolerance of any tools used. DoD programs can address this issue by utilizing the SRDR data set, other commercial data sets, and models as appropriate. Continuous improvement of the estimation tools and processes is achieved by feeding actuals and lessons learned back into the estimation process.
- **Estimating to a Target Assumption:** It is not uncommon for the cost of a program to be predetermined. That is, there is an expectation for what the estimate result should be before it is even created. This can be caused by funding limitations or restrictions, or assumptions by the proposing organization as to Price to Win. The best way to ensure "target estimates" are avoided is to prepare estimates using an independent team that does not have visibility has not been exposed to a prescripted outcome.

8. Managing Modern Software Development Programs

Measure the Right Thing

For federal Agile software development programs, the basic metrics are burndown charts (sprints, features/stories, and or epics), velocity, and backlog. These metrics aid planning and inform decisions about process improvement.

- **Backlog:** A repository for all upcoming and uncompleted work. It consists of features to address user needs and deliver business benefits, as well as architectural features required to build the product.
- **Velocity:** How many "software units" (which could be measured in stories, features, function points or story points) the team typically completes in an iteration (a.k.a sprint). This number should only be used to plan iterations. Comparing team velocities is nonsense, because the metric is based on non-objective estimates. Treating velocity as a success measure is inappropriate and making a specific velocity into a goal distorts its value for estimation and planning. In addition, using velocity as a metric to compare productivity across teams can result in induced velocity inflation or story point inflation, where teams will arbitrarily increase their story counts just to show more work completed. Figure 8 below illustrates the concept of velocity/story point inflation:

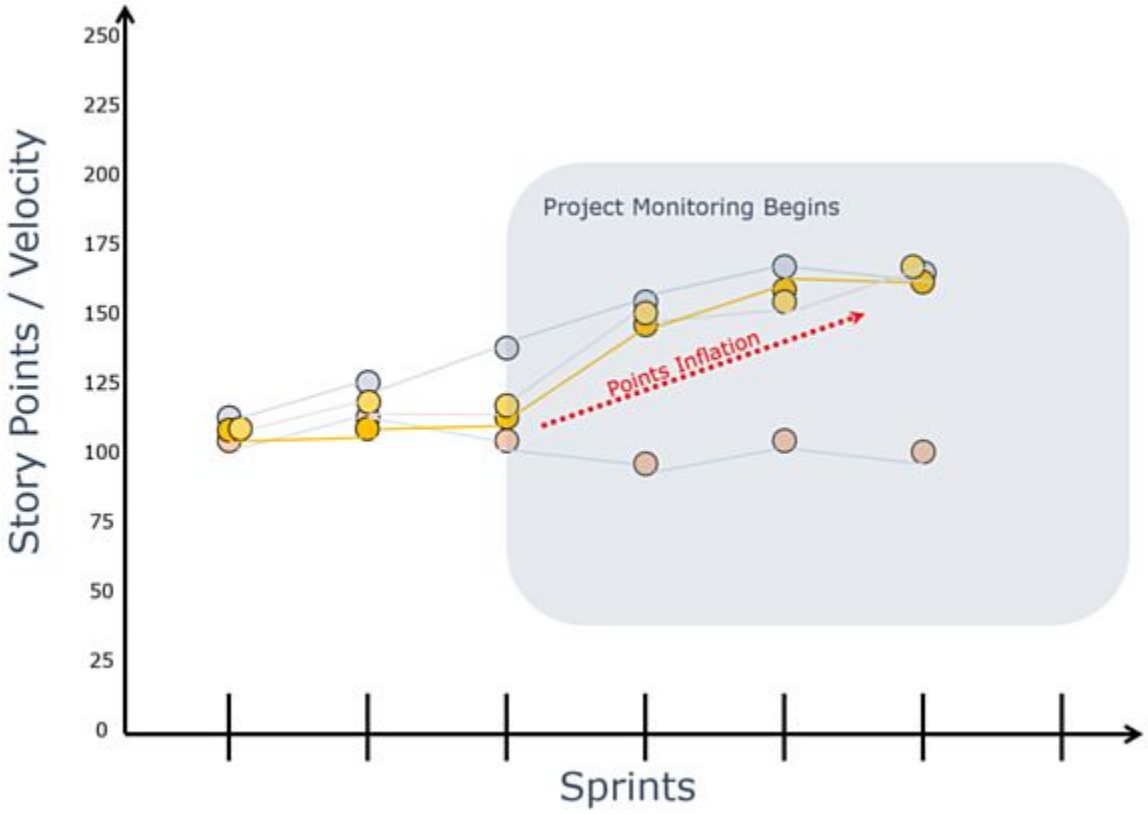

Figure 8. Story Points/Velocity Inflation

- **Burndown Charts:** Teams organize development into time-boxed development periods. The team forecasts how much work they can complete during a development period. A team that consistently meets its forecast is a compelling advertisement for Agile success. Often these charts are maintained at the sprint, feature, and epic level. Traditional metrics such as defect discovery and removal can and should also be applied to Agile programs.

Manage Expectations / Set Realistic Time Frames.

Defense organizations often set overly optimistic schedules. Set realistic expectations up front and keep expectations current in the minds of project team members. Within an Agile iteration, the failure to reach an expected delivery of a feature or story is part of the selected methodology; unfinished work goes back onto a backlog. However, this can also be indicative of faults in the user story point counting, points inflation, or unrealistic expectations of early velocity. All these need to be managed.

Align the Work Streams

You need to identify, align and continuously monitor work streams to ensure smooth progress throughout the organization. Understand dependencies between work streams during project plan development to ensure proper resource allocations and project time frames. Continue to monitor the interdependencies throughout the project.

Seek Objectivity

Assessments conducted by an outside expert add both value to the project implementation and protection against the high cost of failure. Expertise delivers the know-how and the objective oversight needed to overcome organizational roadblocks. It also provides you with peace of mind. Assessments should be conducted by an executive project manager or software implementation expert who has managed enough projects successfully to know how to recognize subtle indicators, intervene to accommodate the situation, and adjust expectations accordingly.

Key Questions to Assess the Quality of the Agile Progress

- **Size Metrics:** "Are the user stories consistent and do they follow the basic structure of 'As a__, I Want, So That…'?"
  User stories should also meet the following characteristics:
  - **Independent:** The user story should be self-contained and include no inherent dependencies with other user stories
  - **Negotiable:** User stories should be open to negotiation with the product owner
  - **Valuable:** The user story must deliver value to the stakeholders and users
  - **Estimate-able:** The user story should be measurable (i.e. sizeable) so that an estimate for it can be developed
  - **Small:** user stories should be of relatively small size, so they can be managed, implemented, planned, prioritized with a good level of accuracy.
  - **Testable:** user stories should include the corresponding acceptance criteria so that they can be verified.

  A second question with respect to sizing is, "Are all the teams using the same story development and 'pointing' process?" If there are concerns about the above, function points may be a better sizing approach.
- **Agile Methods:** "Is the development process using the proper Agile method for work to be performed?"
  Scrum is for complex projects (Schwaber and Beddle); Kanban for delivering as "value

stream" (Mary and Tom Poppendieck); XP, (also known as pair programming), to deliver high-quality software quickly and continuously (Kent Beck). When the developer is using Hybrid Agile, it needs to be clear if multiple Agile methods are being used or if it simply means a scrum method is being modified. The latter is the most common usage.

"Across all health factors, a hybrid mixture of Agile and Waterfall methods produced higher scores than either Agile or Waterfall methods alone," according to Bill Curtis, Senior Vice President and Chief Scientist for CAST, who oversees the administration of the CRASH report. "These results suggest that for business-critical applications the value of agile and iterative methods is enhanced by the types of up-front architectural and design activity characterized by Waterfall methods."

A typical hybridization is to formalize upfront requirements – provide time-boxed development – and then big-bang system testing. If this is the plan, then the richness of requirements discovery is lost as is the improved quality of early fault detection. Hybrid can often mean an ad hoc (or corrupted) Scrum delivery mode. However, this process has been demonstrated to work. The current trend with Hybrid aligns to a Water-Scrum-Fall process where the Water is the epic-level requirements that evolve in the backlog planning, Scrum is incremental design/code/refactor, and Fall as the final user acceptance testing or IOT&E (not systems testing).

- **Agile Integrity:** "Is the program deviating from the Twelve Agile Principles?"
  A critical component to successful Agile is the principle of "self-organized teams produce the best architecture, requirements, and design." This can be accomplished by just-in-time team members; however, it breaks up the cohesiveness and taints the Agile values of "individuals and interactions."

  Organizations should strive to obligate team members for a full release (multiple sprints). The quality improvements found in Agile often come from those members with an "outside view" (Nobel Prize winner - Daniel Kahneman). The outside view allows for alternative perspectives on a problem through multi-dimensional cooperation.

- **Agile Delivery:** The acquisition manager needs to recognize that Agile is a development mindset (not a methodology) created by practitioners trying to resolve the iron triangle of scope, schedule, and resources.

- **Velocity:** "Is velocity based on a historical baseline of the program?"
  What is the contingency for missing the velocity goal on a sprint? (They should be averaged before any changes.) Do they allow stories to be repointed? (This can cause points inflation and is very common.)

  In the absence of a baseline velocity, an industry standard velocity (i.e. what an average industry organization can achieve in a sprint, given certain project characteristics such as platform, application type, programming language, etc.) should be used for estimation purposes. Basing the estimate in an unrealistic or incorrect velocity is a recipe for failure.

- **Governance:** "Is there an experienced Scrum Master?"
  This is a soft firewall between the team and the PMO and the team and the product owner.

- **Expectations:** "Is the team promising faster schedule and cheaper cost?" Research consistently shows Agile is not cheaper. It, like any other software development can be plagued with cost and schedule overruns as well as not delivering the desired product unless it is properly planned and managed through measurement. "If you fail to plan, you plan to fail" (PSM).

- **Agile Manifesto :** "Are the 12 Principles of the Agile Manifesto being embraced or is it simply the method du-jour?"

9. Measuring and Tracking Performance on Modern Federal Software Programs

Agile Software Development Programs

Software development methodologies have evolved since the early 1960s. These methodologies have varied from no rules to very structured approaches. Typical DoD software intensive acquisition projects have followed a highly structured (Waterfall) process. Unfortunately, highly structured programs are rarely the case in information technology (IT) projects. To reduce cost overruns and project failures, many DoD acquisition programs are moving to Hybrid Agile development approaches.

It is not the intention of this paper to fully describe the Agile process. One excellent source for this description is found in the MITRE *Handbook for Implementing Agile in Department of Defense Information Technology Acquisition* (2010). The Agile Manifesto describes the overarching beliefs of Agile software development (Beck et al, 2001):

> "We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
>
> - Individuals and interactions over processes and tools
> - Working software over comprehensive documentation
> - Customer collaboration over contract negotiation
> - Responding to change over following a plan."

Often, traditional earned value approaches do not deal sufficiently with the idiosyncrasies of software intensive programs. This can be especially true when Agile software development processes are employed. However, successful management of Agile software development programs can be achieved by focusing on establishing the requirements, developing a reliable baseline estimate for cost and schedule, selecting effective software metrics (that include both quantity and quality measures), and using analytic processes to project cost and schedule based on actual performance.

Contrary to a popular misconception, Agile, in practice, is a disciplined methodology. Consequently, a reasonable life cycle estimate can be developed, and an execution schedule can be presented. For a hardware system, e.g. a missile, applying EVM can be straightforward in that both quantity (how many units come off the line) and quality (how many units pass test) can be measured.

For a system with a quantity of one, e.g. a satellite or software intensive system, the measure of "quality" can become problematic. In a satellite system one might measure progress towards meeting weight and mass requirements as a measure of quality. In a SLOC or function point based system, measuring defect discovery and removal or defect density (e.g. defects per function point), might be a good measure of quality. The question becomes "what to measure in an Agile/iterative software development program?" The basic tenets of an EVM program are presented in Figure 9 below.

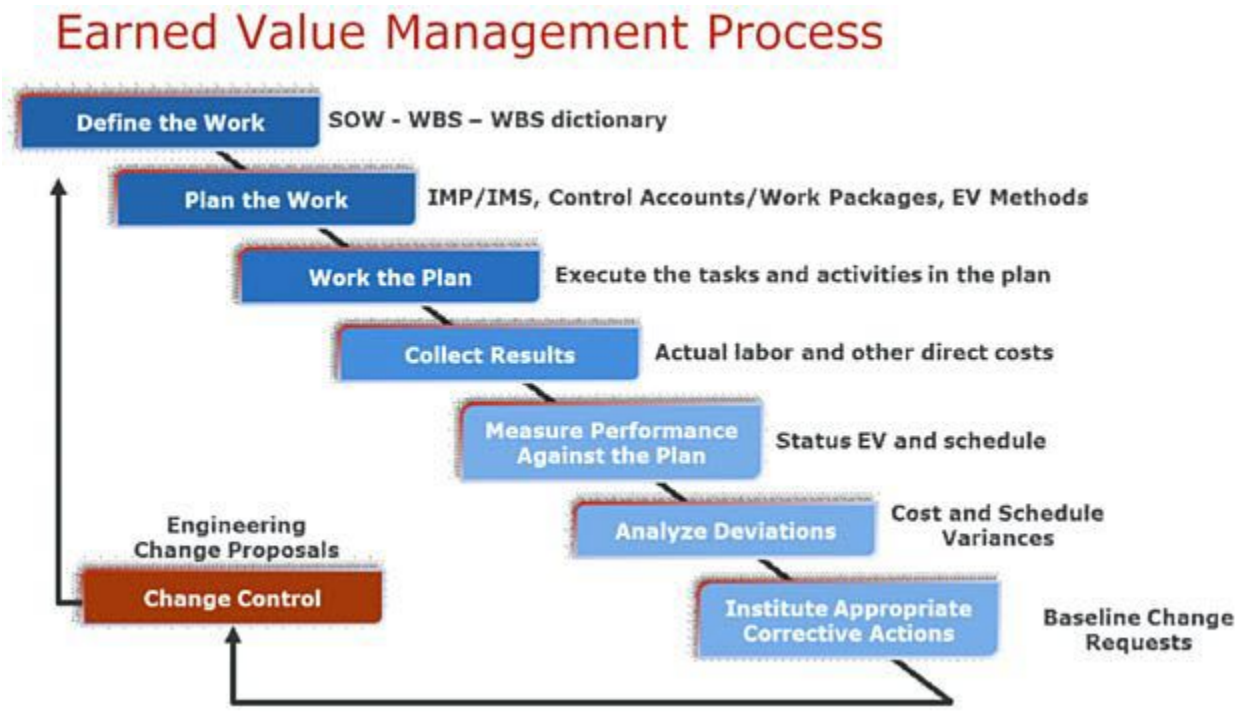

Figure 9. Earned Value Management Process

In practice, many DoD programs are executing what might be called a Hybrid Agile acquisition strategy. This is sometimes referred to as "Water-Scrum-Fall". Figure 10 below, "Typical Hybrid Agile Development," presents a typical Hybrid Agile approach. As this figure shows, many up-front activities, such as requirements documentation, are executed under a structured (Waterfall) approach, while only the actual coding and *sometimes* testing are done using an Agile method. We are also finding that many of these contracts are fixed price or labor rate contracts.

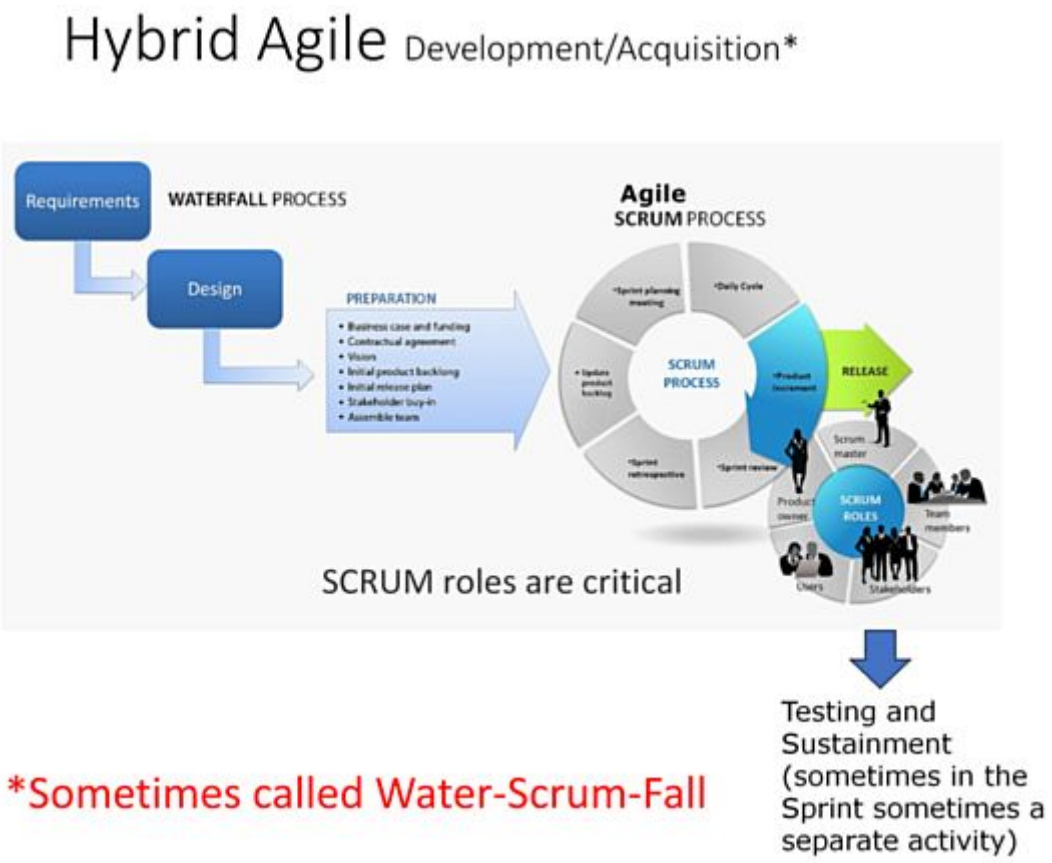

Figure 10. Typical Hybrid Agile (Water-Scrum-Fall) Development

Within the Agile software development context, the sprint is the fundamental work unit. But this is generally at a level that is too detailed for the application of EVM processes. In the Agile software development cycle, sprints contain stories to be completed, stories roll up into epics, epics into themes and themes roll up into releases. Some projects may substitute stories for features and epics for themes.

Likewise, the phraseology may be the only difference. This concept is presented in Figure 11, "EVM Building Blocks for an Agile Program." User stories or features are the key elements for many agile programs. Earned value is usually reported at the feature/user story level rather than the sprint level due to the large number of sprints. When done correctly, there is a direct correlation between the work packages described in the Contractors Work Breakdown Structure (CWBS) and the features/user stories. This is how software requirements are mapped into an EVM structure.

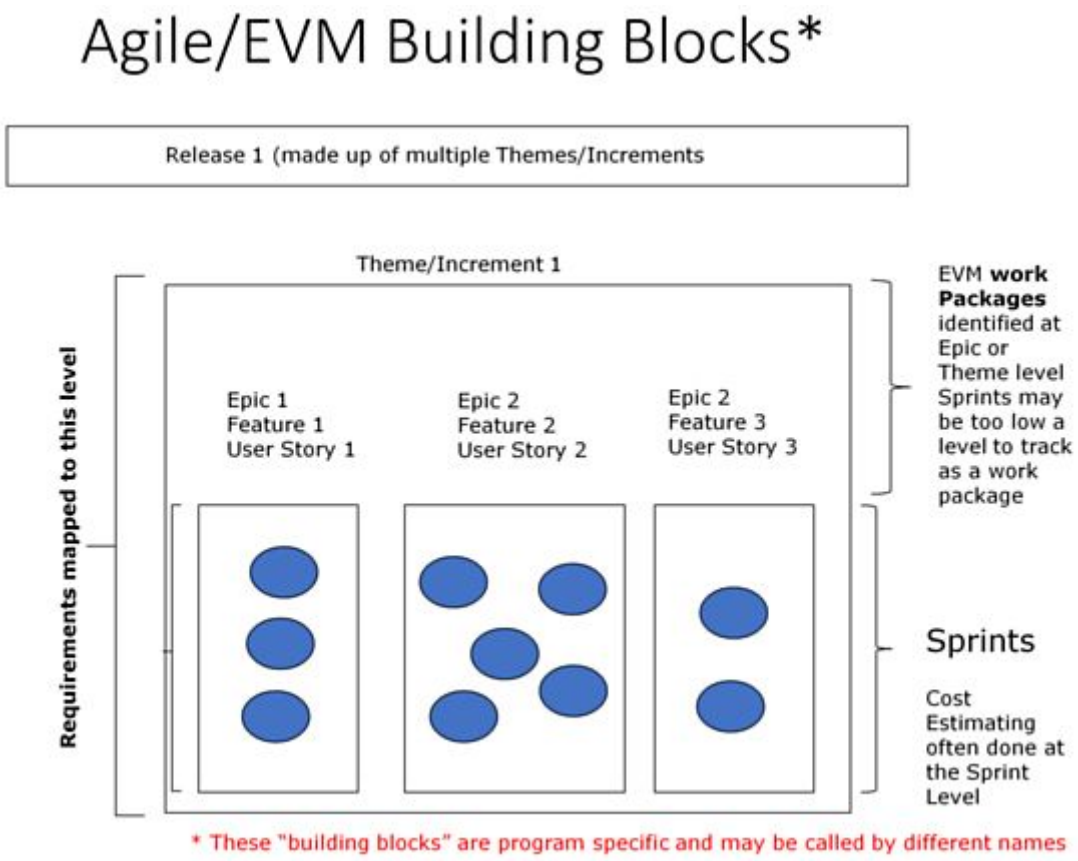

Figure 11. EVM Building Blocks for an Agile Program

Figure 11 requires some additional explanation. Figure 11 is intended to show that earned value is generally captured at a level above the sprint. It also implies that a story or feature is made up of multiple sprints. In the commercial/pure Agile world, there are generally multiple stories implemented within a sprint. In large federal programs using a Hybrid Agile approach, the story/feature is generally defined in a much larger manner, thus a story may require multiple sprints for its implementation.

A key point here is that all stories are not the same. Some stories are in actuality desired "features" and take multiple sprints due to technical difficulty or complexity. These are broken into smaller more manageable (and testable) stories. The typical rule of thumb is that a story should be small enough to complete within a time-box (sprint).

To identify the complexity of a story/feature, generally, a numeric rating schema (points) are assigned to each story/feature using a scale: small-medium-large, Fibonacci sequence, 1-10-100, or modified Fibonacci like1-2-3-5-8-13-40-100, or some other scaling method. Function points can also be applied to measure the size and complexity of a story/feature.

Where stories represent a basic "measurable" work package and size scale represents the technical complexity of each work package. Often "velocity," the rate at which stories are completed, is an additional measure used for scheduling and sprint planning.

Earned value is usually reported at the user story level rather than the sprint level. When done correctly, there is a direct correlation between the work packages described in the Contractors

Work Breakdown Structure (CWBS) and the features/user stories. Ideally there will be a one-to-one correspondence.

If there is a plan and product(s), EVM can be applied. The product backlog defines the product, and sprints are used to time phase the work. The status of each sprint is rolled up to a level, which in this example is the Control Account (CA). Below the epic is the feature at the Work Package (WP) level, which breaks the epic into functional packages. Features are decomposed into stories and story points. Sprints are statused by, in this case, stories and story points, which are maintained in an Agile program management tool.

As stories are completed the percent complete is rolled up to the epic level

The key Agile/EVM integration touch points are at the organization level, in the planning, budgeting, scheduling process, and in the analysis and management reports.

Figure 12. Agile EVM Touch Points

The Agile EVM interconnection is further shown in the chart below.

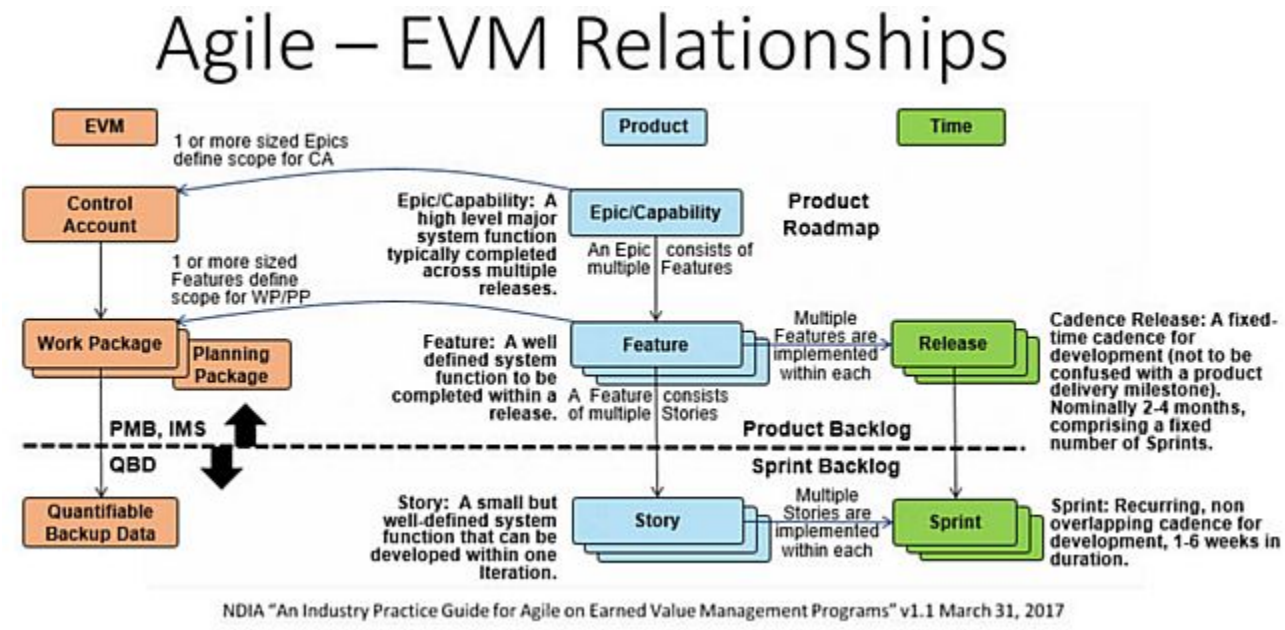

Figure 13. Agile EVM Relationships

Key Agile management charts are presented below.

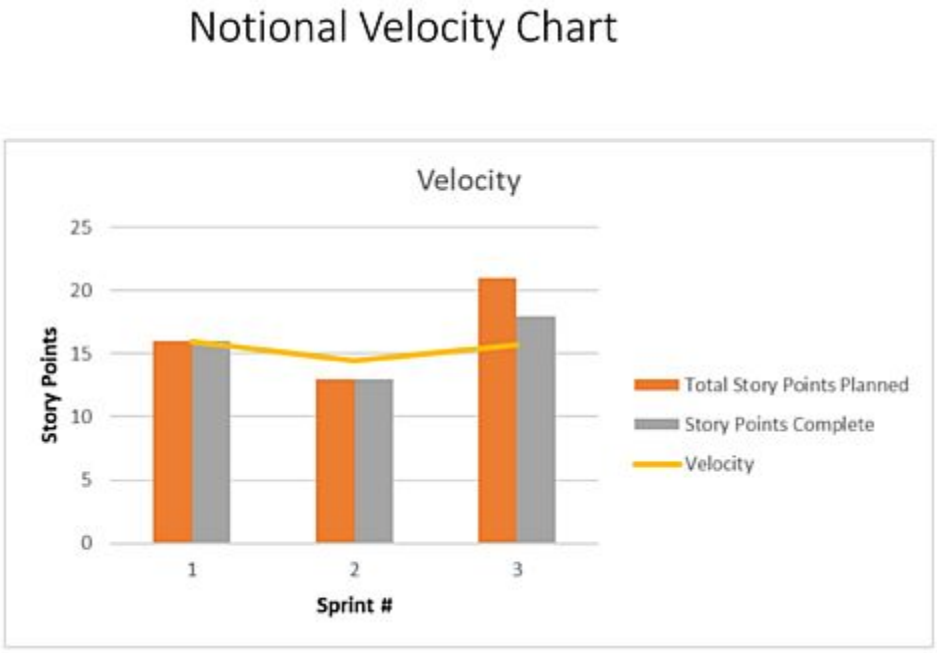

Figure 14. Agile Velocity

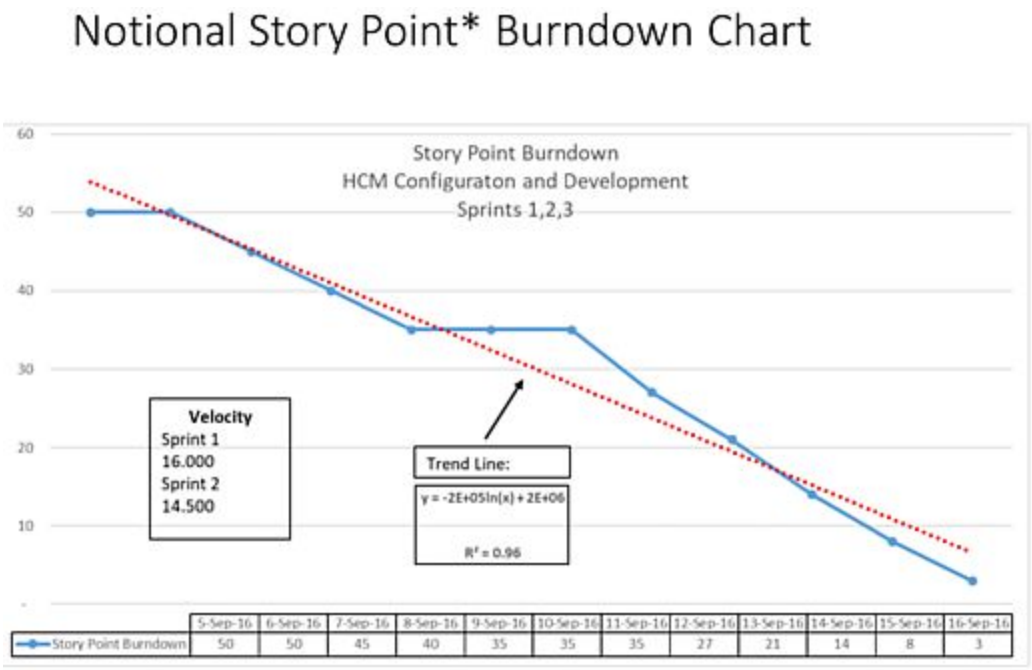

Figure 15. Story Point Burn Down

10. Conclusion

Many studies on project success are conducted yearly. Some of them have been referenced in this paper (Standish and Gartner) and they all converge on one fact: A large proportion of the IT projects either undergo challenges or fail. Inaccurate estimation and project tracking processes are certainly big drivers behind these numbers. The adoption of Agile practices in federal programs is growing and this is certainly changing the way we approach software development in the federal world. While the world continues to adopt these Agile practices, it is important to remember that projects still need good estimates and robust tracking mechanisms.

Modern federal software development programs are evolving into incremental development practices and some of them are starting to adopt the Agile development framework. Some of these programs are fully adopting the framework while some others are "tropicalizing" it and executing what is commonly referred to as Hybrid Agile. Regardless of level of Agile adoption, these federal programs can benefit from using a formal estimation and Earned Value Management (EVM) processes. The authors of this paper agree that good estimation and tracking mechanisms can bring the following benefits to federal programs:

- Robust cost baseline based on a formal estimation process: Using formal estimation practices such as function points and parametric estimation allows the generation of achievable and trustable estimates. These estimates then become the baseline from which performance is measured. Your project is committed and measured against a realistic estimate.

- Single system to track work, schedule and cost: Earned Value Management. This is probably the best benefit from using Earned Value Management in Agile projects: EVM can measure the amount of work completed (user stories, features and epics completed throughout the sprints), the program budget that has been burned and forecast the cost and completion date; The actual

performance of the project is compared against the estimated baseline for project tracking. performance indicators (CPI and SPI) are used to measure deviations from the baseline.

- Earned Value Management is compatible with Agile: The work distribution among the sprints can be rolled up to user stories, features and epics and eventually be rolled into EVM building blocks, allowing the capture of metrics and the application of the earned value system. In essence, federal programs can get the benefits from traditional EVM while executing in an Agile environment.

- Early detection of risks and opportunities. EVM is a real time tracking mechanism for projects. Periodically, the project actuals are analyzed against the estimated baseline and deviations from the baseline are assessed. Negative deviations can help to reveal risks in real time for which mitigation actions can be implemented before the risk evolves into a problem. Positive variations (e.g. when the actual performance is better than the planned baseline) can represent opportunities that, if adequately boosted, can potentially represent savings in cost and the ability to secure milestones.

Federal programs need to adopt continuous iterative development best practices (continuing through sustainment) for software while keeping the best management and tracking processes from the traditional development world. Even though the Agile practices are proposing a reduction in the amount of formal estimation and project tracking and control, the lack of these formalities can represent a risk for federal programs. The authors of this paper recommend taking the best practices from Agile and leveraging those with the management and estimation practices that have proven to be successful for many years.

Author Biographies

Dan Galorath

During his over three decades in the industry, Daniel D. Galorath has been solving a variety of management, costing, systems, and software problems for both information technology and embedded systems. He has performed all aspects of software development and software management. One of his strengths has been reorganizing troubled software projects, assessing their progress applying methodology and plans for completion and estimated cost to complete. He has personally managed some of these projects to successful completion. He has created and implemented software management policies and reorganized (as well as designed and managed) development projects. His company, Galorath Incorporated, has developed the SEER applications, methods, and training for 1) software, 2) hardware, electronics & systems, 3) information technology, 4) manufacturing, 5) systems engineering, 6) space systems cost, schedule, risk analysis, and management decision support. He is one of the principal developers of the SEER-SEM software evaluation model. His teaching experience includes development and presentation of courses in Software Cost, Schedule, and Risk Analysis; Software Management; Software Engineering; to name a few. Mr. Galorath is a sought-after speaker. Among Mr. Galorath's published works are papers encompassing software cost modeling, testing theory, software life cycle error prediction and reduction, and software and systems requirements definition. Mr. Galorath was awarded the 2009 Society of Cost Estimation and Analysis (SCEA) Lifetime Achievement award for contributions to the industry; and the International Society of Parametric Analysts (ISPA) Freiman Award, lifetime achievement award, awarded to individuals

who have made outstanding contributions to the theoretical or applied aspects of parametric modeling. Dan is a contributing author of "IT Measurement, Advice from the Experts" (Prentice Hall) and Mr. Galorath's is the primary author of the book "Software Sizing, Estimation, and Risk Management: When Performance Is Measured Performance Improves." His widely read blog, Dan Galorath on estimating, covers estimation, planning, measurement, control and risk analysis.

Robert (Bob) Hunt

Bob Hunt has over 40 years of cost estimating and analysis experience. He received his Society for Cost Estimating and Analysis (SCEA) Certification in 1991. He has served in senior technical and management positions at Dulos Incorporated (President), Galorath Federal Incorporated (President), Galorath Incorporated (Vice President for Services), CALIBRE Systems (Vice President), CALIBRE Services (President), SAIC (Vice President), the U.S. Army Cost and Economic Analysis Center (Chief of the Vehicles, Ammunition, and Electronics ICE Division), U.S. Army Directorate of Cost Analysis (Deputy Director for Automation and Modeling), and other Army analysis positions. He is the author of multiple technical papers published for ICEAA, IEEE, DCAS, and other professional journals. His published works include Price To Win, Should Cost, Earned Value, and Agile Estimating. He has served as a track chair and technical presenter for multiple SCEA/ISPA/ICEAA Conferences.

Mr. Hunt has provided information technology systems and software program assessments, and IT and software cost estimating for commercial and federal clients including the U.S. Army, Customs and Border Patrol, the Department of Defense, NASA, and various commercial clients. In addition to his ICEAA Treasurer responsibilities, Mr. Hunt has held leadership positions and made technical presentations for the American Institute of Aeronautics and Astronautics (AIAA), served as the Chairman of the Economics Technical Subcommittee of the AIAA, the National Association of Environmental Professionals (NAEP), and the IEEE.

Mr. Hunt received his Master's Degree from Virginia State University and his Bachelors of Science degree from Virginia Commonwealth University in Mathematics Education.

Mr. Hunt has been married to his wife Barbara for 50 years. They have two children and three grandchildren. He is an active member of Regester Chapel United Methodist Church. Mr. Hunt has been an active participant in local Public Service. He was elected to the Tri-County/City (TCC) District of the Virginia Soil and Water Conservation Board (SWCB). He served as Chairman of that Board in 2009 and Treasurer in 2010/2011. He was also elected to the Stafford County School Board from the Aquia District. He was appointed to the Stafford County Utilities Commission and Served as Chairman in 2008 and 2009. He serves on the Agricultural/Purchase of Development Rights Commission of Stafford County. Mr. Hunt has served as President and Secretary of the Stafford Education Foundation; a 501(c)3 IRS Charity dedicated to supporting public education in Stafford County.

Ian Brown

Ian Brown, the Director of Operations and Systems Analysis at Galorath, Inc., leads the company's IT and Software consulting practice. He has over 20 years of experience in software

measurement and analysis, systems estimation, software sizing, parametric analysis, and goal-question-metric (GQM) implementation. He has generated cost and schedule estimates and cost risk analyses on hundreds of software development and enhancement projects. A Certified Function Point Specialist from 2001 – 2009, Ian served on the Board of Directors of the International Function Point Users Group (IFPUG) from 2004-2007. Ian earned a bachelor's degree from Cornell University and a master's degree in public policy from Harvard University. He has published articles on the topic of software estimation and measurement and has been a featured speaker on the topic at numerous international conferences. Ian was also a contributing editor to Dan Galorath's 2005 book "Software Sizing, Estimation, and Risk Management: When Performance is Measured Performance Improves".

David DeWitt

Primary responsibilities include direct customer liaison and technical assistance for the SEER for Software and SEER for IT software products.

Certified Scrum Master and was instrumental in developing the SEER for Software Agile knowledge bases, Agile Estimation Webinars, Story Point Proxy development and training, and Agile Planner. He has delivered story point training to international customers and supported Agile capability deployments.

David was the principal architect of the Galorath Software *Estimation Maturity Assessment* product as well as the principal investigator for SEER enhancements to support Agile development and Software as a Service (SaaS). David has performed assessments in many technologically diverse commercial organizations such as Healthcare, Banking, Automotive, and Petroleum Refinement.

At Northrop Grumman, he developed the SEER for Software cost model for the SBIRS SPA re-baseline, wrote the cost control account plan for entry into Earned Value Management System (EVMS), developed build plan with SLOC estimates, drafted Risk Mitigation Plans, and coordinated all activities with the "watchdog" teams, Congressional Representatives, and employees. Served as the Cost Account Manager.

David served as the Integrated Product Team lead for the Spaced Based Microwave Imager Sounder (SSMIS) program. He managed the Flight, Early Orbit, and Ground software development teams. Participated in architecture/design changes, directed all programming activities, traveled to support program reviews and business development. Led inaugural launch support team, calibration/validation and Early Orbit support activities.

Numerous years as an embedded systems developer for space based and communications systems. Early Ada adopter and University Instructor (San Diego State University and Mesa College).

Esteban Sanchez

Born and raised in the tropical Costa Rica, Esteban spent the first 10 years of his career as software developer and project manager for mission critical embedded software and hardware applications. Some of the software he developed and managed is currently flying on the 787

Dreamliner aircraft from Boeing and some other aircrafts. He currently performs as senior Cost Estimation Consultant at Galorath Incorporated.

Esteban graduated from the Costa Rican Institute of Technology with bachelor's degree in Electronics Engineering in 2005, and later from the Costa Rican National University where he obtained his Honored Masters in Software and IT Project Management. Esteban is also a Certified SCRUM Master from the SCRUM Alliance, PMI Project Management Professional Certified as well as Certified Function Points Specialist (IFPUG CFPS), Certified SNAP Practitioner (IFPUG CSP) and active member of the Functional Sizing Standards Committee at IFPUG. His expertise includes development, project management, risk analysis, measurement and estimation for software and IT projects, Earned Value Management, Agile and Dev Ops. Throughout his career, Esteban has helped customers in the banking, retail, healthcare, mining, aerospace and defense industries to successfully estimate projects.

Esteban joined Galorath in 2011, where he currently performs as Senior Cost Estimation Consultant. At Galorath, in addition to helping customers around the world to be successful at estimating software, hardware and IT projects, Esteban has been the solution analyst of many internal research and development projects, including the calibration of the DO178C Knowledge Bases for SEER for Software, the Galorath Estimation Process Catalog and the development of the SEER Agile Planner. Esteban is also a frequent international speaker at the major Function Points and Cost Estimation venues (such as IFPUG and NESMA). He is a contributor to the ongoing development of the ICEAA Software Cost Estimation Body of Knowledge (SCEBoK). Esteban has several publications on software estimation including the paper "Use of Function Points in Embedded Systems – Stall Warning Protection Computer (SWPC) System" which is available from the International Function Points Users Group online store.

Joe Dean

Joe Dean is currently the Software Subject Matter Expert for Galorath Federal. He recently retired as the Chief for the Air Force Cost Analysis Agency (AFCAA) at Hanscom AFB in Massachusetts. He spent 19 years with Tecolote Research Inc., as a Senior Cost Analyst and Technical Expert. His focus is on cost estimating Software intensive systems and he applies his skills on everything from submarines to satellites. He is a former member of the Software Engineering Institute's Software Acquisition Metrics Working Group and past SCEA National Vice President. He is also retired from the Air Force where he spent the last seven of his 22 year career at ESC/FMC, Hanscom AFB as a Cost Analyst and Cost Research Analyst where he developed the "ESC Cost per Line of Code Model". Before that he was a Software Systems Analyst for the Communications Computer Programming Center at Tinker AFB in Oklahoma. Among many other things Joe has a Bachelor of Science degree in Math, a Masters Degree in Business Administration, is a Co-author of, Practical Software Measurement, Objective Information for Decision Makers and enjoys being a reenactor of the Revolutionary War era.