Earned Value Management Software Projects

Robert A. Hanna, PMP, *Member, IEEE*

***Abstract*—JPL has been attempting to apply EVM techniques for the last 15 years with mixed success, especially when applied to software tasks. One of the main causes is that software presents particular challenges that were not considered during the genesis of EVM. A small number of tasks have been able to use EVM effectively and derive great benefits. This paper discusses challenges specific to software management and how they can be successfully addressed. Specifically, it details lessons learned from the $28 million dollar Next Generation Navigation Software Project that replaced NASA's deep space navigation software for operations. The project was able to successfully use EVM techniques during a four year period, producing 12 releases on their promised delivery dates and projects actual costs that were within 0.3% of the planned budget (under), in contrast to performance before the techniques were put into practice.**

***Index Terms*—Cost, Schedule, Software, Management**

I. Introduction

EARNED value management is a technique for providing an objective analysis of the progress of a project in terms of program dollars. The technique integrates schedule and budget performance to provide metrics that enable good decision-making. Software projects exhibit a high level of volatility, with a large project failure rate and must be addressed accordingly to gain the benefit of earned value management.

During the 1990s the first uses of earned value management in software focused on large projects implementing the waterfall methodology. For most cases during that time when EVM was used there was an emphasis on a large amount of initial planning. [1][2]

From 2000 to 2005, there was literature discussing the application of EVM to iterative techniques. In these instances the suggestion was that all requirements should be testable and unit test coverage should be at 100%. [3]

In the current period, more approaches to bringing the benefits of EVM have been described. However, the two dominant approaches in the literature at this time appear to be the Performance Based EVM and the AgileEVM approaches.

This techniques discussed here are not in conflict with these approaches, it is more an attempt to address some specific concerns and provide practical solutions to address them. Advocates of these existing systems may find these useful.

Those practicing AgileEVM may find some value here. Since Agile EVM does not have any consideration for variance analysis, combined with the non-homogeneity of scrum points as they are defined, high volatility tasks such as prototyping and architectural changes may be a challenge.

Practitioners of the PBEVM method may also find some value. The PBEVM approach certainly has an examination of variances from planned values and a traditional approach to risk management. However, the wide variety of potential metrics that could possibly be used to measure the quality of a project may seem overwhelming, and a specific discussion of a specific EV method for measuring task progress may be of interest.

Manuscript received May 1, 2009.

R. A. Hanna is with the Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91125 USA (phone: 818-354-1183; e-mail: Robert.A.Hanna@jpl.nasa.gov).

II. The Challenges

Software projects are unique in several areas that make earned value management a challenge. The following are some of the most common challenges when trying to apply earned value to a software project.

A. Innovation and Prototypes

Many software projects may be exploring functionality that has never been developed before or has never been applied to a particular functional area. This is a common challenge for software projects.

The Nature of the Challenge

When there are tasks associated with exploring new structures and functionality, there are often problems with estimation of those tasks with respect to cost and schedule. Their estimates may be precise, but they are not accurate because there is a large uncertainty associated with the estimate. The designs necessary to implement the new functionality are often unknown early in the project and depending upon how the project plan has been formed may remain unknown until just before project deliverables are expected. Sometimes an innovation that will be delivered is build upon many foundational pieces of software; as a result prototypes may be implemented late in the project.

When a project decides to build prototypes in order to try out a piece of functionality or an architectural construct or to decide between opposing architectural directions a similar problem occurs. There is a large dependence upon the skill and experience of the team and the number and complexity of these prototypes will vary accordingly. On well-formed and experienced teams the variance of the estimates for these prototypes is smaller than that or less experienced teams. In addition the prediction of how many prototypes will be needed is more accurate.

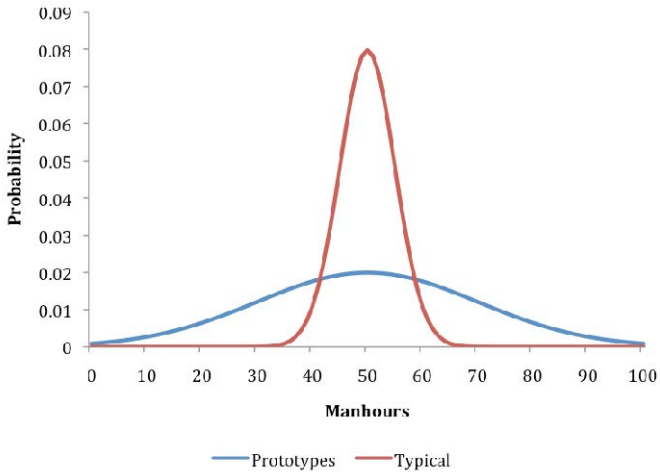

Fig. 1. An illustration of how prototype tasks have a larger variance than typical tasks. Both tasks shown here have a nominal value of 50 hours, but different characteristics when examined away from the nominal value.

When there is a large amount of uncertainty, projects may experience a two-fold issue with respect to schedule and budget. The first is how many prototype iterations are needed and the second is how much effort each prototype will take.

The Affect on Earned Value Management

As mentioned, prototypes and innovations are not always successful during the execution of a software project. When prototypes fail, or innovations fail to meet the expectations of project customers, the result is often unplanned rework. If the BCWS does not have a way to account for or handle such an eventuality, it can result in negative values for the performance indices such as SPI and CPI. When those values go negative, the calculations for the budget at completion (BAC) of the project as well as estimates for completion dates become unstable, resulting in non-sense answers in many EVM systems.

If the project manager decides to take the unplanned rework and add it to the project as new scope, resulting in a re-planning effort, all of the value of previously collected metrics may be lost. Again, this results in an inaccurate view of the project when examining the output of the EVM process, especially with respect to budget at completion (BAC.) [4]

B. *Defect Discovery and Resolution*

All software has defects. In addition, during the process of development of future releases the discovered defects of a previous release must be resolved or fixed. This process also presents a challenge to using earned value management on software projects.

The Nature of the Challenge

Defects can be discovered in different areas of the development process, and the effect on the resolution of those defects changes with each stage. While many projects and programs may delineate the phases of development in different ways, there are basically three phases where defects can be discovered. Those are operations defects, system test defects and development defects.

Operations defects, discovered by the end user of the software after the software has been delivered are the most expensive and carry the largest risk. They also are the most disruptive to project schedules and progress. In some cases, all work may halt while a high criticality defect, discovered in operational use, is quickly resolved. These situations may not be avoidable, but they can be managed.

System test defects, discovered in the process of software validation, are less expensive than operations defects. The software version may not have yet been delivered to operations, so the amount of work to fix the defect may be lower than operations defects.

Development defects, discovered by the development staff during the process of developing new functionality have the lowest impact. The developer discovers and resolves the defect, perhaps with the help of another member of the development staff.

When a development project does not have a stable development process, the number and severity of the defects discovered in the various phases can have large fluctuations. Even with a stable development process the number and complexity of defects discovered will have a fair amount of variation.

The Affect on Earned Value Management

The nature and size of the defects discovered can be hard to predict, resulting in a large amount of volatility and long delays with no positive progress (BCWP). This then results in performance indices that trend downward and without reaching equilibrium (CPI and SPI). This reduces the ability of a software project to accurately forecast (EAC, ETC).

Estimating the number of defects that are created per one thousand lines of code (the defect density) can be problematic. Process improvements, new processes, tool changes and project size may affect the defect density from one project to another. New team formation, communication and team cohesion can also affect the defect density for a specific software project. [5]

Even if there is an estimate of the defect density for new code development, there may not be an accurate representation or prediction of where in the process that the defects will be discovered. The amount of testing resources, the development process used as well as the availability of workarounds in the operational environment all affect where in the process the defects are discovered and resolved, and consequently affect project progress.

C. *Architectural Changes*

When the development of a software effort begins, certain architectural choices are made. As time progresses, and more engineering knowledge is gained of the system and the functionality it will deliver, some of the architectural choices made may come into question and need to be changed.

The Nature of the Challenge

As software projects progress, more knowledge is gained concerning the implementation of functionality and high-risk features. The architectural structures needed to implement the functionality may not be apparent early in the project, so fundamental architectural changes may be needed, resulting in large amounts of unplanned rework.

When a fundamental change to the software architecture is made, the affect of this change may be manifested in multiple software modules and components. Depending upon how atomic the different components and modules of the software system are each component and module may have a different amount of rework to perform to accommodate the architectural change.

In addition, the amount of change needed in each of the different modules and components may be difficult to predict. Even when predictions are accurate the amount of variance associated with the estimate may be large.

The Affect on Earned Value Management

As with challenges in other areas, an architectural change can result in unplanned rework. This unplanned rework can result in no project progress (BCWP) or even negative project progress for an extended period of time. Negative project progress is not allowed by most EVM software packages and when confronted with this possibility project managers will often want to re-plan the entire project, losing accumulated data that would be useful in estimations of project completion (EAC and ETC).

To compound the problem further, the architectural changes often result in a large number new defects being introduced into the software. So the challenges associated with defect discovery and resolution are also at play in this challenge. Namely, long delays where resources cannot be allocated to make project progress resulting in a constant BCWP and downward trending SPI and CPI indices.

III. The Solutions

There are several approaches that can make a software project more suitable for an earned value management approach.

A. Measuring Volatility through Metrics

Earned value management approaches suffer from a limitation when they are applied to an environment that has a high degree of volatility that can affect the metrics used for the measurement of the key indices of BCWP, CPI and SPI. In order to ensure that the values of the metrics collected and communicated accurately reflect the project and it's progress, the volatility of the project and the measures taken must be examined.

For example, if project progress is measured according to expenditures and schedule progress, but during the execution of the project the amount of testing is not stabilized, then, when testing is increased it will give an indication of a higher defect discover rate. This rate will not be a measure of an issue that has arisen in the development process, but rather a reflection of the amount of testing performed.

Because the measures of earned value management are cumulative in nature, it will take several measurement periods in order to stabilize the measurements and improve their accuracy. This is possible when the underlying project being described has reached an equilibrium state and is stable. If the system is still approaching equilibrium, the measures, which have no sense for measuring a dynamically shifting system, will fail to indicate the true progress.

Examining not only the traditional earned value management metrics, but also providing additional measures to gauge the volatility or stability of the project processes is a technique that is addressed by the PBEVM approach.

B. Task Definitions

On software projects a key to being able to accurately track work progress (BCWP) with earned value is a set of guidelines used for defining task duration. On projects that are small or large, if the duration of an individual task is too long, the tracking of that task becomes highly inaccurate (especially using the percent complete method.) If task durations are too short, the tracking becomes onerous and does not reflect the work performed. Guidelines for task duration discussed in this paper must be put into practices for successful use of EVM on software projects.

PERT Analysis

Using PERT analysis to discover not only the nominal value of the probability distribution for the number of hours to complete a particular task, but also to discover the variance or uncertainty of the final estimate is very important. PERT analysis accomplishes this by gathering the best case, worst case and most likely case duration estimates for each task. In each of the challenge areas we discussed earlier, a key aspect was a large amount of uncertainty in the estimates. If the uncertainty is not collected for planning, there can be little chance to prepare for, let alone react to, results that are far from the nominal value.

A simple calculation of the mean value of the distribution of an estimate of a task's duration uses the pessimistic estimate (P), the most likely estimate (M) and the optimistic estimate (O) and is as follows:

$$Mean = \frac{(P + 4M + O)}{6}$$

The one-sigma values can then be calculated using:

$$Sigma = \frac{(P - O)}{6}$$

From these values, any confidence interval can be calculated for schedule duration inputs.

Confidence Intervals

Planning for tasks should never be done using the nominal values. If the nominal values are used, that means that there is a 50% chance that the task will not complete on time. That leaves no room for the regular variations on project progress and for tasks that have a large uncertainty the penalty incurred by the project may be very large.

A common confidence interval for software projects is on the order if 80%. As with any project, the confidence interval that is chosen for the planning of the project should be negotiated with the project sponsor and match their tolerance for risk.

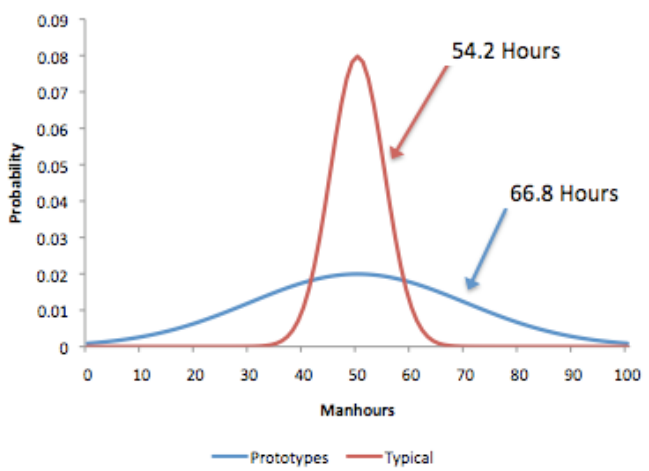

Fig. 2. An identification of the 80% confidence interval for the two example tasks described in Fig. 1. Notice the substantial difference in the value used for input into schedule duration for the two tasks.

Short Duration Tasks

There is a temptation to make tasks that are general in nature with long durations. Since there is so much uncertainty in each of the activities the belief is that the estimates of the summary tasks may be more accurate. In addition, software project managers often place a lot of hidden schedule reserve in the estimation of summary tasks, however, that reserve is usually based upon analogous experience and not on the particular uncertainty associated with the tasks at hand.

By keeping the duration of the tasks that are scheduled to a short duration, no more than two to four weeks, it ensures that schedule estimation errors are minimized and localized. If the task durations have been scheduled using an 80% confidence interval, then it eliminates the need for hiding schedule contingency in the summary task duration and ensures that the uncertainty of the specific effort is accounted for. This leads to greater schedule stability and enables EVM processes.

C. *Scheduling Techniques*

Scheduling tasks for an earned value management software task has special considerations. The fluidity with which tasks are started and stopped on a software project can become very frustrating for a schedule analyst who has been trained in the tradition techniques of earned value management. The desire for hard linking all low level tasks in each work package can work against accurate measurement.

Resource Loading

This should go without saying, but every task must be assigned to a resource in order to ensure that the task list can be measured against the available resources. With most software projects the majority of the expenditures are labor costs associated with development. If the tasks are not resource loaded there can be little assurance that the estimated completion date is reasonable.

In addition, there can also be specific software tasks that are only appropriate for certain development team members. If there is not a reasonable identification of the limited technical resources on the project going forward, resource bottlenecks may appear and slow the project. These considerations may be overlooked in agile methods for estimating project completion dates.

Weighted Milestones

When it comes to showing schedule progress on individual tasks, technical managers often opt for the precision of the percent complete method. Representing a percentage of completion of each task within one percent at each schedule update. While precise, this is usually not accurate. Tasks quickly reach the 90% complete mark and then stall while issues and rework are addressed, affecting the quality of schedule especially with respect to EVM. [6]

A technique of weighted milestones addresses this problem by associating specific percentages with specific events. One effective set of weighted milestones is to grant 25% for starting a task, 50% once significant progress has been made that demonstrates an understanding of the challenges involved in the task, 75% on the first report of completion, and then 100% upon confirmation of completion of the task. The 50-50 technique also works well when the duration of tasks is kept small. That technique applied 50% completion for starting the task and then 100% for completing it.

Besides being more accurate, while less precise in the measure of progress, these weighted milestone techniques free the project manager from having to spend time generating an artificial level of precision in measuring project progress.

Task Queuing Technique

When linking tasks to estimate project duration, the tradition view is that all tasks must have specific predecessors and successors at the lowest level in order for EVM to work effectively. This however does not reflect how the work is often performed on a software project. An alternative technique of task queuing is much more effective and reflects more accurately how the work is actually performed. It also can be formed in order to enable earned value management.

Task queuing is the technique of creating a set of collectively exhaustive tasks that are mutually exclusive based upon the specific resource that has been assigned to them. This means that resource leveling can be performed effectively, but the critical path for the project will pass through a summary task. For some commercial EVM software systems, this can be

difficult to handle so a summary task schedule may be needed for input into those systems.

For example, consider a project that is preparing for their next release. This project has all of the work in this next release in two different components. Each component represents a work package in the cost account that is being used to collect costs for the upcoming release. There are several different tasks associated with each component. In addition there are a total of five development staff members who can perform the work. Three have the expertise to perform work on module A and three can work on module B. Each has different approaches and may choose a different task to perform first. In the schedule for these activities, the tasks are spread across the resources that are available to do them and then the schedule is resource leveled at the summary task level for each module. This gives an indication of the forecasted completion date and cost for the work package associated with each component. If the EVM system cannot support this approach to task scheduling without defining predecessors and successors within the work package, then a summary task schedule report is created for input into the EVM software. As work progresses, a weighted milestone technique is used for each of the low level tasks, and a percent complete is calculated from those inputs for the summary task representing the work package. This percent complete is then used as the input for the EVM software.

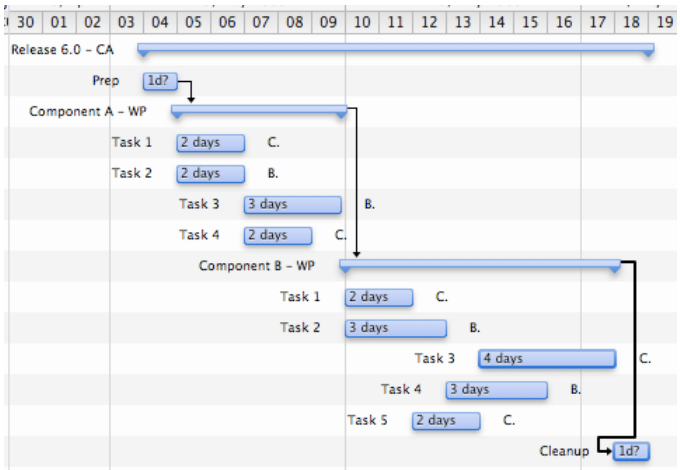

Fig. 3. An example of the task queuing technique using a set of mutually exhaustive tasks that are resource allocated is pictured. Notice that the definition of predecessors and successors is defined through the summary task at the work package level.

Currently, there seems to be little evidence of a set of EVM software that supports this type of task scheduling effectively. Future software sets may be more flexible to help provide value to project with high amounts of latitude in the selection order for tasks to perform.

D. *Cost Account Structure*

Some amount of rework during development is expected in any effort. If that rework is tracked under the cost accounts used for tracking the development of functionality, the uncertainty of the repair effort diminishes the ability for accurate tracking of progress across the project. Through the separation of the defect resolution into a level of effort cost account, the volatility can be removed from the tracking of the development effort while allowing some flexibility to address minor architectural changes.

Deliverables Based WBS

The WBS should be deliverables based in order to align the WBS to the work package structure of the cost accounts. On software projects the deliverables can be particular releases or specific modules or subassemblies depending on how the work will progress.

The alignment of the WBS to the deliverables of the project allows for the cost accounts to align to the deliverable as well. In this way the work breakdown structure provides a reasonable basis for the EVM process.

Repair and Maintenance

Separating the repair and maintenance costs in the work breakdown structure and then subsequently in the cost account structure allows for a separation between affects on project progress due to the repair of defects and affects due to poor estimation of tasks that lead to deliverables. This is extremely useful when trying to determine the cause of a project delay. Without this information efforts to correct issues with project progress may actually be harmful rather than helpful in many cases.

Management

Separating out the costs associated with managing the task is also a beneficial technique to ensure that measures of project progress are accurate. A management cost account can be created and tracked as a level of effort task in the EVM system. In this way, the regular expenditures associated with management will not give an artificial sense of cost growth associated with development of software code.

E. *Incremental Implementation Approach*

In order to reduce the amount of uncertainty in a software project, an incremental approach for implementing functionality is helpful. The guiding principle is that at each interim stage of development the product should be functional. This decision reduces volatility by ensuring that architectural dependencies are not so far reaching that they cannot be resolved until late in the project.

High Risk Tasks

When there are tasks that are high risk that may incur large costs or schedule delays or even result in a cancellation of a software effort as infeasible, they should be addressed in the schedule specifically. By taking those high risk tasks and moving them up into the schedule as early as possible, if the risk does occur, then there is more flexibility in forming a response. In addition, in the case where the risk occurrence results in the cancellation of a project, the costs incurred are minimized and made available for other efforts.

High Value Tasks

Tasks that deliver a high amount of value to the customer need special consideration as well. An incremental approach works to ensure that every delivery to the customer is in working order and can provide some value to the customer. It is in contrast to waiting until the completion of a large software effort that may take years to produce and will deliver no functionality until project completion.

With this type of approach it is important to schedule the functionality that is of the highest value to the customer early in the project. In this way, each incremental delivery has value to the customer. If the case is that the project is cancelled for budgetary reasons or de-scoped, value has already been delivered to the customer.

F. Bias Identification and Calculation

When a project progresses the initial guesses about the performance and progress indexes will be inaccurate. As more data comes into the system, those inaccuracies will resolve themselves resulting in a better view of the project and how it will continue to progress.

Estimation Bias

Estimates gathered on planned tasks are usually from development staff. Each member of the project team may have a different orientation when it comes to providing estimates. They may be optimistic or pessimistic to various degrees. However, we can assert that their estimates will be consistently optimistic or pessimistic, in about the same degree, over time. This is the estimation bias that every individual brings to the process.

The individual estimate, the particular staff members providing and the eventually the difference between the estimate and the actual value can be tracked. From these values a project manager can discover the estimation bias of the development staff members who are providing the estimates and use them to offset future estimates from those individuals.

Financial Bias

In many cases, there may be systemic biases that exist in the financial system for an organization or enterprise. These biases exist in many cases due to confidentiality with regard to compensation and other concerns. When actual project expenditures seem to be misaligned to known values of expended hours on particular tasks, if the issue is a financial systems bias, tracking those differences can result in more accurate future estimates.

G. Component Volatility Measures

A large software project may have several releases over time. The various components of the software as they are delivered in each release may have different levels of architectural volatility. A technique to help in the estimation of tasks is to qualitatively measure the volatility of a particular component.

For example, if there is a software component that is very atomic, there is no future scope that will need to be incorporated into the component in future releases, and the component is functioning in current releases, then the component could be qualitatively listed as low volatility.

Alternatively, if a component is not very atomic, with a large amount of dependencies on other components, has planned functionality to be added in future releases, and has not been released into operations, then the component could be qualitatively listed as high volatility.

A component diagram, or a functional diagram of the system can be used to display and track the qualitative volatility of the software components. This information can be used in future planning and estimation exercises.

IV. Applying the Solutions

How are the solutions spelled out in this paper applied to the challenges of applying earned value management to the development of software applications?

A. Innovation and Prototypes

When there are innovations and prototypes that are part of a software implementation, the uncertainty in their execution can wreck havoc with progress tracking techniques such as earned value management. Applying the solutions from this paper we can see how they can be addressed.

Effective Tracking of Innovations and Prototypes

Oftentimes developers and architects will be reluctant to provide estimates for such work because there is so much uncertainty. The approach here should be to focus the scheduling of the task on the various phases of the process of discovery that can be estimated. For example, the architect may need time to examine the requirements with respect to the functional design. They may need to design the architecture with respect to the information storage components, the user interactions that needs to be supported or the interfaces to existing systems. The design of the system with respect to a certain functional set may be needed. The breakdown of the work of design may not be an exact match to the work as it is performed, but going through the process of scheduling it in some form is very helpful.

Keeping to the goal of defining small tasks for scheduling that are on the scale of two to four weeks is important to apply in these phases of development.

This is one of the areas of a project where the temptation to re-plan the work once more information is known is very tempting, but it must be avoided. Planning packages should represent areas of work that are in the future. When the project discovers a discrepancy in the work that is being performed from what was planned, the technique should not be to throw out the existing plan and represent it with one that is perceived as more accurate, but rather to use the collected biases and apply them to the work that will need to be performed in the planning packages to allow the EVM process to provide a

more accurate estimate of future expenditures.

However, this does not mean that re-planning is not appropriate when there are large changes in the scope of the project. There are times when the prototype of innovation may lead to the discovery or limitation of potential scope items that could be part of the project. When a large change in scope occurs, the re-planning should be careful not to throw out or invalidate existing data that contributes to existing calculation of the CPI and SPI for the project.

Reducing the Impact of Innovations and Prototypes

Using the incremental approach to bring high risk items into the development process early helps alleviate the amount of rework that may occur later in the project. If the prototype work or an innovation takes longer than planned, or results in a large change in scope, there will still be time and resources to respond to the change if those tasks are performed early.

B. Defect Discovery and Resolution

All projects have defects that need to be resolved during the development process. When there is a lot of volatility in the progress of the development of a software system, it has an impact on the rates of defect discovery and may also introduce new defects into software that was previously seen as stable.

Development Defects

These defects are discovered and resolved during the development process so their resolution should be tracked as a part of the work package that describes the work associated with a particular release. These defects may still be tracked in a bug tracking system, but the work performed to resolve them should be charged to development accounts, not repair accounts. The reason being that these defects are too frequent and short in their resolutions to be seen as an activity that is separated from the work of code development.

System Test Defects

In an integrated development environment where there are dependencies on the work of different development staff, there is a need to track these types of defects in the repair cost account. In some cases, a project may be performing very regular test and integration, perhaps even nightly, and therefore the effort expended on resolving these defects consumes development resources, but it does not contribute to project progress. Tracking these items in a repair account will help to identify changes in the creation or discovery of defects. This can also lead to the identification of systemic process defects that need to be resolved.

Operations Defects

Operations defects more than any other type of defect need to have their associated costs separated from the cost accounts that are collecting expenditures for the development of software deliverables. The amount of time that it takes to triage, analyze and understand these defects also needs to be tracked as part of a cost account for repair. If it is not practical for a development process to separate out the costs associated with development or system test defects, operation defects must be separated if there is any chance to see the impact of a change in the defect density that has arisen from a project process change.

Effective Tracking of the Premium Incurred from Defects

When software projects follow a process throughout the life of the project, there will often be a stabilization of the number of defects introduced per thousand lines of code (the defect density). Using a technique to estimate this rate of introduction of defects will help in the initial planning of the cost collection mechanism for repairs. The initial sizing can come from an method of analogy, but once the project is in progress the actual discovery rate can be used to refine the budget estimate to be applied to the level of effort cost account for repairs.

Reducing the Effect of Defects Discovered

If there are a large number of defects that are discovered and need to be resolved, the resources to correct those defects are usually the same set of resources that are used for the generation of new features in the software system. In this way, the initial progress milestones may be met on a project, but the overall objectives for cost and schedule may not, as the resources needed for repairing the software will be taken from the resources dedicated toward meeting those cost and schedule goals. Separating the repair account, and tracking it as a level of effort task will not affect the CPI associated with the development cost accounts impacted by the increased defect density, although the level of effort costs may need to be adjusted. If a process of development does not consider the discovery and resolution of defects, it will be very difficult to meet the planned cost and schedule objectives, or subsequently those cost and schedule items will be passed on to be paid later, potentially at higher cost, to the maintenance process initiated after project closeout.

C. Architectural Changes

As software projects progress and as more and more functionality is added to the system being developed there can be instances where there is a large change to the architectural system. These changes can result in large amounts of rework that was not scheduled when the project was formulated.

Effective Tracking of Architectural Changes

In order to effectively track the progress of architectural changes, the adaptation costs for the architectural change to the existing modules should be captured under the development cost account and as part of the specific deliverable that the architectural change is needed to enable. In addition, the confidence interval should not be adjusted to reflect the risk in this work, the result of using PERT analysis in this area should capture the uncertainty for the estimation of tasks associated with the architectural change. Estimation biases that have been captured from estimates of previous work should be applied to the estimation data gathered in

order to improve it's accuracy and then apply an 80% confidence interval to the result for scheduling duration of the tasks.

For example, let's say that a software project had completed release 5.0 and delivered that version to their customers. The lead architect identifies a major architectural change that is needed for the next major release. PERT analysis is used in the estimation of the tasks, capturing the best case, most likely and worst case scenarios. The values gathered in the estimation of the tasks are then adjusted for the estimation biases of the developers who provided the information based on their previous estimates. The nominal and 80% confidence interval of those tasks is calculated and that is used for the scheduling duration of those tasks. The work of the architectural change is captured under each of the work packages for release 6.0 that are associated with development costs, leaving the repair and management accounts unaffected. The estimate for the defect density is calculated based upon the number of lines of code generated and the number of lines of code altered to estimate the number of repairs.

Reducing the Effect of Architectural Changes

In order to reduce the effect of architectural changes the iterations should be sized in order to give the team time to react to future instances of architectural changes in future releases. The goal should be that when there is a large amount of instability in the architecture, the releases that are planned and managed should be shorter in duration. In this way, when there is an overrun on schedule performance, the bias can be captured and the consideration of the unstable architecture can be an input into the planning for future iterations. If there are schedule losses to be suffered, they are incurred in early iterations and the later iterations do not suffer the same inaccuracies.

This is much more desirable than the situation that can occur when the iterations are large and have long time periods between deliveries. Long iterations can only be supported later in the project when there is a very solid sense that all of the biases have been discovered and are accounted for, as well as an assurance that the architecture is stable and does not manifest instances of needed rework. In a large iteration the amount of rework can compound itself when since many pieces of existing functionality may need to be readdressed.

V. Conclusion

There are a variety of challenges that face software projects when they are working to deliver high quality software on time and within budget constraints. Many of those challenges are related to a large degree of uncertainty, either in schedule duration, quality factors, or design issues. By applying techniques that help quantify the nature of the uncertainty, separate the distribution of uncertainty in project schedule and insulate the project budget from the effects of the uncertainty, projects can be successful in gaining value from EVM.

If a practitioner were using the AgileEVM method, the suggestion would be to augment the approach with a calculation of the uncertainty when assigning scrum points to particular efforts. In addition, using a confidence interval for the calculation of the point assignment may prove helpful and avoid issues that may occur on larger software efforts.

For PBEVM, since it is not specific to software, there are some suggestions for the adaptation to software. The use of the weighted milestone approach for software projects may prove helpful because of the nature and relatively small size of many software efforts. Of the possible quality measures, using the defect density and discovery rates as an important metric may also be beneficial for software projects. Finally, ensuring that there is a level of effort account that is created to collect cost for repairs and that it is sized based upon collected data and estimates for defect density.

Even in the case of applying traditional EVM practices, as the Next Generation Navigation Software project was, augmenting the EVM process by identifying, quantifying and isolating uncertainty as development is monitored and controlled may provide great value to software project managers.

References

[1] D. S. Christensen, D. V. Ferens, "Using Earned Value for Performance Measurement on Software Development Projects," *Acquisition Review Quarterly*, Fort Belvoir, Virginia: DAU Press Spring 1995, pp. 155–170.

[2] Q. Fleming, J. Koppelman, "Earned Value Project Management – A Powerful Tool for Software Projects," *Crosstalk – The Jourrnal of Defense Software Engineering*, Ogden, Utah July 1998. Available: http://www.stsc.hill.af.mil/crosstalk/1998/07/value.asp

[3] G. B. Alleman, "Project Management == Herding Cats, A Field Report, Agile Project Management," *PMFORUM, pmforum.org*, February 2003. Available: http://www.pmforum.org/viewpoints/2003/0203agilepm.htm

[4] G. Horine, "The Absolute Beginner's Guide to Project Management: Controlling a Project," *INFORMIT, informit.com*, October 2008. Available: http://www.informit.com/articles/article.aspx?p=1271256

[5] Y.K. Malaiya, J. Denton, "Requirements Volatility and Defect Density," *Software Reliability Engineering Proceedings, 10th International Symposium,* Boca Raton, Florida, November 1999.

[6] S. Baker, K. Baker, G. M. Campbell, *Complete Idiot's Guide to Project Management* Third Edition, Alpha Books, 2003, pp. 339