Applying Earned Value Management to Agile Software Development Programs

By Robert P Hunt, Michael Thompson, and Dan Galorath
Galorath Federal Incorporated

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.

Introduction

The Department of Defense estimates that software now accounts for 40% of all research, development, test and evaluation (RDT&E) spending[i]. 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 technical and earned value management (EVM) techniques can significantly improve the current situation.

Inaccurate estimates can threaten project success causing poor project implementations, the shortcutting critical processes, and emergency staffing to recover schedule. The lack of well-defined project requirement and specifications may result in significant growth in cost and schedule. Symptoms of this growth may include constantly changing project goals, frustration, customer dissatisfaction, cost overruns, missed schedules, and the failure of a project to meet its objectives.

A general movement to Agile software development process is one method used to address and potentially resolve these problems. While current Agile software metric indicate near term efficiency gains of up to 20%, the movement to Agile Software Development introduces several unique problems for EVM assessments.

PMI published an analysis of several government defense and intelligence agency large-scale acquisition programs that experienced significant cost and schedule growth. This analysis shows that several critical factors need to be addressed in the pre-acquisition phase of the acquisition cycle. The principal causes of growth on these large-scale programs can be traced to several causes related to overzealous advocacy, immature technology, lack of corporate technology roadmaps, requirements instability, ineffective acquisition strategy, unrealistic program baselines, inadequate systems engineering, and work-force issues.[ii] This paper will discuss some key element associated with:

1. Agile/Hybrid Agile software development

2. Establishing a process for developing the technical, cost and schedule baseline

3. Identifying critical software management metrics, and
4. Applying Earned Value Management to agile programs

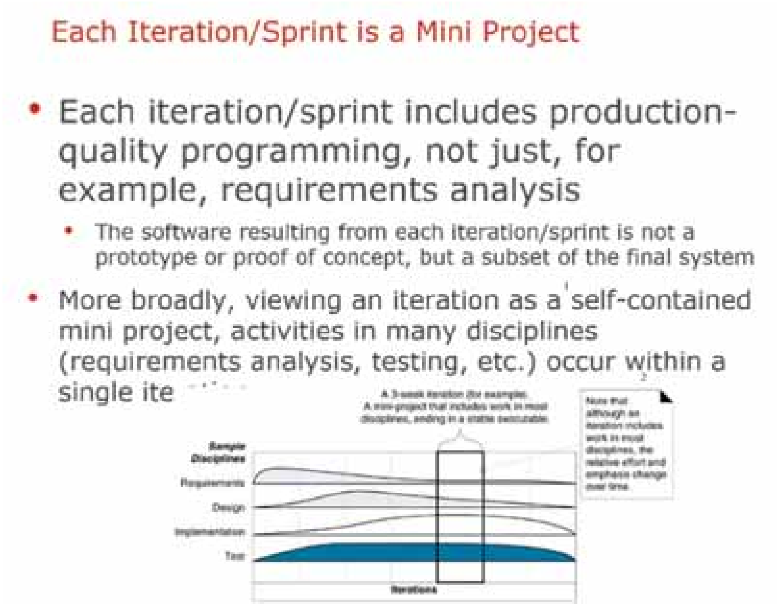
1.0 Agile/Hybrid Agile Software Development Programs

Software development methodologies have evolved since the early 1960‘s. 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. In an effort to reduce cost overruns and project failures, many DoD acquisition programs are moving to an Agile/Hybrid Agile development approach.

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." 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."

Figure 1, Each Iteration/Sprint is A Mini-Project Presents a common illustration of the Agile process where each sprint is a "mini-project.



Each Iteration/Sprint is a Mini Project

- Each iteration/sprint includes production-quality programming, not just, for example, requirements analysis
  - The software resulting from each iteration/sprint is not a prototype or proof of concept, but a subset of the final system
- More broadly, viewing an iteration as a self-contained mini project, activities in many disciplines (requirements analysis, testing, etc.) occur within a single ite

A 3-week iteration (for example).
A mini-project that includes work in most disciplines, ending in a stable executable.
Note that although an iteration includes work in most disciplines, the relative effort and emphasis change over time.
Sample Disciplines
Requirements
Design
Implementation
Test
Iterations

Figure 1 Each Iteration/Sprint is A Mini-Project

In practice many DoD programs are actually executing what might be called a Hybrid Agile acquisition strategy. Figure 2, Typical Hybrid Agile Development below presents a typical Hybrid Agile approach.   As this figure presents many up front activities, such requirements documentation is executed under a structured (waterfall) approach. While, only the actual coding and sometimes testing is done under the agile approach.

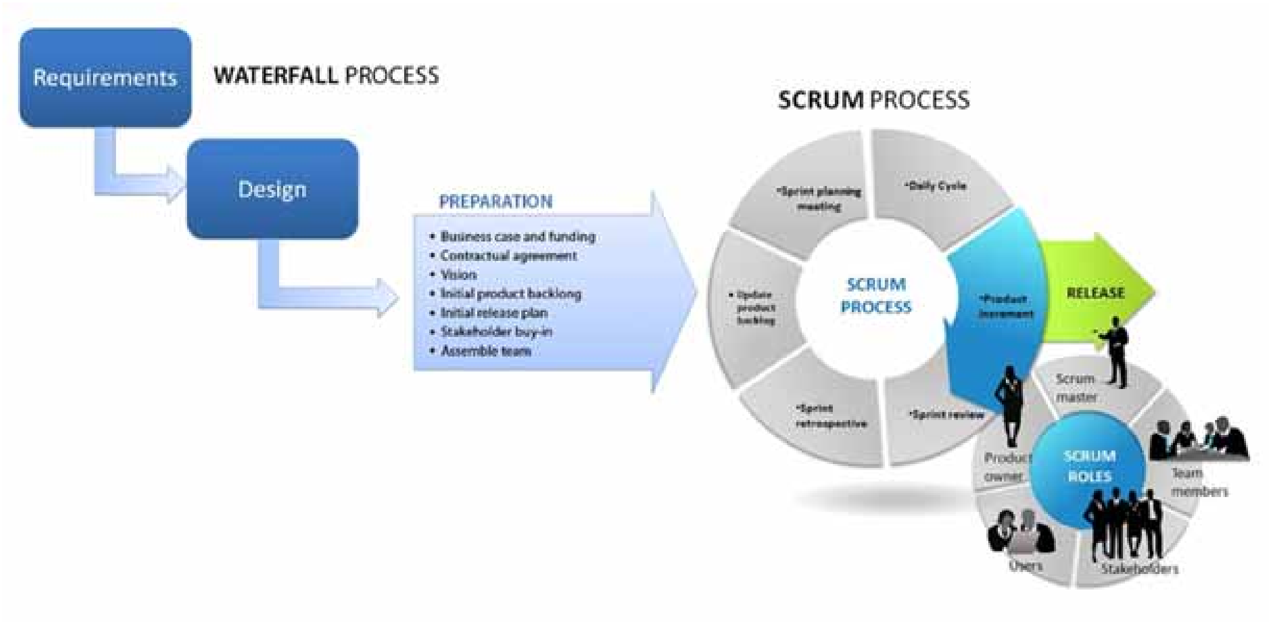


Figure 2, Typical Hybrid Agile Development

Within the agile software development context, the sprint is the fundamental work element. In the agile software development cycle, sprints roll up into features, features roll up into themes, and themes roll up into releases. *Some projects may substitute stories for features and epics for themes, but the general roll up process is the same.* This concept is presented in Figure3, Agile Building Blocks.

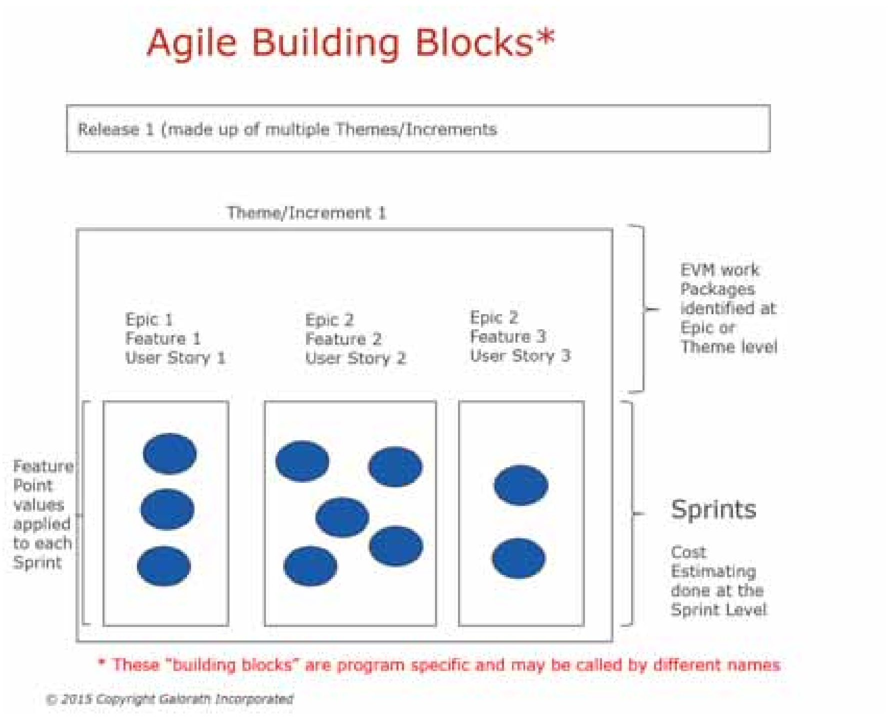
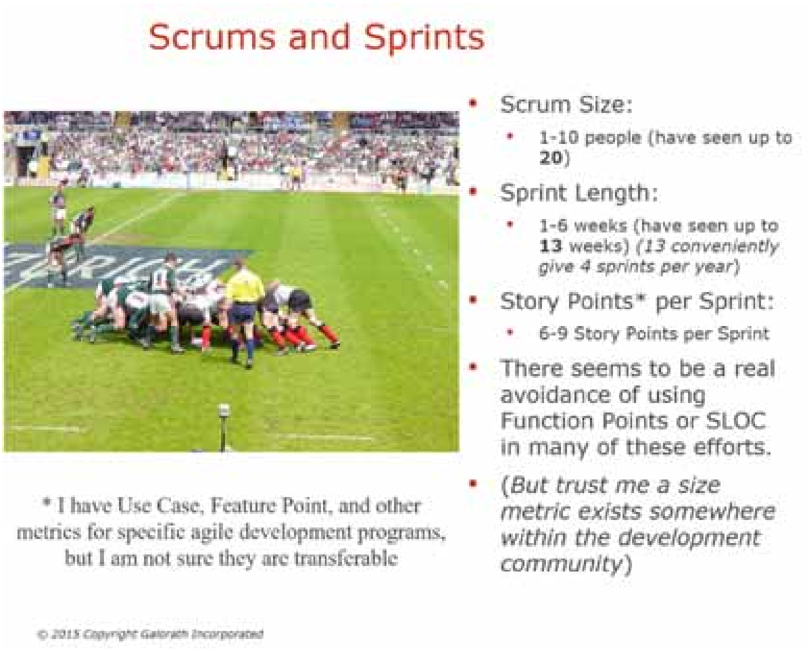

Agile Building Blocks*
Release 1 (made up of multiple Themes/Increments
Theme/Increment 1
Epic 1
Feature 1
User Story 1
Epic 2
Feature 2
User Story 2
Epic 2
Feature 3
User Story 3
EVM work Packages identified at Epic or Theme level
Feature Point values applied to each Sprint
Sprints
Cost Estimating done at the Sprint Level
* These "building blocks" are program specific and may be called by different names
© 2015 Copyright Galorath Incorporated

Figure3, Agile Building Blocks

A key point here us that all features are not the same. Some features take multiple sprints due to technical difficulty or complexity. Generally, a numeric rating schema (points) are assigned to each feature using small-medium-large, Fibonacci sequence, 1-10-100, 1-2-3-5-8-13-40-100, Planning Poker, or some other rating schema. For most Agile software development program there is some measure similar to Features and Feature Points. Where Features represent a basic "measurable" work package and Feature Points represent the technical difficulty of each work package. Often Feature Velocity (the rate at which features are completed) is an additional measure.

Figure 4, Sprints and Scrums, provides some data on scrums and sprints.



Scrums and Sprints

- Scrum Size:
  - 1-10 people (have seen up to **20**)
- Sprint Length:
  - 1-6 weeks (have seen up to **13** weeks) *(13 conveniently give 4 sprints per year)*
- Story Points* per Sprint:
  - 6-9 Story Points per Sprint
- There seems to be a real avoidance of using Function Points or SLOC in many of these efforts.
- (*But trust me a size metric exists somewhere within the development community*)

* I have Use Case, Feature Point, and other metrics for specific agile development programs, but I am not sure they are transferable

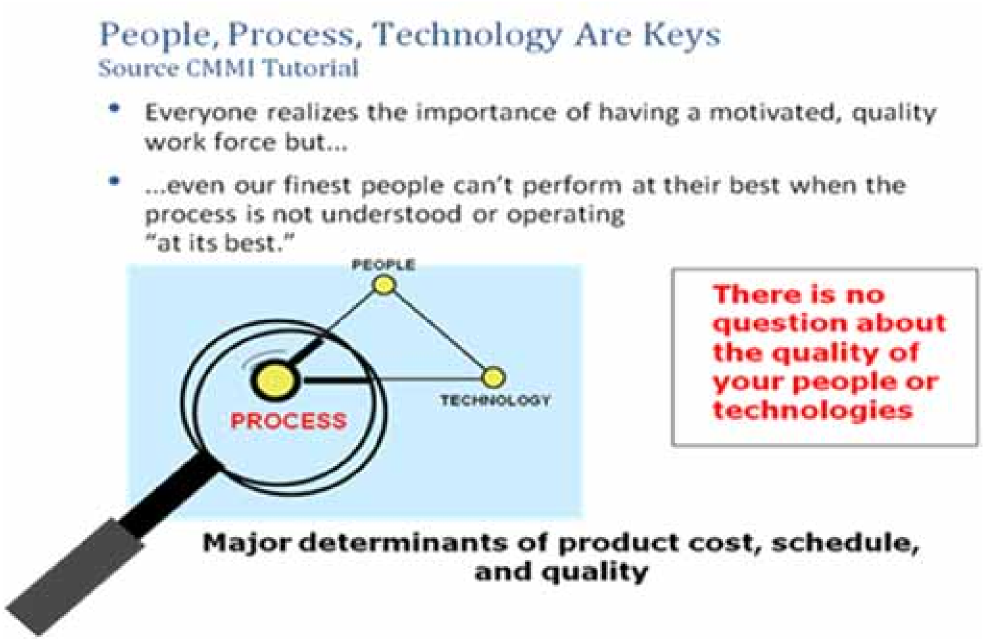
Figure 4, Sprints and Scrums

2.0 Establishing a Process for the Technical, Cost and Schedule Baselines

A software intensive program life cycle cost estimate is the most knowledgeable statement one can make at a particular point in time regarding effort/cost, schedule, staffing, risk, and reliability. However, the most important business decisions about a software intensive project are often made at the time of minimum knowledge and maximum uncertainty. Cost and schedule estimators recognize that the estimate is not a point, but rather a well formed estimate defined by a probability distribution, a joint cost and schedule risk distribution.

A well-defined process is critical to defining the requirements and completing the initial cost and schedule estimate. The proper use of EVM provides for integration of project technical scope, schedule, and cost objectives; and the establishment of a baseline plan for performance measurement. Additionally, the use of an analytic tool to project likely cost and schedule based on actual performance provides for realistic projections of future performance. Success of the project can be aided by defining the best objectives, by planning resources and costs which are directly related to those objectives, by measuring accomplishments objectively against the plan, by identifying performance trends and problems as early as possible, and by taking timely corrective actions.

A CMMI tutorial recognizes that people, process, and technology (sometimes called the iron triangle) are major determinants of product cost, schedule, and quality. We all realize the importance of having a motivated, quality work force but even our finest people can't perform at their best when the process is not understood or not operating at its best. Figure 5, People, Process, Technology are Keys, presents this concept.



People, Process, Technology Are Keys

Source CMMI Tutorial

- Everyone realizes the importance of having a motivated, quality work force but...
- ...even our finest people can't perform at their best when the process is not understood or operating "at its best."

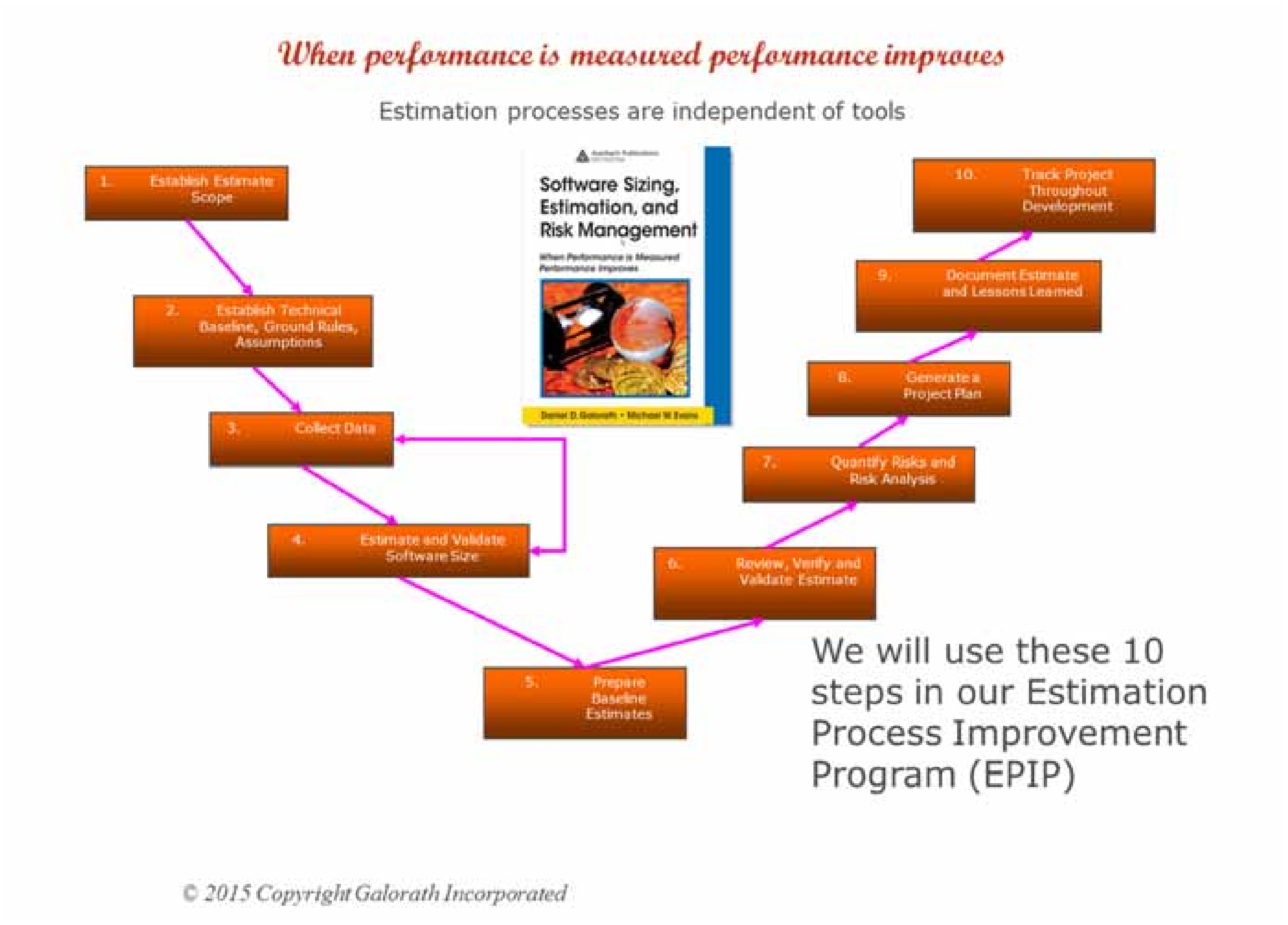
PEOPLE
PROCESS
TECHNOLOGY

There is no question about the quality of your people or technologies

Major determinants of product cost, schedule, and quality

Figure 5, People, Process, Technology are Keys

In the book, "Software Sizing, Estimation and Risk Management" (Dan Galorath and Michael Evans, 2007) a ten step process is presented for program requirements generations and estimation. Figure 6, 10 Step Software Process, outlines the ten steps.


When performance is measured performance improves

Estimation processes are independent of tools

1. Establish Estimate Scope
2. Establish Technical Baseline, Ground Rules, Assumptions
3. Collect Data
4. Estimate and Validate Software Size
5. Prepare Baseline Estimates
6. Review, Verify and Validate Estimate
7. Quantify Risks and Risk Analysis
8. Generate a Project Plan
9. Document Estimate and Lessons Learned
10. Track Project Throughout Development
Software Sizing, Estimation, and Risk Management
When Performance is Measured Performance Improves
Daniel D. Galorath • Michael W. Evans

We will use these 10 steps in our Estimation Process Improvement Program (EPIP)

Figure 6, 10 Step Software Process

While the Galorath process includes ten steps, other process may include more or less steps (e.g. the GAO Cost Guide includes a 12 steps process). Note specifically the importance of step 4, *estimating and validating the software size metric*. The key here is to establish an auditable, repeatable set of steps to establish the requirements and develop the baseline estimate of cost and schedule. This is the key to articulating an accurate requirement and establishing a reliable baseline for cost and schedule.

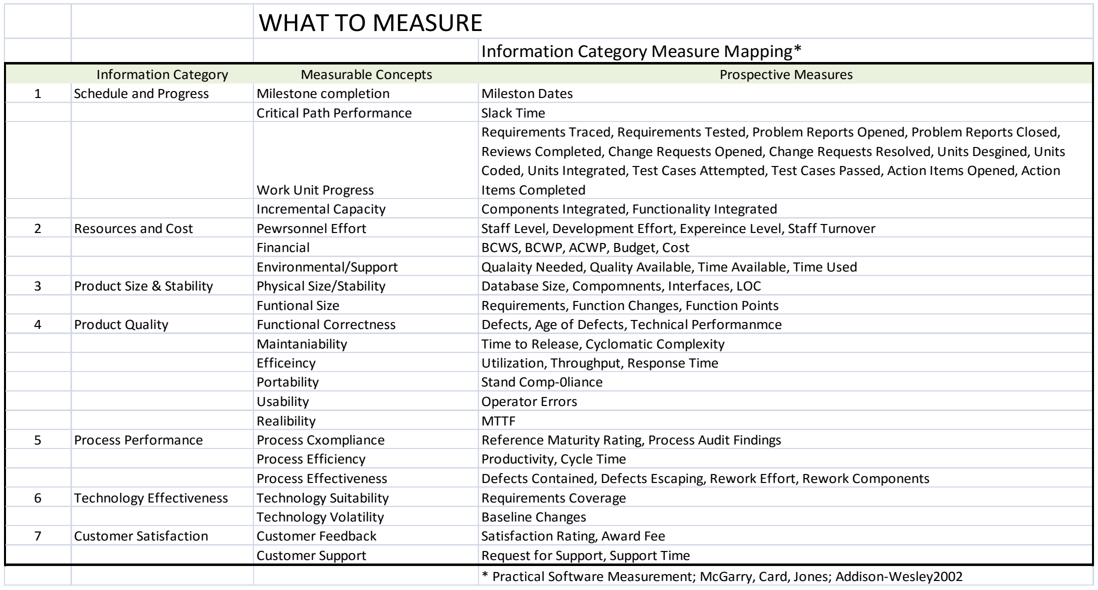
The two primary keys to establishing an Agile Baseline are to make sure that the sprints are located at an appropriate level of the WBS in order that when some sprint tasks go into “backlog” they don’t affect the baseline of the program and that the sprints average 10 to 20 days (working) in duration. The preferred Earned Value technique, for the sprints becomes a Weighted Milestone Method, enabling each sprint to become an apportioned part of the Epic.

Preparing for an IBR is not dissimilar using an Agile methodology, what should be kept in mind are form of the lessons learned, from past Agile programs. A disciplined approach to Release Planning is imperative, the Agile methodology is built around a disciplined fast moving approach, in order to be an effective tool. Sprint Planning, Execution, and Measuring Technical Performance is critical, even when using a Hybrid approach to Agile, it is necessary to rely on accurate planning and execution and provide

Technical Performance criteria that enable accurate performance measurement techniques. In an IBR schedule is a prime driver, when using the Agile methodology, you need to assign the same importance to Risk, that you do to Critical Path Methodology. Being able to identify and control program risk will feed into accurately being able to use the Integrated Master Schedule to control the program.

3.0 Identifying Critical Software Management Metrics

That most large software intensive programs get into trouble is a demonstrated phenomenon. Therefore, selecting the correct set of software metrics to track is critical to program success. Practical Software Measurement (McGarry, Card, Jones; Addison-Wesley, 2002) identifies seven information categories and then expands these information categories into measurable concepts and then prospective metrics[iii]. This taxonomy is presented in the Figure 7, What To Measure.



WHAT TO MEASURE

Information Category Measure Mapping*

	Information Category	Measurable Concepts	Prospective Measures
1	Schedule and Progress	Milestone completion	Mileston Dates
		Critical Path Performance	Slack Time
		Work Unit Progress	Requirements Traced, Requirements Tested, Problem Reports Opened, Problem Reports Closed, Reviews Completed, Change Requests Opened, Change Requests Resolved, Units Desgined, Units Coded, Units Integrated, Test Cases Attempted, Test Cases Passed, Action Items Opened, Action Items Completed
		Incremental Capacity	Components Integrated, Functionality Integrated
2	Resources and Cost	Pewrsonnel Effort	Staff Level, Development Effort, Expereince Level, Staff Turnover
		Financial	BCWS, BCWP, ACWP, Budget, Cost
		Environmental/Support	Qualaity Needed, Quality Available, Time Available, Time Used
3	Product Size & Stability	Physical Size/Stability	Database Size, Compomnents, Interfaces, LOC
		Funtional Size	Requirements, Function Changes, Function Points
4	Product Quality	Functional Correctness	Defects, Age of Defects, Technical Performanmce
		Maintaniability	Time to Release, Cyclomatic Complexity
		Efficeincy	Utilization, Throughput, Response Time
		Portability	Stand Comp-0liance
		Usability	Operator Errors
		Realibility	MTTF
5	Process Performance	Process Cxompliance	Reference Maturity Rating, Process Audit Findings
		Process Efficiency	Productivity, Cycle Time
		Process Effectiveness	Defects Contained, Defects Escaping, Rework Effort, Rework Components
6	Technology Effectiveness	Technology Suitability	Requirements Coverage
		Technology Volatility	Baseline Changes
7	Customer Satisfaction	Customer Feedback	Satisfaction Rating, Award Fee
		Customer Support	Request for Support, Support Time

* Practical Software Measurement; McGarry, Card, Jones; Addison-Wesley2002

Figure 7, What To Measure

Collecting and tracking data on all the prospective metrics is impractical for typical software intensive programs. Software developers often produce their software deliverables in unique environments and with unique processes. In selecting the appropriate software metrics, the analyst must "do your own thing, but carefully." [iv]

For Earned Value purposes, the most effective software metrics are those that relate to product size, schedule, quality, and progress. For software intensive programs, measures of quantity (e.g. number of lines of code completed) may not accurately reflect the quality aspects of the work performed on neither the program nor the actual progress since items such as lines of code completed do not capture items such as integration, testing, etc.

Size is often measured as Source Lines of Code (SLOC) or Function Points and used as a sizing measure for budgets and for earned value using a percent of completion method. It should be noted that most agile development programs avoid any reference to SLOC or Function Points. There are critical problems with this approach. First, there has traditionally been a significant error in estimating SLOC. And, the number of lines of code completed does not necessarily reflect the quality or total progress toward a performance goal. Therefore, any progress metric based solely on SLOC is highly volatile. Whether SLOC, function points, Use Cases, Features or some other size artifact is selected, a careful process must be utilized to establish a credible size metric. It is recommended that in addition to tracking progress toward a goal, size growth should also be tracked.

Schedule metrics and procedures normally relate to completion milestones are also a common tracking metric. Sometimes these milestone definitions and completion criteria lack quantifiable objectives. Often an incremental build is released that does not incorporate all the planned functional requirements or a developer claims victory after just testing the nominal cases.

Progress metrics can be very difficult for large software programs. It is generally agreed that no software is delivered defect free. Software engineers have hoped that new languages and new processes would greatly reduce the number of delivered defects. However, this has not been the case. Software is still delivered with a significant number of defects. Capers Jones estimates that there are about 5 bugs per Function Point created during Development[v]. Watts Humphrey found "… even experienced software engineers normally inject 100 or more defects per KSLOC[vi]. Capers Jones says, "A series of studies the defect density of software ranges from 49.5 to 94.5 errors per thousand lines of code[vii]." The physical and practical limitations of software testing (the only way to determine if a program will work is to write the code and run it) ensure that large programs will be released with undetected errors. Therefore, defects discovery and removal is a key metric for assessing program quality for SLOC and/or Function Point sized systems. Unfortunately, at this time no such metric exists for Agile development programs. Feature backlog is one useful metric, but no macro database exists at this time to allow for cross project comparisons.

The analyst should review the list of potential measures defined in Figure 7, What To Measure, and select the set of metrics that are most appropriate for a specific program.

The critical issue in an Agile software development environment is that SLOC, Function Points, and other normal metrics are not collected. Features, Feature Points, and Velocity (or their counterparts) are often the best metrics. The figures below present how Features, Feature points, and Velocity might be tracked in a Hybrid Agile environment.

Figure 8, Feature Delivery, demonstrates how Feature completion can be plan and tracked. This may be a good measure of quantity in an Agile environment.

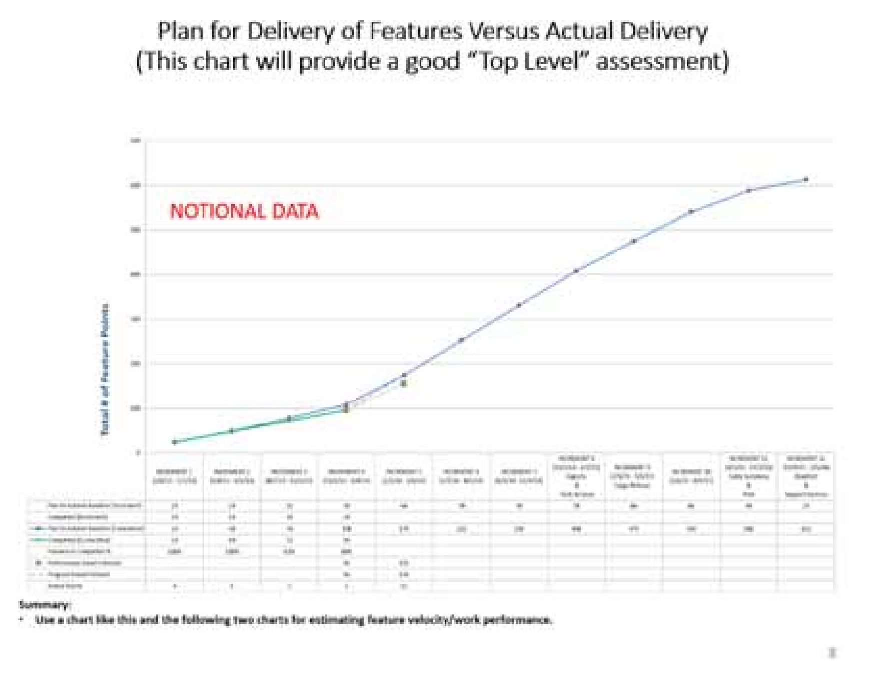
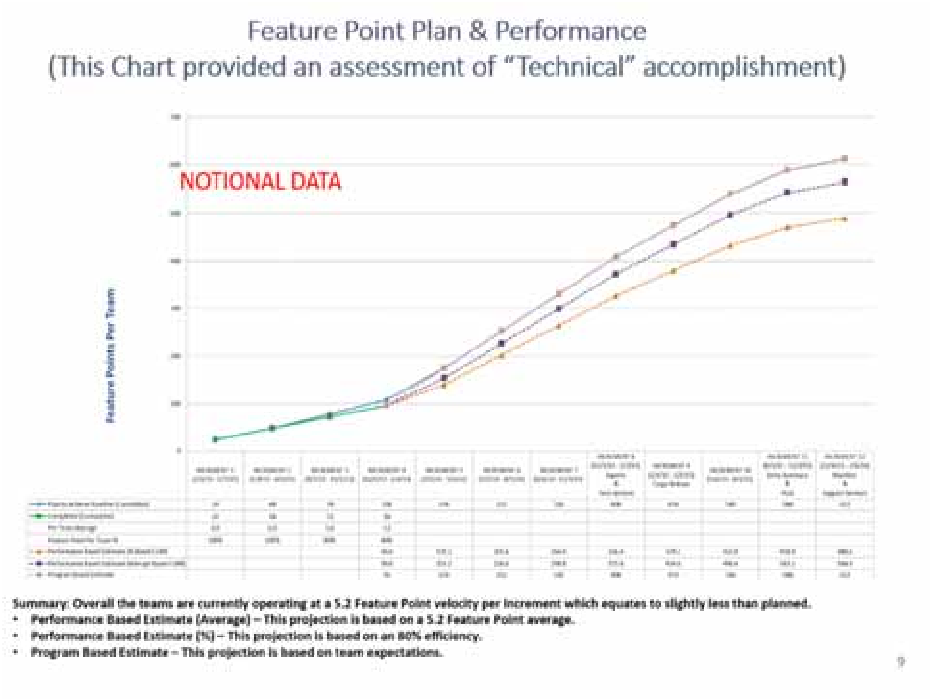


Figure 8, Feature Delivery

Figure 9, Feature Point Delivery, shows how feature points could be planned and measures. Feature points, provide a measure of both quantity and technical progress.


Feature Point Plan & Performance
(This Chart provided an assessment of "Technical" accomplishment)

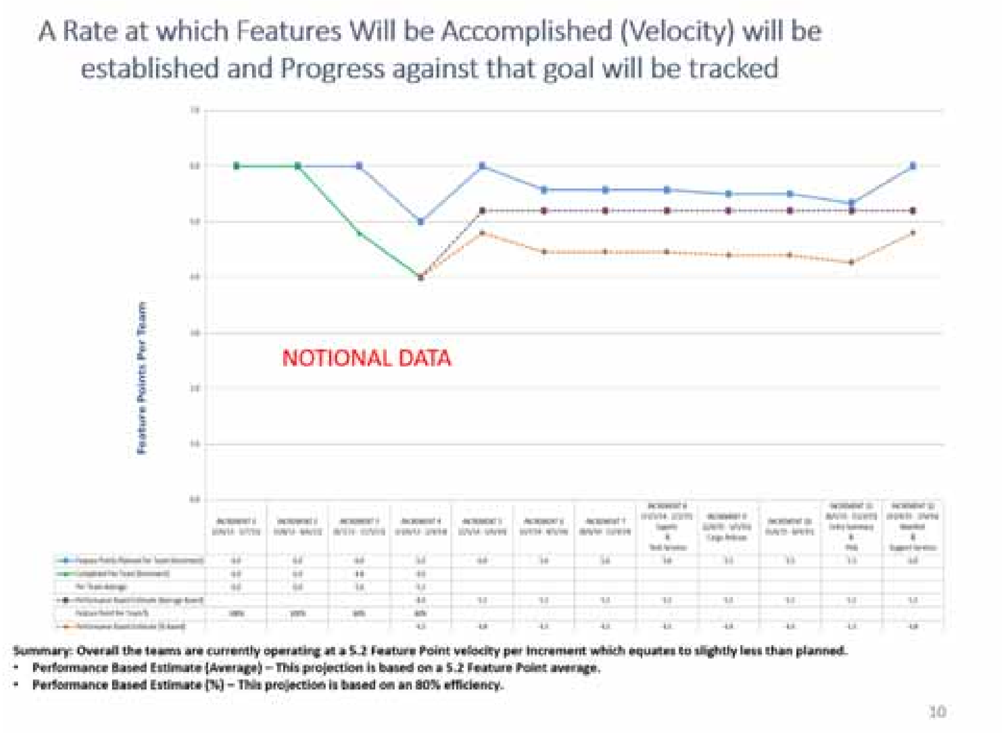
Summary: Overall the teams are currently operating at a 5.2 Feature Point velocity per increment which equates to slightly less than planned.

- Performance Based Estimate (Average) – This projection is based on a 5.2 Feature Point average.
- Performance Based Estimate (%) – This projection is based on an 80% efficiency.
- Program Based Estimate – This projection is based on team expectations.

9

Figure 9, Feature Point Delivery

Figure 10, Feature Velocity, measures the rate at which features are completed. This measure provides a picture of whether a project is ahead or behind schedule.


A Rate at which Features Will be Accomplished (Velocity) will be established and Progress against that goal will be tracked

Summary: Overall the teams are currently operating at a 5.2 Feature Point velocity per increment which equates to slightly less than planned.

- Performance Based Estimate (Average) – This projection is based on a 5.2 Feature Point average.
- Performance Based Estimate (%) – This projection is based on an 80% efficiency.

10

Figure 10, Feature Velocity

4.0 Applying Earned Value Management to Agile Programs

According to the Defense Acquisition University (DAU) website**,** Earned value is a management technique that relates resource planning to schedules and to technical cost and schedule requirements. All work is planned, budgeted, and scheduled in time-phased "planned value" increments constituting a cost and schedule measurement baseline. This cost and schedule baseline is documented in the Integrated Baseline Review (IBR). There are two major objectives of an earned value system:

- To encourage the developer to use effective internal cost and schedule management control systems; and
- To permit the customer to be able to rely on timely data produced by those systems for determining product-oriented contract status.

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, might be a good measure of quality. A standard “quality” measure for agile software development has not been established.

Program managers expect accurate reporting of integrated cost, schedule, and technical performance when the systems integrator’s EVMS procedure complies with the EVMS Standard. However, EVM data will be reliable and accurate only if the following occurs:

- The indicated quality of the evolving product is measured.
- The right base measures of technical performance are selected.
- Progress is objectively assessed.

Using EVM also incurs significant costs. However, if you are measuring the wrong things or not measuring the right way, then EVM may be costlier to administer and may provide less management value.

A well-defined software intensive program EVM process has many distinguishing characteristics:

1. The plan is driven by product quality requirements.
2. The focus is on technical maturity and quality, in addition to work units delivered.
3. The focus is on progress toward meeting success criteria of technical reviews.
4. The program adheres to standards and models for systems engineering, software engineering, and project management.
5. The plan is based on smart work package planning.
6. The plan uses tangible results to measure (function points, completed Features or Epics)
7. The plan enables insightful variance analysis.
8. The plan ensures a lean and cost-effective approach.

9. The plan enables scalable scope and complexity depending on risk.
10. The plan integrates risk management activities with the performance measurement baseline.
11. The plan integrates risk management outcomes with the Estimate at Completion.

Conclusion

Using earned value to plan and manage software intensive projects can prevent expensive failures. Earned value should be based on the foundation of establishing the requirements, developing a reliable baseline estimate for cost and schedule, selecting effective software metrics, applying Earned Value Management, and using analytic processes to project cost and schedule based on actual performance.

Authors Biographies

Bob Hunt has over 40 years of cost estimating and analysis experience. He received his SCEA Certification in 1991. He has served in senior technical and management positions at 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 IEEE and DCAS. 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. Mr. Hunt was the principal author of the initial U. S. Army Cost Analysis Manual. In addition to his ICEAA Treasurer responsibilities, Mr. Hunt has held leadership positions and made technical presentations for the American Institute of Aeronautic and Astronautics (AIAA), the National Association of Environmental Professionals (NAEP), and the IEEE.

Mr. Hunt received his Masters Degree from Virginia State University and his Bachelors of Science degree from Virginia Commonwealth University in Mathematics Education. Mr. He also served as the Chairman of the Economics Technical Subcommittee of the AIAA.

Michael Thompson is a internationally recognized as a leader, teacher, and consultant on Earned Value Management (EVM). He presented several papers on EVM, Price/Cost Analysis, and Scheduling. He has taught hundreds of professionals and led EVMS implementation, compliance reviews, Integrated Baseline Reviews, independent assessment reviews, and process improvement teams. He is certified as an EVM Professional (EVP). Mr. Thompson has been a member of the Cost Community for more than 30 years supporting DoD agencies, FAA, BLM, EPA, and Army Corps of Engineers.

[i] Page 134, Trillions For Military Technology; John A.Alic, Palgrave MacMillian, 2007

[ii] PMI Project Management Journal, March 2008; "Best Project Management and Systems Engineering Practices in the Preacquisition Phase for Federal Intelligence and Defense Agencies" by Steven R. Meier
[iii] Page 37, Practical Software Measurement; McGarry, Card, Jones; Addison-Wesley 2002
[iv] P. 68, A Practical Guide to Earned Value Project Management, Charles Budd and Charlene Budd; Management Concepts, 2005
[v] Geriatric Issues of Aging Software, Capers Jones, CrossTalk, Dec 2007, Vol. 20 No 12
[vi] Humphrey, W.,"A Personal Commitment to Software Quality." Pittsburg, PA: The Software Engineering Institute (SEI)
[vii] Jones, T.C. Programming Productivity. New York: McGraw-Hill, 1972