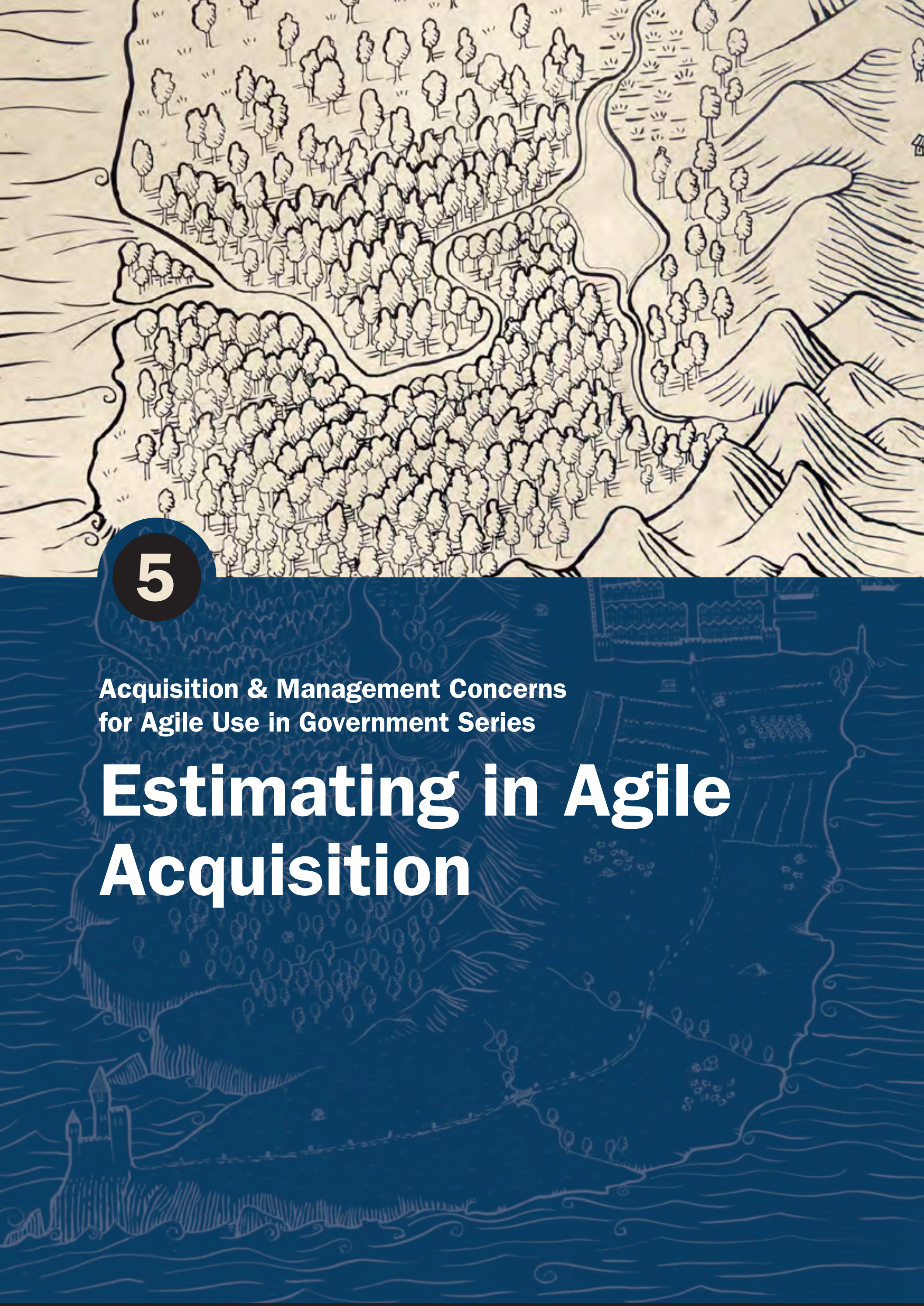

5

Acquisition & Management Concerns for Agile Use in Government Series

Estimating in Agile Acquisition

Software Engineering Institute

Carnegie Mellon University

Acquisition & Management Concerns for Agile Use in Government

This booklet is part of a series based on material originally published in a 2011 report titled *Agile Methods: Selected DoD Management and Acquisition Concerns* (CMU/SEI-2011-TN-002).

The material has been slightly updated and modified for stand-alone publication.

Booklet 1: Agile Development and DoD Acquisitions

Booklet 2: Agile Culture in the DoD

Booklet 3: Management and Contracting Practices for Agile Programs

Booklet 4: Agile Acquisition and Milestone Reviews

Booklet 5: Estimating in Agile Acquisition

Booklet 6: Adopting Agile in DoD IT Acquisitions

Estimating in Agile Acquisition

Introduction

Estimation activities occur throughout the DoD acquisition lifecycle. Estimates are used by all DoD programs in a variety of ways, and they are generated and processed in a variety of ways. Books have been written, inside and outside the DoD, on cost estimation for large, complex, software-intensive programs [Stutzke 2005]. We cannot deal with all the many connections between DoD estimation practices and Agile estimation in this report. We address the issues that we have most frequently seen and discussed, in interviews and through reviewers. Also, in all of the generalizations we make below, it should be understood that the needs and constraints of a particular program could result in estimates being treated in similar or quite different ways than what we describe in this booklet.

Some general estimation activities that government program offices support on many (though certainly not all) programs include:[1]

- **Producing a Program Life-Cycle Cost Estimate.** Prior to Milestones A and B, the Acquisition Program Office (the government) must develop a program life cycle cost estimate (PLCCE). The PLCCE is presented to the program's Milestone Decision Authority (MDA) at each milestone. The PLCCE must look forward from the current program state to the end of the system's life, and assess the cost of the product or system over its entire life.
- **Program Monitoring.** During source selection, the Acquisition Program Office may want to gain insight into the manner by which the bidding contractors have prepared their estimates. During contract execution, the Acquisition Program Office is constantly reviewing the performance of the contractor with respect to the contractor's estimate. This is typically done by reviewing the contractor's earned value management (EVM) data, although there will be further opportunities to review the contractor's estimation process each time an engineering change is processed.[2]
- **Transition to Sustainment.** After the system is fielded and in sustainment, there is generally a two-year cycle of maintenance, technology refresh, and upgrade.[3] These system enhancements are estimated and budgeted by the relevant program office, and by the contractor, who may or may not be the same organization that originally developed the system.

To understand the varied uses of estimates by the program office staff throughout the acquisition life cycle, and how these uses may relate to the use of an Agile development process by a contractor, the reader must understand estimation practices in general, and Agile estimation practices in particular. The next section

[1] The specifics for what is and is not required are defined in the FAR and DFAR for EV programs.

[2] How this occurs and the various ways programs can implement this estimation process is covered in the earned value management system description (EVMSD) and its work instruction.

[3] The timing and nature of sustainment activities is, of course, ultimately dependent on the particular program contracting and technical characteristics. However, we have observed this pattern in many programs.

of this booklet will begin to provide that insight and we will show how a government program office (including the program manager, the staff, the contracting officer's technical representative (COTR), and the procurement staff) could take actions with their cost estimating practices that would enable an Agile acquisition of a new system or sustainment of an existing system in the DoD.

Estimating to Support Request for Proposal (RFP) Preparation

The following discussion assumes that the government program office is acquiring software products (i.e., buying a system through a cost or incentive contract as covered by DFAR 234.201). We are not discussing acquiring software development capacity (i.e., software development expertise of a certain capability over a period of time), which is an alternate way we have seen government software needs being met. This model is more common in sustainment and operations and maintenance (O&M) programs. It generally consists of determining how many resources you can afford and how much capability those resources will allow you to build. This model is what some successful Agile programs have used, but it is not available to all programs.

During the RFP preparation phase of a new system acquisition, the government program office will make the decisions that are pivotal to enabling or disabling an Agile development contractor to bid and meet the program's needs. It is during this phase that the government program office will prepare its PLCCE, which will be based on the government's work breakdown structure (WBS). The prohibition during this timeframe against engaging with the development team when this is a competitive contract is a significant barrier to establishing the trust that is key to Agile project success; however, the considerations below could help to mitigate this issue.

If the program office wants to allow a developer using Agile methods to effectively compete, there are considerations that relate to both the acquisition strategy and its follow-on activities, as well as considerations related to execution of the Agile methods within the boundaries of the Program Management Baseline (PMB). From the acquisition perspective, the government program office must address how typical Agile methods artifacts fit into the traditionally specified artifacts of an acquisition, for example:

- The acquisition strategy should describe how the program office would interact with its contractor in order to provide the subject matter expertise needed on a continuous basis throughout the iterations of an Agile development.
- To ensure that the Agile acquisition strategy is enacted, the statement of work (SOW) or program work statement (PWS) must include language that allows the program office to provide subject matter experts with the ability to participate in the development of the software. This may be complicated by the hierarchical structure of contracts in a large system acquisition. The program life cycle cost estimate and budget must include funding for these subject matter experts throughout the development of the system. Because the SMEs usually come from government operational units, agreements must be crafted (e.g., memorandum of agreement [MOA], memorandum of understanding [MOU]) that make clear the expectations of participation of different stakeholders.

- The government program office must have a notional plan for what to do with the interim product releases that come from an Agile development process. Specifying these in the SOW is one way to emphasize the importance of working software being available in short iterations. There should be an evaluation environment established along with a feedback mechanism in place that permits the end-user community to try out these interim releases in a safe, secure environment, while waiting for required acceptance and certification testing activities to take place.[4]

Generally speaking, the most visible element of a software product estimate in DoD programs is the estimate of product size.[5] Even though modern software development tools and techniques reduce dependence on handcrafted source code, size is still frequently expressed in source lines of code (SLOC) or in function points. In Agile development environments, the development team may use “story points” as an alternative to either of these. Story points can be problematic in acquisition settings accustomed to SLOC or function points because they are explicitly a relative measure of size, not an absolute measure. Therefore, when story points are used outside of the team that generated them, it is necessary, though not trivial, to make some translation between story points and, typically, function points. Some of the programs we interviewed acknowledged they made the translation from story points to product size to provide cost estimates to those outside the development team. We saw proprietary tools that address this translation, and the commercial vendors for estimating tools are starting to address this new market need. In acquisition settings where trust has already been established between the contractor and the acquisition program office, this dependence on an absolute, versus relative, measure may be reduced.

Most parametric cost-estimation models base their outputs on software size, so errors in the size estimate will propagate into the estimate of effort and schedule. According to the GAO *Best Practices Guide for Estimation,* the keys to producing a defensible software cost estimate are (1) to have a reliable method for estimating the size of the product and (2) to employ a method for transforming the size estimate into an estimate of cost and schedule demonstrated to be accurate on similar projects [GAO 2009].

One popular parametric cost-estimation tool is the constructive cost model (COCOMO). According to Boehm, “COCOMO is an algorithmic-based parametric software cost-estimation model for estimating a software project as an ‘effort equation,’ which applies a value to tasks based on the scope of the project (ranging from a small, familiar system to a complex system that is new to the organization).

[4] Note that certification and accreditation (C&A) issues within the DoD acquisition life cycle are currently being addressed on multiple policy and implementation fronts, all with the goal of reducing the time, usually spent at the end of a program, to get the software system certified and then accredited by the appropriate governance body. We are not dealing with the specific requirements of the DIACAP process in this report.

[5] Software size may not be the most reliable predictor of software effort and cost (see Capers Jones, for example, who cites programmer skill as a better predictor of software outcome than size, among other attributes).

COCOMO II is the successor of COCOMO 81, incorporating more contemporary software development processes such as code reuse, use of off-the-shelf software components, and updated project databases" [Boehm 1981].

At the heart of the COCOMO II model are the cost parameters themselves. These parameters include five scale factors and seventeen effort multipliers. Scale factors represent areas where economies of scale may apply. Effort multipliers represent the established cost drivers for software system development. They are used to adjust the nominal software development effort to reflect the reality of the current product being developed.

It would be reasonable to assert that an Agile development process would have an impact on some of these parameters. For example, the COCOMO II model includes an effort multiplier for domain knowledge, or applications experience. The cost estimating multiplier based on the domain knowledge and capability of the software developer staff is called "application experience" (APEX). The rating for this cost driver is dependent on the level of applications experience of the project team developing the software system or subsystem. The ratings are defined in terms of the project team's equivalent level of experience with this type of application.

In an Agile development environment, there would be subject matter experts (users) participating with the system developers. The participation of users in the development process should improve the domain knowledge of the development team. The magnitude of the improvement can be assessed by changing the assignment of this effort multiplier, and observing the impact on the estimate.

A selected list of COCOMO II scale factors and effort multipliers is provided in the Appendix. Factors listed there that we would expect to be impacted by the use of an Agile development process include

- the development flexibility factor
- the architecture/risk resolution factor
- the team cohesion factor
- the analyst capability effort multiplier
- the programmer capability effort multiplier
- the application experience effort multiplier

The Appendix also contains information about Agile COCOMO, a 2004 prototype product that reflects some Agile estimation principles while relating back to concepts familiar to COCOMO users [CSSE 2011].

Among the many software estimation tools generally available (including Price-S, Software Lifecycle Management-Estimate [SLIM], and others) is the Software Evaluation and Estimation of Resources (SEER) model. It is one of those that actively updates its products to accommodate Agile estimation.

> SEER for Software (SEER-Software Estimation Model [SEM]) is an algorithmic project management software application designed specifically to estimate, plan, and monitor the effort and resources required for any type of software development and/or maintenance project. SEER, which comes from the noun referring to one having the ability to foresee the future, relies on parametric algorithms, knowledge bases, simulation-based probability, and historical precedents to allow project managers, engineers, and cost analysts to accurately estimate a project's cost schedule, risk and effort before the project is started [SEER-SEM 2011].

For Agile projects, SEER uses three kinds of estimates. These are planning, forecast, and working. The planning estimate is still used to determine how big the project will be and is usually based on analogies of previous projects of similar size. The forecast estimate is accomplished after you have built your backlog. Several things can be defined at this time, such as incremental delivery, release cycle, the length of the iteration, exit criteria for a deliverable, and negotiation for scope change requests. (Baseline change requests accomplish this in the DoD acquisition cycle.) Finally, working estimates are done for all iterations after the first iteration is complete. This allows assessment of the team and customer as well as an understanding of the individual team velocities.

SEER, like COCOMO, uses a variety of parameters for their model, including

- requirements formality
- requirements volatility
- personnel capabilities – analyst and programmers
- familiarity with the process
- process maturity
- staffing complexity
- development system volatility
- automated tools usage
- testing level
- quality assurance participation
- infrastructure and tooling costs

Before the build, your estimate considers these parameters in relationship to your team. We recommend that you revisit your forecast estimate as your team changes.[6]

[6] DeWitt, D. Demystifying Agile Project Cost and Schedule Estimates. Webinar. Galorath Incorporated, 2010.

Source Selection

In the source selection phase of an acquisition, the program office will have to evaluate estimates that are prepared by the bidding contractors. In many cases, the program office will seek to understand the contractor's process for producing the estimate. It is very important for the program office to establish a high degree of confidence in the bidding organization's estimation process.

The following discussion focuses on a notional Agile estimation process from the development estimator's viewpoint. We have synthesized this description from our various interviews and include some clarification information from the Agile literature to help readers new to Agile methods relate the Agile approach to knowledge they already have from using traditional estimation practices. We hope that this approach will enable government program office personnel new to Agile approaches to gain insight into why estimates for an Agile project may look different from traditional ones.

Estimating from the Development Estimator's Viewpoint

We focus this section on the development estimator's viewpoint, which could either be for a government organization (such as an Air Logistics Center of the U.S. Air Force), or a commercial development contractor. In either case, the viewpoint is based on knowledge of the team that will be producing the software, knowledge of the tools and development environment that are available, as well as knowledge of the practices that are intended to be used. We also distinguish between new software development estimation and sustainment-focused estimation, since the basis of each is different.

In the case of new software development (some new feature being implemented in software for the first time or a significant upgrade to existing software being treated as its own project), some initial work will need to be estimated for creating an overall architecture that will be the basis for the rest of the project. That architecture will determine some of the requirements prioritization, though not all of it. Overall system design is outside the normal scope of software development estimation, so some ideas of architecture and its implications may be established prior to estimation. In any case, working the initial aspects of the architecture and platform infrastructure is usually estimated separately from the actual requirements implementation, and in Scrum, the most commonly used Agile project management method, this is usually called "Sprint 0" [Ozkaya 2011].

Often, especially in the DoD programs we interviewed, the early iterations and stories are more about building the infrastructure needed to ensure a stable architecture than about delivering end-user functionality. If using the RUP as a framework for an Agile project, this kind of work is done during the Inception and Elaboration phases. In cases where this was necessary, some of the programs we talked to mixed infrastructure building with end-user functionality, so that end users received working software at least every other release. Others coupled architectural infrastructure elements to end-user functionality so they could deliver on just a piece of the architecture. In either case, the emphasis was on ensuring that end users saw progress quickly and frequently. Once a general pattern of releases was generated, a more detailed estimation of future releases and sprints occurred.

After user stories are generally prioritized, they become a product backlog. From the product backlog, releases are constructed that deliver evolving capability to the end users. Each release has a nominal set of user stories (based on team velocity, vital factors, and initially estimated story points). Up to this point, the estimation has been coarse-grained, since it is known that user priorities will change over the course of a project, especially one that is longer than one year.

From the product backlog reflected in the first release, the user stories for the next iteration within that release are selected (a process sometimes called “grooming” the backlog) and the team working on each story does more fine-grained estimates of the appropriate story points for that release. Based on the team velocity, an estimate of feasibility is made as to whether the proposed set of user stories can be built within the iteration timeframe.

In most Agile methods, the end users and other project stakeholders are present in the iteration-planning meeting where these issues are discussed, so that re-prioritization can occur if necessary. These meetings also enable an essential element of Agile methods: the development team and the end users decide on the character and timing of user/developer working sessions. This kind of joint decision making is one of the things that the programs we interviewed emphasized as being essential to their progress.

This rhythm of each iteration being estimated at a fine-grained level while releases and the overall project are estimated much more coarsely actually reflects the common practice in DoD cost accounting discussed earlier: rolling wave planning [Department of Defense 2011]. More detail on this part of the process is found in the section of this booklet titled *Contract Execution and Monitoring.* One important aspect of rolling wave planning related to estimation is that the period of performance covered by the rolling wave must align with the iterations in the life cycle so that planning does not occur, for example, for only half of an iteration.

In the case of sustainment or enhanced legacy software, if the architecture is stable, then prioritizing the known requirements is a first step in estimating. In Agile methods, these are gathered as user stories—descriptions of discrete functionality known to be needed by a particular user segment that is part of the project’s audience, and other stories that address infrastructure and quality attributes that are pervasive to the product (e.g., security or usability). Although user stories are generally constructed to be discrete and separable (one of the things that permits reasonable prioritization), they can often be bundled into a related feature set to be delivered, called an epic. It is not unusual for a release to be defined by the completion of one or more epics. Where the user stories come from (government operators or contractor subject matter experts), is highly dependent on the contracting vehicle and agreements that are in place for the effort.

Evaluating Estimates from the Acquirer's (Source Selection Team) Viewpoint

In this section, we change focus from what an Agile estimation experience looks like from the development estimator's viewpoint to what it looks like from the estimate evaluator's viewpoint, usually the source selection team or other members of the government program office.

The biggest difference between evaluating an estimate for an Agile project and a traditional project is that the Agile project admits up front that not all requirements can be known early in the project and so the overall estimate will be amended as more knowledge is gained. Where a contract vehicle has been constructed that allows these amendments to occur without having to process baseline change requests, (such as a time and materials contract type) the overall process has been easier for both acquisition personnel and the development contractor.

Estimates for near-term activities—usually through a single release—can be made more accurately than the typical traditional project because the period for estimation is usually less than four months. The four months is the equivalent of eight two-week iterations, an approach consistently used for several years on one of the programs we interviewed. The team's capabilities, in terms of how quickly they can typically address a story point's worth of work, are well understood after the first couple of iterations. This accuracy is dependent on knowledge of the team's progress characteristics.

In discussing government evaluation of development contractor estimates in Agile projects, we gleaned that the questions in the following list were considered useful by a variety of our interviewees. Not all programs used all questions; this list is a union, not an intersection, of the questions. Which questions apply in a particular acquisition situation also depends on the acquisition strategy decided upon and the contract vehicle used. Not all of these questions can be used for all contract types. Some of them (e.g., the first one) assume that the developer already understands and has worked in Agile projects, while others do not make that assumption. The questions different programs ask about an Agile project's initial development estimates include:

- If the project involves new software development, did the development team leave separate time for constructing the product's architecture and infrastructure needed (e.g., the continuous integration and test environment) to operate the project?
- Do the initial user stories adequately reflect the known end-user project priorities, tempered by any programmatic constraints that have been shared with the estimator? (Clearly they will not reflect those that are unknown at the time of estimation.)
- Has the team performing the work used Agile methods before as a team? If so, do they have evidence of their velocity on similar projects? (If they have not worked on similar Agile projects before, calculating velocity from their individual performances would be inappropriate and misleading.)

- If this team has not worked together before, how did they derive their velocity?
- If this team has not used Agile methods before, have they left some slack to account for a learning curve?
- Does the estimate include frequent opportunities for user feedback (e.g., pre-release demonstrations of working software at the end of each iteration)?
- Does the estimate include time for side-by-side working sessions with end users during iterations?
- Does the estimate characterize the “vital factors,” such as distributed team, new project domain, complex operational processing, and other factors, and how they affect the estimate? [Bhalareo 2009]

These factors are somewhat different from the COCOMO II factors that estimate evaluators may be familiar with, but they bear some relationship and may be able to be resolved (though we have not run into this in interviews with Agile DoD programs so far).

Also important for evaluators to remember is that you will be receiving new estimates for each release or iteration depending on the project norms and contract vehicle. This gives you the opportunity and an obligation, as an acquirer, to reevaluate requirements priorities (via the product backlog) based on user feedback for the most recent releases. Depending on the project, releases for informal early adopter use, usually in sandbox environments, may happen as often as every two months.

Note that from an acquisition life cycle viewpoint, the releases we are talking about here are generally development releases, so the user community intended to receive them must be carefully selected. Our interviewees usually had subject matter experts on the development team who were knowledgeable about certification requirements for their software and who participated in identifying the appropriate user audience for different classes of release. Certification requirements are a type of constraint that can prevent early release of software, even on a development basis. The effect of these interim releases on estimation varies. Depending on the constraints of the contract, interim releases may be accomplished easily and often, or they may be almost as much work as a fully deployed release to fielding.

Contract Execution and Monitoring

When working with any development contractor, it is important to understand how the work is being planned and executed, so that the program office can understand how to interpret the progress data provided by the contractor. When working with an Agile contractor, it is especially important for the acquirer to understand the methods and techniques employed by their contractor, as it is likely that the techniques used by the contractor will be new to the program office. This understanding provides the foundation for any discussions between the acquirer and the contractor. In addition, this understanding does not replace specific constraints or directions levied by the FAR/DFAR, but it allows the acquirers to understand the implications of their contract vehicle in the Agile environment. This section of the report provides insight into the techniques often used to plan Agile projects, mostly from the development team viewpoint, and into the techniques used to monitor and control Agile projects, mostly from the acquirer's viewpoint.

Story Point Estimation

In Agile projects, user stories and technical stories are typically estimated in story points. Story points are commonly used in several Agile methods for estimation at both the release and iteration levels. They do not use lines of code as their base unit of measure. Tasks, on the other hand, are generally estimated in hours and are used only for detailed iteration-level planning. Tasks are the activities that developers determine will be necessary to successfully complete the story. If you are evaluating developer estimates, being able to understand the source of the developer estimates can improve your ability to interpret them.

The following is a common definition of story points:

> Story points are a unit of measure for expressing the overall size of a user story, feature, or other piece of work.... The number of story points associated with a story represents the overall size of the story. There is no set formula for defining the size of a story. Rather a story-point estimate is an amalgamation of the amount of effort involved in developing the feature, the complexity of developing it, the risk inherent in it and so on. [Cohn 2006]

One of our reviewers commented, "This [concept] is really important as it can thwart meaningful comparison and tracking of trends. It certainly can undermine the ability to do cross-team comparisons."

It is important to note that story point estimates are both relative and local. They are relative in that estimates are typically derived by comparing the size of one story to another or by assigning a point value to one or more reference stories, which are then used to calibrate the sizes of newly created stories. Story point estimates are *local* in that different teams may arrive at different sizing conventions. A story that is assigned five points in team A may, for example, be assigned three points in team B. One implication of this is that, for most DoD programs, at some point estimates must be converted from relative to absolute estimates, especially for programs using EVM (earned value management).

Story point estimation is typically conducted as a team-based activity and is guided by defined techniques. Two popular team-based estimation techniques are Planning Poker and the Team Estimation Game [Larman 2004]. In Planning Poker, stories may be assigned point values of 1, 2, 3, 5, 8, 13, 20, 40 or 100 (an adaptation of a Fibonacci series). Other Agile estimation techniques use similar scales. The spacing between the point values is designed to reflect both the principle that "we are best at estimating things that fall within one order of magnitude" and "greater uncertainty is associated with estimates for larger units of work" [Cohn 2006]. Stories planned for incorporation within an upcoming iteration will typically be assigned point values at the lower range of the estimation scale while stories coming later will be assigned point values at the higher ranges, especially if they reflect a lack of knowledge until some of the earlier stories are executed. In addition to using story points to estimate effort, at least one Agile author (Larman) recommends that stakeholders independently estimate story point value at the same time developers are estimating effort, allowing for an explicit prioritization of effort for value [Larman 2004].

While story points are the most widely advocated metric for story and feature size estimation, some within the Agile community also advocate for the use of "ideal days" for this purpose. Similar to story points, ideal days are intended to be used as a sizing estimation metric, expressed as the number of days a story or feature would take to develop, assuming

> The story being estimated is the only thing you'll work on.
>
> Everything you need will be on hand when you start.
>
> There will be no interruptions [Cohn 2006].

It is important to note that when estimating in ideal days, as with story points, the estimate is intended to include the aggregated work required from all team members for all tasks required to successfully complete development (e.g., elaborate story details, write unit tests, design, code, build, execute acceptance tests, write required user documentation). In addition, as with story points, estimates in ideal days are a local metric. Once again, a story that is assigned five ideal days in team A may, for example, be assigned three ideal days in team B.

Velocity

As discussed above, both story points and ideal days are relative, local sizing metrics, rather than objective projections of effort and duration. Therefore, story points cannot be used directly for absolute estimation purposes. Rather, within Agile practices, story points provide input to the calculation of other measures like team "velocity," which is in turn used to derive estimates for releases and iterations. As stated by Mike Cohn, "…a key tenet of agile estimating and planning, is that we estimate size and derive duration"[7] [Cohn 2006]. However, unlike traditional projects, Agile projects estimate *relative* size, rather than *absolute* size. Current expressions of estimates within DoD programs use absolute estimates of size and duration, requiring translation from Agile estimation approaches, as we have mentioned previously.

"Velocity is a measure of a team's rate of progress. It is calculated by summing the number of story points assigned to each user story that the team completed during the iteration. If the team completes three stories, each estimated at five story points, their velocity is fifteen. If the team completes two five-point stories, their velocity is ten."[8]

[7] It is worth noting that deriving effort from size is a common way of estimating software projects in traditional methods as well.

[8] DeWitt, D. Demystifying Agile Project Cost and Schedule Estimates. Webinar. Galorath Incorporated, 2010.

Mike Cohn describes three potential options for estimating the velocity of a given team.

Use historical values.

Run an iteration.

Make a forecast [Cohn 2006].

Each of these activities takes place within a particular context and is based on specific assumptions, such as team skill and history with the domain. Velocity is sufficiently tied to the specific team's characteristics that cross-team velocity comparison can be misleading.

Running an iteration is a common approach to learning about a team's velocity in the commercial space. Depending on how the contract is constructed, this may or may not be an option within a DoD contract.

Agile Release Planning

The Iron Triangle of cost, time, and scope is fundamental to traditional release planning. An Agile perspective on this triumvirate is expressed by Dan Rawsthorne in the following equation:

Time x Capacity = Scope

where

Time = # of iterations * iteration length

Capacity = average velocity per iteration[9]

Scope = total # of story points that can be completed in the release

Using the above equation, a team with an iteration length of two weeks and an average velocity of 30 could complete 300 story points in approximately 10 iterations or 20 weeks. While seemingly straightforward, this equation must be understood within the context of the Agile approach to project scoping [Rawsthorne 2010].

Whether developing within a traditional or an Agile methods environment, the first step in planning any release is to establish the high-level goals and purpose of the release. That purpose may include delivering capabilities to a particular group of stakeholders, increasing customer satisfaction, or gaining market share. Once the goals and mission are established, the focus then turns to scoping the release contents.

On a traditional project, establishing scope for a release begins with the creation of a detailed requirements specification. Within an Agile project, establishing scope for a release begins by examining the product backlog. The product backlog is the name commonly used for the repository of stories associated with a given product or

[9] Note that there is an implicit assumption that capacity does not vary. And while it is true that it varies less if the team is stable in terms of membership and type of tasks performed, capacity is quite likely to change when the domain, programming environment, or other significant environmental factors change, even if team composition does not.

project. If the project is a completely new start, a new product backlog will have to be defined by delineating the user and technical stories relevant to that project. Elements within the product backlog may also include features, capabilities, and defects, as well as stories. It is a recommended (although not universally adopted) practice to attach story point estimates to all items within the product backlog [Cohn 2008].

For an Agile project, scope is often expressed in stories. The major activities involved in scoping and planning an Agile project include:

- selecting features and stories from the product backlog for incorporation into the release (some features will already be expressed as stories, depending on their prior history)
- decomposing features into stories that reflect each feature's intended business value (in doing this, it may become clear that some stories must take precedence over others; stories that are not suitable for the current release get returned to the product backlog)
- decomposing larger stories into smaller stories that can be completed within a single iteration (again, after this step, some stories may be returned to the product backlog)
- assigning a story point value to each "iteration-sized" story (although story point values may have already been assigned within the product backlog these estimates will typically be re-examined and validated during release planning)
- prioritizing stories and assigning them to specific iterations

Whether all of these activities are done upfront at the start of the release or whether some are conducted on a per-iteration basis will depend upon the team, the project, and the associated program expectations and constraints. "Some teams in some environments prefer to create a release plan that shows what they expect to develop during each iteration. Other teams prefer simply to determine what they think will be developed during the overall release, leaving the specifics of each iteration for later. This is something for the team to discuss and decide during release planning" [Schenker 2007].

It is important to note that even after stories have been broken down and estimated, they generally are not specified to the degree that would be found in a traditional requirements document. This does not necessarily imply increased risk, because successful Agile teams rely on ongoing dialog with users, user proxies, and subject matter experts throughout the course of the release to gain insights needed to satisfy the users, usually better insight than could be gained from typical requirements-specification documents. If the user interaction that makes this dialog possible is missing, the benefits associated with user stories as an anchor for the requirements will be lost.

Prioritization of stories across iterations is another important aspect of release planning. Cohn identifies the following four factors as critical considerations during prioritization [Cohn 2006]:

- value of the story
- cost of developing the story
- knowledge generation, including
  - knowledge about requirements, the domain, and user needs
  - knowledge about the underlying product technology, architecture, and design
- risk, including
  - technology risk
  - business risk
  - schedule risk
  - cost risk
  - functionality risk

While a certain amount of prioritization will take place during initial release planning, on Agile development projects, prioritization is ongoing and stories are often reprioritized at the end of each iteration. Successful adoption and execution of this dynamic approach to prioritization once again requires a close relationship and ongoing dialog with program stakeholders.

The role of the product owner (usually played by the acquisition program manager) in release planning cannot be underestimated. Product owners resolve the concerns of multiple stakeholders with conflicting priorities. They maintain the integrity of the product and ensure that it actually delivers the promised value to end users. In release planning, they often know the most about the programmatic constraints that must be met prior to release to the end user, and they are the people who will have to seek waivers or other relief if needed from processes that disable a project's intended Agile practices from working.

Agile Release Tracking

As discussed previously, Agile release planning relies upon the following three factors:

1. the team's estimate of their projected average velocity for the release
2. the set of stories selected for inclusion in the release (i.e., the stakeholders' estimate of the desired release contents)
3. the sum of the story point values for the stakeholder-selected stories

Referring back to the previous example, if the team estimates its velocity at 30 story points per two-week iteration and estimates the sum of the sizes of the stakeholder-selected stories at 300 points, then the release should take 10 iterations or 20 weeks.

Agile release tracking focuses on these same three factors and examines how closely the initial estimates are tracking to actual results. Agile release tracking, therefore, asks the following three questions [Rawsthorne 2010]:

1. Is the team's velocity tracking to its initial estimates (i.e., how many story points have been completed to date and how does this compare to the plan)?
2. Have the stakeholders added stories or removed stories from the release? If so, have these changes increased or decreased the sum of the story points for the release?
3. Has the team changed its point value estimates for any stories?

One of the most commonly used charts for tracking progress on Agile releases is the release burndown chart. On the release burndown chart, the x-axis is expressed in iterations, while the y-axis is expressed in story points remaining to be completed. Under ideal conditions, a release burndown chart for our sample project, with 300 story points and a velocity of 30, would appear as follows:

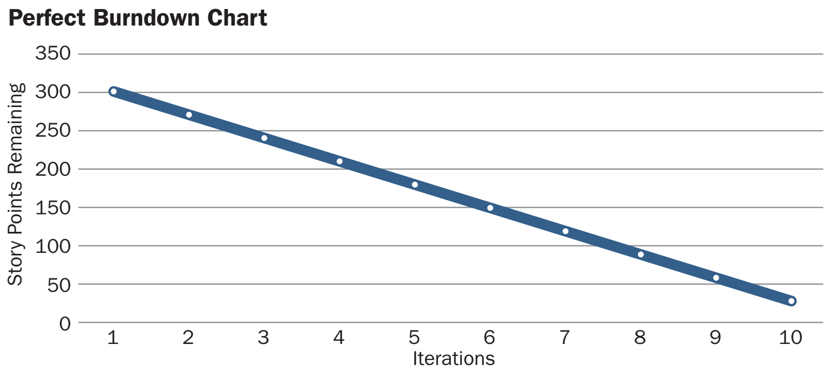

Figure 1: Perfect Burndown Chart

In reality of course, no project will ever execute in precise conformance to initial estimates. Therefore, it is more likely that by iteration 5, the release burndown chart for the project will look something like this:

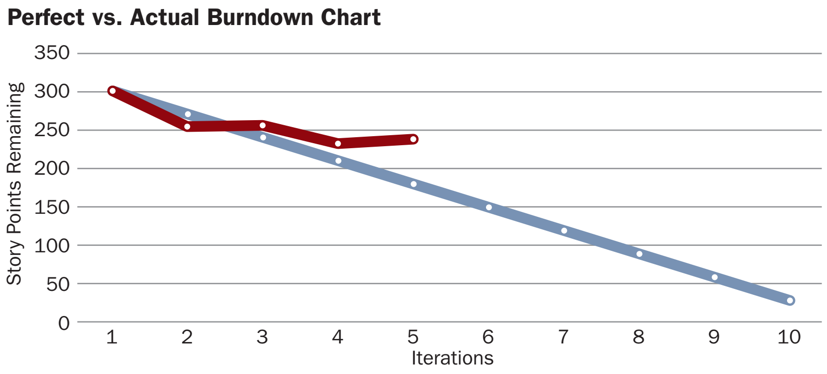

Figure 2: Perfect vs. Actual Burndown Chart

This chart for our sample project clearly shows that by the end of iteration 5, we have more stories remaining to complete than we had originally planned. However, the chart itself does not give an indication of why this is the case. Any of the three factors discussed above could be behind the discrepancy:

1. The team's velocity could be less than initially anticipated.
2. The project stakeholder may have added stories to the release.
3. The point estimates for certain stories may have increased as the team gained further knowledge of the technology and the domain.

The chart therefore provides an early indicator of potential future issues, but only discussions with the development team will reveal the reason for the discrepancy and what actions, if any, need to be undertaken. Other visualizations can increase the insight into reasons behind a particular burn down phenomenon, which are discussed in detail in Cohn's *Agile Estimation and Planning* [Cohn 2006].

As with any other progress tracking method, using user stories to generate velocity measures can lead to some anomalous results. For example, if the user stories are more than an order of magnitude sizing difference during an iteration, velocity could appear lower than is warranted. This sizing difference could also result in one story taking an inordinately long time to complete, possibly even resulting in a velocity of 0. A development team should develop its own norms in terms of the relationship of the number of stories to the number of team members to iteration duration.

Verification and Validation

The amount of total effort estimated for verification and validation (V&V) activities may not be that different in amount when comparing Agile and traditional projects. However, the timing of verification and validation activities is expected to be different and that should be reflected in the way the CDRLs are handled for the contract. Most V&V estimates for traditional projects show a bimodal distribution of effort—high at the beginning when test plans and environments are being determined, low in the middle during design and implementation, high at the end during execution of verification and validation activities. However, most Agile methods involve some type of continuous integration and testing, and some methods, like test-driven development, actually demand that test cases be written before designs are implemented in code. Thus, the profile of V&V activities may well look more like a steady level of effort than a bi-modal distribution of effort.

Some of the projects we interviewed included a separate iteration for acceptance testing, including, if appropriate, some of the information assurance (IA) testing that is required for certification. (Note that although information assurance is a specialty engineering discipline that is involved throughout the project, there is a certain amount of testing for IA that usually occurs as part of the overall V&V effort.) Others considered acceptance, certification, and other operational testing to be outside of their Agile life cycle and their delivery to those testing environments was the completion of their Agile project life cycle, other than rework that was required to address defects found in the acceptance test cycle. The decision about how to treat V&V is a contract-specific issue and, as can be seen from some of the variants expressed here, the effects on estimation will be determined by which process and method selections are made.

Agile EVM (Earned Value Management)

Earned value is one of the primary tools that the Department of Defense uses to measure contractor performance. For programs valued at more than $20 million, an earned value management system (EVMS) is required to be used, and for programs more than $50 million, a validated EVMS must be used. “EVM techniques, however, assume complete planning of a project to discrete work package levels, then assigning cost and duration to these packages” [Sulaiman 2006].

It should also be noted that the application of traditional EVM methods within the DoD acquisition process is currently being reexamined.

> EVMS has experienced a number of issues, notably with contractor implementation and data quality. However, for the Panel’s purposes, the most significant limitations are that EVMS only measures the performance of a contractor, not of the organization which is managing the acquisition. Furthermore, EVMS would generate no negative information about a contractor performing on cost, on schedule, and meeting all contract requirements even if (or perhaps especially if) the contract in question had a wildly inflated price or a schedule or set of contract requirements that utterly failed to meet warfighter needs. Thus, EVMS, while a valuable tool, is not sufficient to fulfill the Panel’s recommendations [House Armed Services Committee 2010].

Accommodating the Agile principles of incremental and adaptive planning, and embracing change in the pursuit of value, can be challenging, especially when faced with the significant implementation guidance related to EVM that mentions nothing about its use in Agile projects. AgileEVM is a new, exploratory practice area within the Agile development community. Proponents suggest that EVM may be applied usefully and validly to Agile software development projects. Proponents also believe that AgileEVM addresses some of the above-mentioned shortcomings of traditional EVMS. However, for AgileEVM to work, it is important that tasks are small and that iterations are short. The most comprehensive treatment to date of AgileEVM may be found in an IEEE Software 2006 article, entitled “AgileEVM—Earned Value Management in Scrum Projects” [Sulaiman 2006]. The described method computes AgileEVM for a single release of software and makes use of story points as the fundamental units of work and the fundamental units of earned value.

The above-referenced method of calculating AgileEVM requires the development team/contractor to supply the following data prior to the start of development:

1. performance measurement baseline (PMB)
   (expressed as total number of story points planned for the release)

2. schedule baseline
   (expressed as total number of sprints planned for the release * length (in time) for each sprint)

3. budget at completion
   (expressed as the total budget planned for the release)

During project execution, the following data is collected on a per-iteration basis and used to generate updated AgileEVM calculations:[10]

1. story points completed
2. story points added
3. iteration cost

The above-described method covers the generation of all standard EVMS equations. The assertion that AgileEVM addresses shortcomings within traditional EVMS is based upon the following: AgileEVM calculations are based upon delivery of completed, tested units of functionality. No credit is given for delivery of intermediate work products. Therefore, AgileEVM may be seen as incorporating quality standards into the metric and may be seen as providing stricter evidence with respect to delivery of value.

Because the performance measurement baseline (PMB) is expressed as “number of story points planned” rather than at the level of specific tasks, it allows course corrections to be made without disruption or re-baselining of the PMB. This addresses the criticism expressed in the Defense Acquisition Reform Findings and Recommendations (DARFAR) report regarding the inability of traditional EVMS to identify issues related to “contract requirements that utterly failed to meet warfighter needs” [House Armed Services Committee 2010].

Sustainment

Sustainment of existing software-intensive systems—corrective maintenance and evolution of capability—is a large part of the software activity performed by or on behalf of the U.S. Department of Defense [Defense Acquisition University 2011c]. Agile methods have been successfully used in sustainment as well as new developments in commercial industry, and in fact, some of the program offices that we interviewed either started as sustainment projects or transitioned into sustainment projects during the course of the project’s life cycle. One of our reviewers commented, “Agile is perfect for continuous maintenance, [including] many of the NASA Deep Space systems.” Among other benefits, one reviewer commented that, for programs they had worked in an Agile fashion for both development and sustainment, “…there is very little change in process or planning artifacts when a product transitions from development to sustainment. This can save an enormous amount of time and money.” There is also, generally, alignment between a sustainment effort’s periodic releases for patches and the short iterations used in Agile methods.

[10] One of our reviewers who has used AgileEVM noted, “This is fine as long as the iterations are short. When an iteration is longer than about three weeks, it will be important to calculate percent complete of an iteration based on percent of story points planned for the iteration that are complete to this point in the iteration. This is a type of “information radiator” that can be implemented that basically shows current percent complete of the iteration.”

In sustainment contexts for IT systems with long life, contracting mechanisms tend toward service contracts, in which the contracted element is the staffing of a set of skills anticipated to be needed to sustain the software at a certain capacity. Projects we interviewed in these kinds of sustainment contexts found estimating and tracking using Agile methods and measurements to be useful to the customer as well as the development team. This is because of the strong communication between end users and the development team that resulted in a deep understanding of the priorities of the user community being served and a commitment to providing as much value as possible.

In service contracts of less than $20 million EVM threshold, where Agile methods were in use, the use of story point estimation and the formulating of iterations based on product backlog (often consisting of requests and defect reports from the field) were consistently in use. Much of the content in the **Contract Execution and Monitoring** section applies equally well to sustainment situations as to new start situations.

The biggest difference in sustainment is that an architecture for the system has been defined and implemented. Depending on how well it has been communicated to the sustainment team, there may be constraints on the team's ability to evolve the product to serve end-user needs. This is because the team also needs to adhere to the architectural constraints that are often in place to meet security or other non-functional requirements that may not be obvious to a team taking over an implemented product. In this situation, there may be iterations that are needed from time to time that are expressly focused on evolving the architecture to address new infrastructure or quality attribute requirements. From an estimation process viewpoint, these iterations are likely to be estimated using either non-user technical stories, or some other estimation method, such as ideal days.

Conclusion

The biggest difference between evaluating an estimate for an Agile project and a traditional project is that the Agile project admits up front that not all requirements can be known early in the project and so the overall estimate will be amended as more knowledge is gained. Where a contract vehicle has been constructed that allows these amendments to occur without having to process baseline change requests, (such as a time and materials contract type) the overall process has been easier for both acquisition personnel and the development contractor.

During the RFP preparation phase of a new system acquisition, the government program office makes decisions that are pivotal to enabling or disabling an Agile development contractor to bid and meet the program's needs. During this phase, the government program office prepares its PLCCE, based on the government's work breakdown structure (WBS). While standard government acquisition procedures provide several barriers to establishing the trust that is key to Agile project success, it is possible to mitigate this issue and produce an estimate that is consistent with Agile methods, using the approaches described in this booklet.

Effective estimation begins with an acquisition strategy that describes how the program office will interact with its contractor in order to provide the subject matter expertise needed on a continuous basis throughout the iterations of an Agile development. The statement of work (SOW) or program work statement (PWS) should include language that allows the program office to provide subject matter experts with the ability to participate in the development of the software. The program life cycle cost estimate and budget must include funding for these subject matter experts throughout the development of the system. And the government program office must have a notional plan for what to do with the interim product releases that come from an Agile development process. Specifying these in the SOW is one way to emphasize the importance of working software being available in short iterations.

References

[Bhalareo 2009]
Bhalareo, S. "Incorporating Vital Factors in Agile Estimation through Algorithmic Method," *International Journal of Computer Science and Applications 6,* 1 (2009) 85-97.

[Boehm 1981]
Boehm, B. *Software Engineering Economics,* Prentice Hall, 1981.

[Cohn 2006]
Cohn, M. *Agile Estimating and Planning.* Addison-Wesley, 2006.

[Cohn 2008]
Cohn, M. "When Should We Estimate the Product Backlog." *Mike Cohn's Blog – Succeeding with Agile* (March 16, 2008). http://blog.mountaingoatsoftware.com/when-should-we-estimate-the-product-backlog

[CSSE 2011]
Center for Systems and Software Engineering, University of Southern California. *Agile COCOMO II.* http://csse.usc.edu/csse/research/AgileCOCOMO/ (2011).

[Defense Acquisition University 2011c]
Defense Acquisition University. Ch. 4.3.6, "Evolutionary Acquisition Programs," 280-281. Defense Acquisition Guidebook. http://at.dod.mil/docs/DefenseAcquisitionGuidebook.pdf. Accessed Sep 12, 2011.

[Department of Defense 2011]
Department of Defense. Section 2.5.2.4.1, "Guidance," 57. *Earned Value Management Implementation Guide.* Department of Defense, 2006. https://acc.dau.mil/CommunityBrowser.aspx?id=386074&lang=en-US. Accessed July 13, 2011.

[GAO 2009]
Government Accountability Office. *GAO Cost Estimating and Assessment Guide: Best Practices for Developing and Managing Capital Program Costs* (GAO-09-3SP). United States Government Accountability Office, 2009.

[House Armed Services Committee 2010]
House Armed Services Committee. *House Armed Services Committee Panel on Defense Acquisition Reform Findings and Recommendations* (DAR Final Report [3-23-2010]). United States House of Representatives, 2010.

[Larman 2004]
Larman, C. *Agile and Iterative Development: A Manager's Guide.* Addison-Wesley, 2004.

[Ozkaya 2011]
Ozkaya, I.; Brown, N.; & Nord, R. Ch. 3, "Communicating the Value of Architecting within Agile Development," 11-22. *Results of SEI Independent Research and Development Projects (FY 2010)* (CMU/SEI-2011-TR-002). Software Engineering Institute, Carnegie Mellon University, 2011. http://www.sei.cmu.edu/library/abstracts/reports/11tr002.cfm

[Rawsthorne 2010]
Rawsthorne, D. *Agile Release Planning and Monitoring.* CollabNet, 2010. http://www.open.collab.net/media/pdfs/SBU_ReleasePlanning.pdf. Accessed July 13, 2011.

[Schenker 2007]
Schenker, F. & Jacobs, R. *Project Management by Functional Capability.* Presented at the 7th Annual CMMI Technology Conference and User Group, Denver, CO, November 2007. www.dtic.mil/ndia/2007cmmi/Thursday/3amSchenker.pdf

[SEER-SEM 2011]
"SEER-SEM." http://en.wikipedia.org/wiki/SEER-SEM. Accessed July 13, 2011.

[Stutzke 2005]
Stutzke, R. *Estimating Software-Intensive Systems: Projects, Products, and Processes*. Addison-Wesley, 2005.

[Sulaiman 2006]
Sulaiman, T; Barton, B.; & Blackburn, T. "AgileEVM—Earned Value Management in Scrum Projects," 10-16. *AGILE '06 Proceedings of the Conference on AGILE 2006.* Minneapolis, MN, July 2006. IEEE Computer Society, 2006.

Appendix: COCOMO Factors List

One popular parametric cost-estimation tool is the COCOMO model. First published by Dr. Barry Boehm in his 1981 book, *Software Engineering Economics,* COCOMO (Constructive Cost Model) is an algorithmic-based parametric software cost-estimation model for estimating a software project as an "effort equation," which applies a value to tasks based on the scope of the project (ranging from a small, familiar system to a complex system that is new to the organization). COCOMO II is the successor of COCOMO 81, incorporating more contemporary software development processes, such as code reuse, use of off-the-shelf software components, and updated project databases [Boehm 1981].

At the heart of the COCOMO II model are the cost parameters themselves. These parameters are scale factors (5) and effort multipliers (17). Scale factors represent areas where economies of scale may apply. Effort multipliers represent the established cost drivers for software system development. They are used to adjust the nominal software development effort to reflect the reality of the current product being developed.

It would be reasonable to assert that an Agile development process would have an impact on some of these parameters. The following scale factors and effort multipliers, pulled from COCOMO II, might be impacted by the use of an Agile development process:

Development Flexibility (FLEX) Scale Factor

Definition:	The FLEX scale factor is related to the flexibility in conforming to stated requirements.
Rationale:	The participation of the user in the Agile development process, coupled with an iterative approach to building, should lower cost and schedule variance, because appropriate use of the methods assures continual communication as situations change. This permits appropriate reprioritization when needed.

Architecture/Risk Resolution (RESL) Scale Factor

Definition:	The RESL scale factor is related to early, proactive risk identification and elimination. The goal is to eliminate software risk by Preliminary Design Review (PDR). This factor is also related to the need for software architecture.
Rationale:	Although there is opportunity to tackle high-risk items early in the product lifecycle with an Agile approach, there is no guarantee that this will actually happen. The lack of clear guidance regarding how to accomplish a milestone review in an Agile development process, and the general lack of consensus in the Agile community on the need for or approach to developing a viable architecture, could increase the cost estimate.

Team Cohesion (TEAM) Scale Factor

Definition: The TEAM scale factor accounts for sources of project turbulence and entropy because of difficulties in synchronizing the project's stakeholders (e.g., users, customers, developers, maintainers, and interfacers). These difficulties may arise from differences in stakeholder objectives and cultures, difficulties in reconciling objectives, and stakeholders' lack of experience and familiarity with operating as a team.

Rationale: The Agile culture, in addition to the frequent interchanges between the user and the developers, should provide plenty of opportunity to improve team cohesion and should lower the cost estimate.

Analyst Capability (ACAP) Effort Multiplier

Definition: Analysts are personnel who work on requirements, high-level design, and detailed design. The major attributes that should be considered in this rating are analysis and design ability, efficiency and thoroughness, and the ability to communicate and cooperate.

Rationale: The participation of users in the development process should improve the knowledge of the analysts that elaborate the requirements and produce the software design. The impact of the improvement should lower the cost estimate.

Programmer Capability (PCAP) Effort Multiplier

Definition: Current trends continue to emphasize the importance of highly capable analysts. However, the increasing role of complex COTS packages, and the significant productivity leverage associated with programmers' ability to deal with these COTS packages, indicates a trend toward higher importance of programmer capability as well. Evaluation should be based on the capability of the programmers as a team rather than as individuals. Major factors that should be considered in the rating are ability, efficiency, and thoroughness, and the ability to communicate and cooperate.

Rationale: The participation of users in the development process should improve the knowledge of the programmers who write the software code. This is the factor that most relates to the Agile measure of velocity. The impact of the improvement should lower the cost estimate.

Application Experience (APEX) Effort Multiplier

Definition:	The cost-estimating multiplier based on the domain knowledge and capability of the software development staff is called APEX. The rating for this cost driver is dependent on the level of applications experience of the project team developing the software system or subsystem. The ratings are defined in terms of the project team's equivalent level of experience with this type of application.
Rationale:	The participation of users in the development process should improve the domain knowledge of the development team. The impact of the improvement should lower the cost estimate.

About the SEI

For more than three decades, the Software Engineering Institute (SEI) has been helping government and industry organizations acquire, develop, operate, and sustain software systems that are innovative, affordable, enduring, and trustworthy. We serve the nation as a federally funded research and development center (FFRDC) sponsored by the U.S. Department of Defense (DoD) and based at Carnegie Mellon University, a global research university annually rated among the best for its programs in computer science and engineering.

Contact Us

Software Engineering Institute
4500 Fifth Avenue, Pittsburgh, PA 15213-2612

Phone:	412.268.5800 | 888.201.4479
Web:	www.sei.cmu.edu | www.cert.org
Email:	info@sei.cmu.edu