How Should We Estimate Agile Software Development Projects and What Data Do We Need?

Glen Alleman and Thomas Coonce

Abstract

This paper proposes and demonstrates how to estimate agile software projects using collected data from prior similar projects. The authors discuss traditional ways of estimating software projects and show why these approaches are not feasible in estimating agile projects. They demonstrate estimating agile projects using actuals from historical features. They recommend a few additions to DoD's Software Resource Data Report (SRDR) as well as a Standard Feature Breakdown Structure (FBS) to permit using this approach.

Cost Estimating Methods

Estimating the cost of Department of Defense systems, the Office of Cost Assessment and Program Evaluation (CAPE) recommends four cost estimation methods:

1. Analogy
2. Parametric (Statistical)
3. Engineering
4. Actual Costs

Estimating techniques for an acquisition program progresses from analogies to actual cost method as the program matures and more information is known. The analogy method is most appropriate early in the program life cycle when the system is not yet fully defined. This assumes there are analogous systems available for comparative evaluation. As systems become more defined (such as when the program enters the Engineering and Manufacturing Development Phase (EMD), estimators are able to apply the parametric method. Estimating by engineering tends to begin in the latter stages of EMD and the Limited Rate Initial Production (LRIP) when the design is fixed and more detailed technical and cost data are available. Once the system is being produced or constructed the actual cost method can be more readily applied.

The analogy method is often used early in the program, when little is known about the specific system to be developed. The parametric technique is useful throughout the program, provided there is a database of sufficient size, quality, and homogeneity to develop valid cost estimating relationships. The engineering estimate is used later in program development and production, when the scope of work is well defined and an exhaustive Work Breakdown Structure can be developed. Finally, estimating by actual costs produces the lowest risk estimate due to the fact that the system cost is derived from a trend from the current contract to estimate.[1]

Software Cost Estimating Methods

Software development effort estimation predicts a realistic amount of effort (in person-hours) required to develop or maintain software based on incomplete uncertain and noisy input. In spite of the above CAPE guidance, published surveys on the software estimation practice suggest that expert estimation (engineering estimating or bottoms up) is the dominant strategy when estimating software development effort. Many times, effort estimates are over-optimistic. The mean effort overrun can be 30 percent and does not decrease over time. [2]

Software researchers and practitioners addressed the problems of effort estimation for software development projects starting the 1960's.[3] Most of the research has focused on constructing formal software effort estimation models that are parametric in nature and rely upon some form of sizing input and modified by other factors. Common parametric software estimating models include:

- COCOMO II by the University of Southern California
- SEER-SEM by Galorath, Inc.
- SLIM by Qualitative Software Management
- TruePlanning by PRICE Systems

Each of these models is based upon the actual results of historical projects characterized by the operating environment, type of platforms, and calibrated for unique factors like the amount of code re-use or other complexity factors unique to the application. The cost models used in sea, space, ground, and air platforms by the DoD services are generally based on the common effort formula shown below.[4]

$$Effort = A \times Size^{B} \times EM$$

Where:

- *Effort* in Person Months
- *A* is a constant derived from historical project data
- *Size* in source lines of code or other size measures
- *B* is a scale factor
- *EM* is an effort multiplier from cost factors.

The models use size expressed as lines of code, function points, object-oriented metrics, and other measures. Each model has respective cost factors for the linear effort multiplier term, and each model specifies the *B* scale factor in different ways (either directly or through other factors). Some models use project type or application domain to improve estimating accuracy.

For each commercial estimating model, the cost estimator needs to know the software Size and other unique attributes of the project. The Size input is computed but requires a design document, such as the Software Requirements Specifications (SRS). Function Points are created from the SRS and converted to Effective Software Lines of Code (ESLOC). Similarly, Use Cases or Objects are computed and used directly (by some of the commercial models) or converted into ESLOC to drive the parametric model. Other project attributes are obtained directly for the Cost Analysis Requirements Description or imputed based on analyst judgement or other expert opinion.

Software Development Methodologies and the Challenges to the Estimator

How does one estimate the software development effort when requirements are not fully known in the early stages? It can be argued that the development should not be approved if the customer does not know requirements. However, to address both of these concerns, one needs to understand the different types of software development methods (or models), why they exist, and what challenges they present to the estimator.

Waterfall Development Model

In software engineering, a software development model (SDM) splits the software development work into distinct phases (or stages) containing activities with the intent of better planning and management. This approach is often considered a subset of the systems development life cycle (SDLC) which emerged in the 1960's.[5] The idea of a SDLC is "to

pursue the development of information systems in a very deliberate, structured and methodical way, requiring each stage of the life cycle—from inception of the idea to delivery of the final system—to be carried out rigidly and sequentially" within the context of the framework being applied. The main target of this methodology framework was "to develop large scale functional business systems in an age of large scale business conglomerates. Information systems activities revolved around heavy data processing and number crunching routines".[6] This framework led to the “waterfall” development method which is a sequential process, development flows steadily downwards (like a waterfall) through several phases, typically:

- Requirements analysis resulting in a software requirements specification
- Software design
- Implementation
- Testing
- Integration, if there are multiple subsystems
- Deployment (or Installation)
- Maintenance

The waterfall principles are

- Project is divided into sequential phases, with some overlap between phases.
- Emphasis is on planning, time schedules, target dates, budgets and implementation of an entire system at one time.
- Tight control is maintained over the life of the project via extensive written documentation, formal reviews, and approval/signoff by the user and information technology management occurring at the end of most phases before beginning the next phase. Written documentation is an explicit deliverable of each phase.

Figure 1.0 depicts the waterfall concept and linear activities.[7]

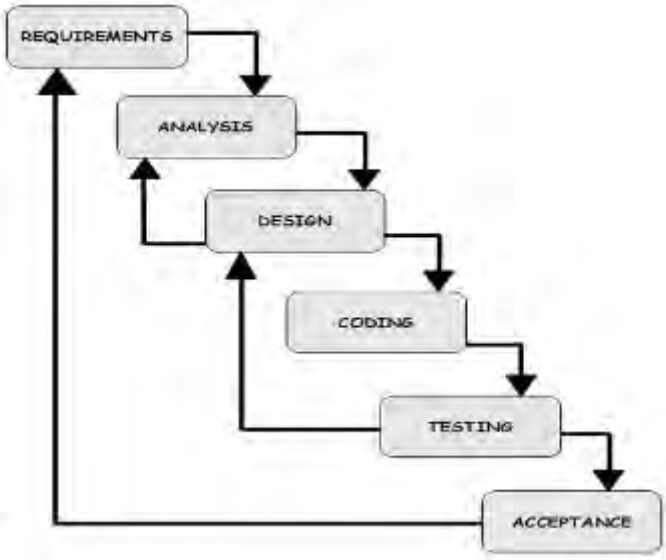

Figure 1.0 Waterfall Software Development Concept

In 1985, the United States Department of Defense captured this approach in DoD-STD-2167A, the standard for working with software development contractors, which stated that "the contractor shall implement a software development cycle that includes the following six phases: Preliminary Design, Detailed Design, Coding and Unit Testing, Integration, and Testing.[8] This methodology and associated processes is depicted in **Figure 2.0**.[9]

These overlapping phases of requirements analysis, architecture and preliminary design, detailed design, coding, and a series of tests beginning with software unit testing followed by integration and associated qualification testing were established in the 1970's.[10] Qualified Software Configuration Items (SCIs) are integrated into subsystems and finally into the complete system.

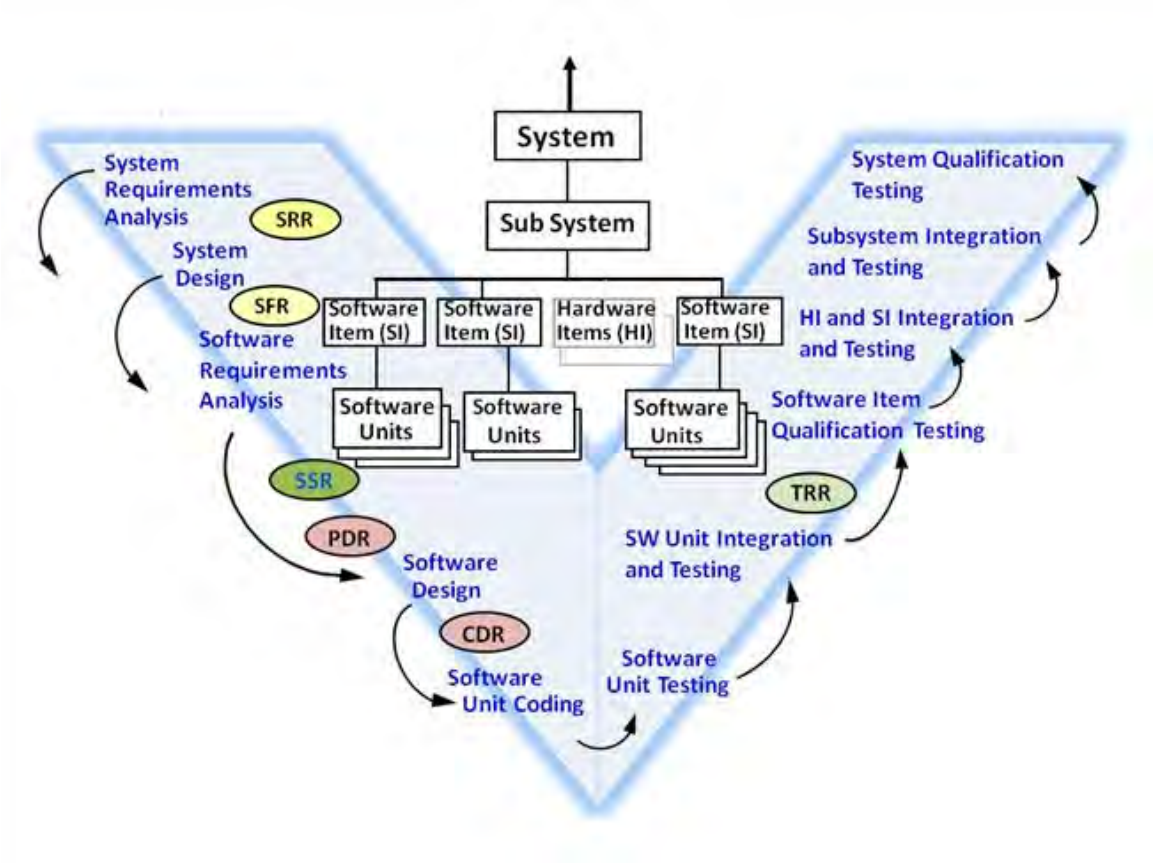

Figure 2.0 Traditional Waterfall Software Development Process and Activities

This traditional waterfall approach discourages revisiting and revising any prior phase once it is complete. This "inflexibility" has been a source of criticism by supporters of other more "flexible" models. It has been widely blamed for several large-scale government projects running over budget, over time, and sometimes failing to deliver on requirements due to the Big Design Up Front (BDUF) approach. The main criticism is this approach does not accommodate situations where the client does not know exactly what their requirements are before they see working software and so change their requirements, leading to redesign, redevelopment, and retesting, and increased costs.[11]

Designers may not be aware of future difficulties when designing a new software product or feature, in which case it is better to revise the design than continue with a design that does not account for newly discovered constraints, requirements, or problems. Except when contractually required, the waterfall model has been superseded by more flexible and versatile methodologies developed specifically for software development.[5] The DoD has

a stated preference against waterfall-type methodologies, starting with MIL-STD-498 (superseded by IEEE 12207).[12]

The weakness of the waterfall model led to the development of other software development approaches to reduce client uncertainty and risk and unfavorable cost and durations. Some of these approaches are

- Incremental development
- Evolutionary development
- Spiral development
- Rapid application development
- Agile development

Incremental Development Model

The incremental life cycle model was one of the first variations to be derived from the waterfall model. The assumption behind the model is that the requirements can be segmented into an incremental series of the products, each of which is developed somewhat independently. The series of releases is referred to as "increments", with each increment providing more functionality to the customers. After the first increment, a core product is delivered, which can already be used by the customer. Based on customer feedback, a plan is developed for the next increments, and modifications are made accordingly.

This process continues, with increments being delivered until the complete product is delivered. The incremental model has the following advantages: (1) Less cost and time is required to make the first delivery. (2) Less risk is incurred to develop the smaller systems represented by the increments. (3) User requirement changes may decrease because of the quicker time to first release. (4) Incremental funding is allowed; that is, only one or two increments might be funded when the program starts.[13]

From the estimators point of view, under this model (like the waterfall model), the client must define all requirements upfront and the estimator can use any number of the above parametric models to obtain reasonable estimates of the person months involved. But what if the client does not know all his specific requirements up front?

The Evolutionary Development Model

The next step in life cycle development is the evolutionary model, which explicitly extends the incremental model to the requirements phase. In this model, the customer defines the requirements for "Build 1" and after delivery, he uses his experience with the first build to define the requirements for the next build. This process continues until all the

requirements are satisfied. The advantage of this approach is that there is greater user involvement and feedback that in the Incremental or classic waterfall models.[14]

From the cost estimators point of view, these parametric models can be used to estimate "Build 1" based on the stated requirements (and derived sizing factors), but the estimator has little information for estimating "Build 2" due to changes that the client may make after reviewing "Build 1." This is a significant challenge for the estimating community. In many cases, the total project is incrementally funded and the cost estimator may obtain requirements for each build and thus able to produce credible estimates for each build.

Rapid Application Development Model

The Rapid Application Develop (RAD) software development methodology favors iterative development and the rapid construction of prototypes instead of large amounts of up-front planning. The "planning" of software developed using RAD is interleaved with writing the software itself. The lack of extensive pre-planning generally allows software to be written much faster, and makes it easier to change requirements. The rapid development process starts with preliminary data models and business process models using structured development techniques. In the next stage of RAD, requirements are verified using prototyping, refining the data and process models. These stages are repeated iteratively; further development results in "a combined business requirements and technical design statement to be used for constructing new systems.

The basic principles of Rapid Application Development are:

- Fast development and delivery of a high quality system at a relatively low investment cost.
- Attempts to reduce inherent project risk by breaking a project into smaller segments and providing more ease-of-change during the development process.
- Produce of high quality systems quickly, primarily via iterative Prototyping (at any stage of development), active user involvement, and computerized development tools. These tools may include Graphical User Interface (GUI) builders, Computer Aided Software Engineering (CASE) tools, Database Management Systems (DBMS), fourth-generation programming languages, code generators, and object-oriented techniques.
- Emphasis is on fulfilling the business need, while technological or engineering excellence is of lesser importance.
- Project controls for prioritizing development and defining delivery deadlines or "time-boxes". If the project starts to slip, emphasis is on reducing requirements to fit the time-box, not in increasing the deadline.

- Generally includes joint application design (JAD), where users are intensely involved in system design, via consensus building in either structured workshops, or electronically facilitated interaction.
- Active user involvement is imperative.
- Iteratively producing production software, as opposed to a throwaway prototype.
- Produces documentation necessary to facilitate future development and maintenance.
- Standard systems analysis and design methods can be fitted into this framework.[12]

It is difficult from the cost estimator's point of view to develop credible estimates for these efforts because there is no set of stable requirements to use as the sizing for the estimating model. A key attribute of this approach is the client and developer are permitted to reduce requirements to fit a time-box, while not increasing the deadline. Many cost estimators will estimate the effort associated with the prototype based on the stated requirements and then extent the personnel resources over the total projected development duration. If requirements are not known, the estimator is left to bounding the amount of effort by using the results of prior analogous efforts.

The Spiral Model

A current development approach is the Spiral model that combines some key aspects of the waterfall model and with rapid prototyping methodologies in a top-down and bottom-up concept. [12] The Spiral model is a risk-driven controlled prototyping approach that develops early deliverables to specifically address risk areas followed by assessment of prototyping results and further determination of risk areas to prototype. Prototyped areas frequently include user requirements and algorithm performance. Prototyping continues until high risk areas are resolved and mitigated to an acceptable level.

This model is appropriate for exploratory projects that are working in an unfamiliar domain or with unproven technical approaches. The iterative nature allows the knowledge gained during early stages to inform subsequent stages. During each iteration or loop, the system is explored at greater depth and more detail is added.[14]

The spiral model starts with an initial pass through a standard waterfall life cycle, using a subset of the total requirements to develop a prototype. The set of requirements is hierarchical in nature, with additional functionality building on the first efforts.[15] With the assumption that functionality can be dropped, the estimator can use the parametric estimating tools. Use of similar historical results also lends credibility to the estimate.

The Agile Development Model

The Agile software paradigm is a collection of software development methods where solutions evolve through collaboration between self-organizing, cross-functional teams. Agile promotes adaptive planning, evolutionary development, continuous delivery, continuous improvement, and rapid and flexible response to change.[12]

Development methods exist on a continuum from *adaptive* to *predictive.* Predictive methods, such as the classical waterfall model, focus is on analyzing and planning the future in detail and cater for known risks. In the extremes, a predictive team reports what features and tasks are planned for the entire length of the development process.

Predictive methods rely on effective early phase analysis and if this goes very wrong, the project may have difficulty changing direction. Agile methods lie on the *adaptive* side of this continuum where the focus is on adapting quickly to changing realities. When the needs of a project change, an adaptive team changes as well.[5]

While the use of the Agile model has been applied in the commercial environment where scope is often open ended, the DoD, and other federal agencies have been encouraging the use of the Agile model when the system being developed needs to rapidly adapt to emerging user needs.[16] Federal government and DoD have emphasized the necessity to shorten acquisition timelines to be responsive to increasing operating tempo and warfighter need with more rapid capability development. In 2009, the Defense Science Board wrote that "The fundamental problem DoD faces is that the deliberate process through which weapon systems and information technology are acquired does not match the speed at which new IT capabilities are being introduced in today's information age". [17]

Additionally, requirements for any given system are likely to evolve between the development of a system concept and the time at which the system is operationally deployed as new threats, vulnerabilities, technologies, and conditions emerge, and users adapt their understanding of their needs as system development progresses. With budgets constrained, ops tempos increasing, and requirements perpetually evolving, software development and acquisition practices must evolve in a way that facilitates faster capability deployment and flexibility in approaching system requirements.[18]

Iterative, incremental software development methodologies commonly referred to as "Agile" methods have been gaining ground in efforts throughout the DoD and federal agencies as a means to achieving these objectives for the acquisition of software-intensive systems and improving visibility into development execution to enable early detection of problems that can derail programs.[18]

Under Agile, an exhaustive set of requirements is not locked down at the start of the program. Rather, Agile development assumes that system and software requirements will evolve over time, rather than be definitized prior to system development. With Agile, high-level vision for the system is defined up front, but specific requirements are fixed at the iteration (or "Sprint") level according to an established cadence. The development team and user representative (generally referred to as a "Product Owner") agree to a set of requirements to accomplish during the defined time interval associated with the iteration. This serves to time-box the delivery of software: increments are completed on a regular, predictable basis. At the end of a sprint or increment, prioritized software requirements are agreed to for the next development iteration. The understanding of the user requirements evolves, guided by the high-level vision (or roadmap), as the software product continues to be developed. [Wrubel, p. 8]

While there are multiple ways for Agile project to be implemented, and there is disagreement in the terms used, the DoD's Performance Assessment and Root Cause Analyses (PARCA) office recently issued a *Program Manger's Guide on Agile* which clarified processes and terms.[19] The guide states that a program's "High Level Vision" should be communicated to the implementing contractor as "Capabilities" and these in turn and decomposed into features. The guide states that "capabilities" refers to a group of Features that are traceable to the technical and operational requirements of the product being delivered.[McGregor, p. 6] One way to view how the high level vision statements are decomposed into capabilities, features and user "stories" is depicted in **Figure 3.0**.

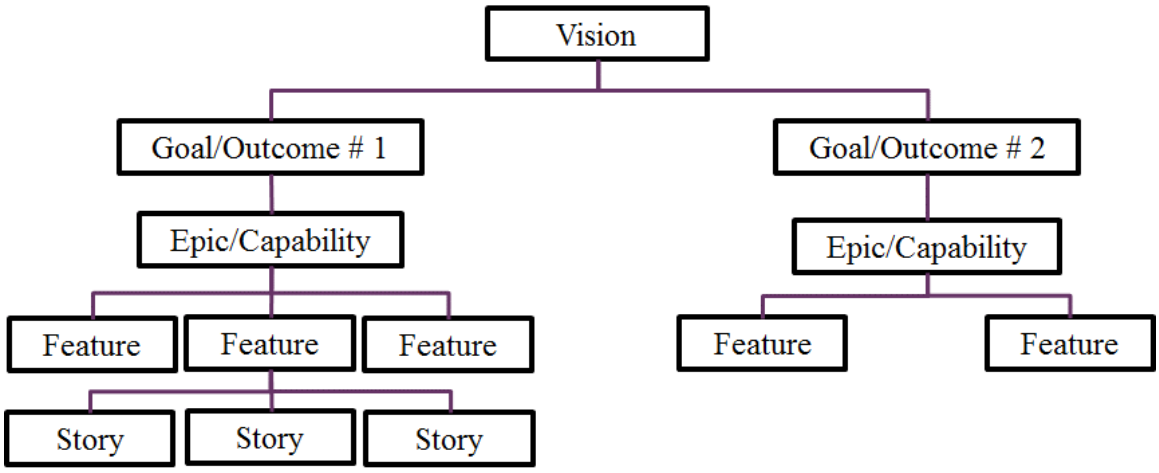

Figure 3.0. Decomposing Customer's Vision into Capabilities, Features and Stories

PARCA's PM Guidebook on Agile and Earned Value Management (EVM) provides useful term definitions and they are reproduced in Table 1.0 below. [McGregor, p. 14]

Capability	Term frequently used interchangeably with **Epic** to describe a high level system functionality defined by the government to meet a specific required need. All Capabilities should have clearly defined objective technical completion criteria. Capabilities are typically found at or above the Control Account level of the WBS and are usually composed of multiple **Features.**
Epic	See **Capability.**
Feature	Term used to describe a discrete system functionality defined by the government to help meet the specific completion criteria of a **Capability**. All Features should have clearly defined objective technical completion criteria. Features are typically found at the Work Package level of the WBS and can typically be completed in a single **Release**.
Story	Term used to describe activities that contribute to the completion of a **Feature** and can be completed within a single **Sprint**.
Release	Term used to describe a concrete time box or cadence used to complete **Features.** Release duration can vary, but is typically three to six months. Many practitioners use the Release cadence as their rolling wave planning period.
Sprint	Term frequently used interchangeably with **Iteration** to describe a concrete time box or cadence used to complete **Stories**. Sprint duration can vary, but is typically two to four weeks.
Iteration	See **Sprint.**

Table 1.0. PARCA's Agile Term Definitions

The Software Engineering Institute recommends that "…when a program pursues incremental delivery approaches, a clear high-level vision (e.g., a concept of operations, or CONOPS) is placed on contract—one that describes Agile concepts and principles and desired outcomes but does not specifically mandate Agile." [Wrubel, p. 27]

Figure 4.0 illustrates how customers' desired capabilities are integrated into the contractors' oversight of Agile efforts after the contract is awarded. The program office either contracts for the full set of Capabilities stated in the CONOPS, or incrementally funds (and baselines) each release of capabilities or epics.

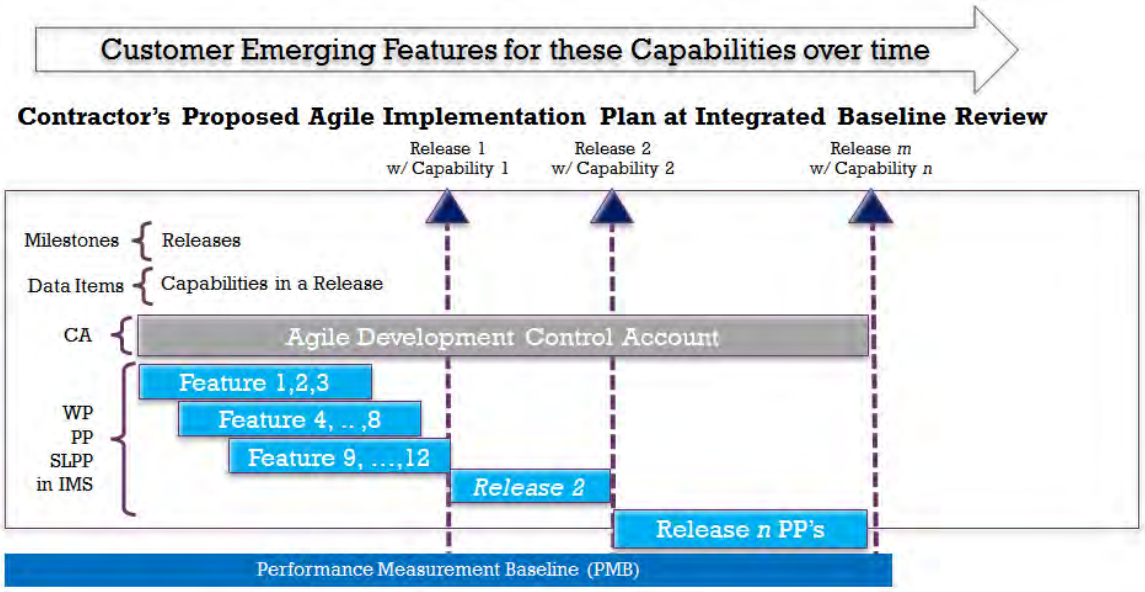

Figure 4.0 How Customer Capabilities Are Developed Using the Agile Model

A Proposed Way to Estimate Agile Software Development Efforts

So how does DoD prepare an estimate for a major software effort that only has a high level vision statement and desired capabilities and features? To answer this question, we first must understand what we are given and what drives the amount of effort and duration of an agile software project.

Before contract award, all we know is the government CONOPS and set of desired features, but we also know from the lessons learned from the data used to drive the parametric models, that effort and duration outcomes are highly dependent on the experience and "productivity" of the team which can vary significantly from one developer to another and is dependent on the extent of common software libraries and automated development tools used. Experienced developers in the application domain may have made extensive use of libraries and modern automated tools are far more productive than teams who are not experienced or make use of existing code and automated tools.

The authors propose that Agile software project can be estimated by using the historical experience of the features of historical analogies, in short, a nearest neighbor analogy method. Such historical development actuals would include feature descriptions and indications about the amount of common library use and extent of automated development tools. With the desired features described in the customer's CONOPS, the cost estimator could find the "nearest neighbor" within the historical actuals. If the cost estimator knew more about the developer's use of libraries and automated tools, the effort

and duration actuals can be selected that comes closest to matching the one in the historical database. If there is no information about the developer and their use of libraries and tools, the estimate of effort and duration can be calculated from historical means.

A notional Feature development example is shown in **Figure 5.0** for an Unmanned Air Vehicle to autonomously execute a mission received from its ground station.

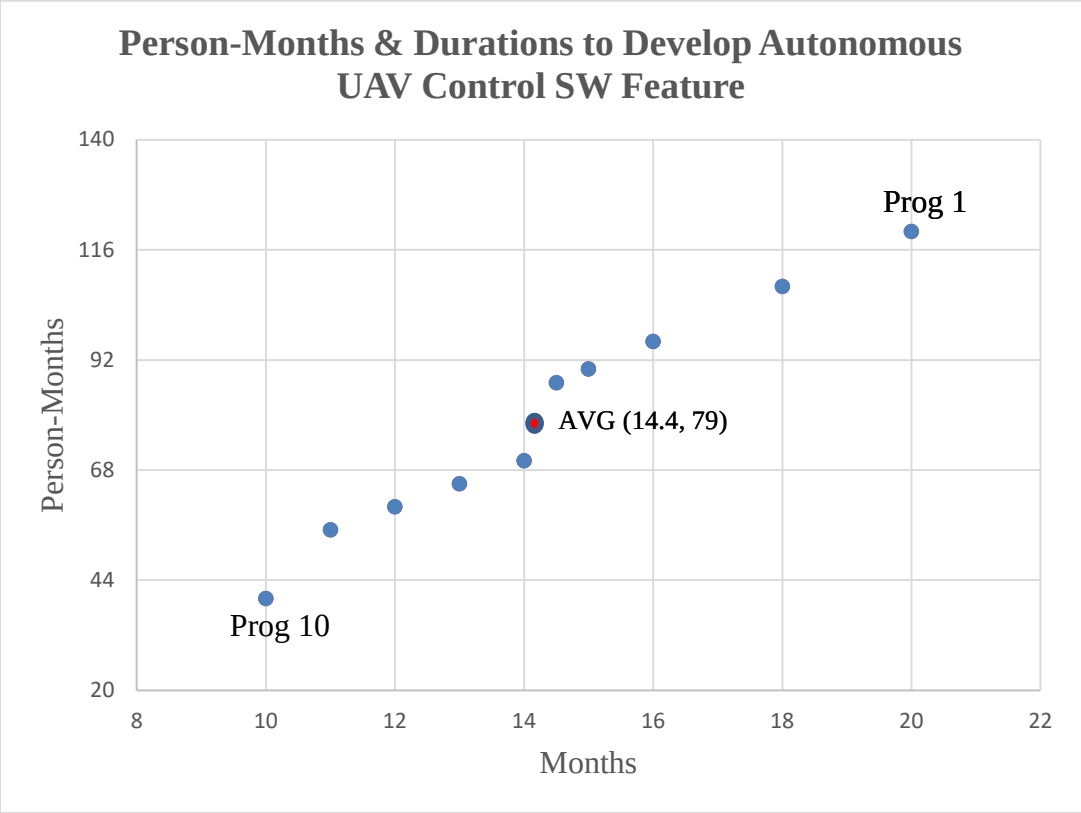

Figure 5.0. Notional Nearest Neighbor Estimating Example

Each data point reflects the historical outcome of similar development efforts for this Feature and each contains information about the extent of library and tool automation use. **Table 2.0** reflect these data point attributes. The cost estimator would select the historical outcome pair nearest to the specifics for the project (if known) or use the means (14.4 months and 79 person-months) if not known.

Prog ID	Feature	DevProc	Lib Use	Tool Use	Durr	PM
Prog 1	Auto Exc Pilot Mission	Agile	None	None	20	120
Prog 2	Auto Exc Pilot Mission	Agile	Min	Min	18	108
Prog 3	Auto Exc Pilot Mission	Agile	Min	Mod	16	96
Prog 4	Auto Exc Pilot Mission	Agile	Min	Heavy	15	90
Prog 5	Auto Exc Pilot Mission	Agile	Mod	Min	14.5	87
Prog 6	Auto Exc Pilot Mission	Agile	Mod	Mod	14	70
Prog 7	Auto Exc Pilot Mission	Agile	Mod	Heavy	13	65
Prog 8	Auto Exc Pilot Mission	Agile	Heavy	Min	12	60
Prog 9	Auto Exc Pilot Mission	Agile	Heavy	Mod	11	55
Prog 10	Auto Exc Pilot Mission	Agile	Heavy	Heavy	10	40

Table 2.0. Data Attributes of Notional Historical Feature Data in **Figure 5.0**

Data Needed to Estimate the Using the Nearest Neighbor Approach

Historical features, their attributes, and actual effort and durations are needed to use this approach. There are many hundreds of software features that have been (and are being) developed that have different outcomes depending on the operating platform and application domains. So, some form of grouping or standardization would be needed. Fortunately, the DoD's CAPE has recently modified it Software Resource Data Report (SRDR) to capture Agile actuals and attributes on future programs. The draft SRDR contains Agile planned and actual hours by the contractor's feature identifier. Most significantly, the draft SRDR requests the development contractor to categorize the effort by operating platform and primary application domain.[20] These applicable platforms and primary applications are shown in **Table 3.0**.

Operating Environments	Primary Application Domain
Air Vehicle-Manned	Command and Control
Air Vehicle-Unmanned	Communication
Missile Systems	Custom AIS Software
Ordnance Systems	Enterprise Information Systems
Other	Enterprise Service Systems
Sea Systems	Microcode and Firmware
Space Systems	Mission Planning
Surface Fixed	Other Real Time Embedded
Surface Mobile	Process Control
Surface Portable	Scientific and Simulation
Surface Vehicle	Signal Processing
	Software Tools
	System Sofware
	TMDE
	Training
	Vehicle Control
	Vehicle Payload

Table 3.0. CAPE SRDR Operating Environments and Primary Application Domains

Within the Primary Application Domain, the CAPE has promulgated about 200 subdomains examples that will go a long way to organizing a feature-rich historical repository. **Table 4.0** depicts the possible set of data fields that would be necessary to use the nearest neighbor analogy approach to estimate features contained within a UAV illustrated in **Figure 5.0** and **Table 2.0**. Note the attribute fields of Operating Environment, Application Doman/Sub-domain are in accordance with the SRDR requirements. Also note the field called Dev Process. This is also per the SRDR and valid fields include: Agile, Hybrid-Agile, Spiral, RAD, and Waterfall. This helps qualify the historical record.

The Library and Tool Use levels are suggested additions to the SRDR that seem importation productivity differentiators. Personnel experience level would also be useful, but was omitted from the table for clarity of the concept. Also note the inclusion of the attributes called Feature Description and Feature Category. The Feature Description appears to be missing from the draft SRDR and would need to be added. The Feature Category constitutes a Feature Breakdown Structure (FBS) and would need to be

developed by the estimating community over time similar to the standardized Work Breakdown Structure. This would ease the estimator effort of finding suitable analogies.

Program Name	Contract Number	Operating Environment	Application Domain/SubDomain	Dev Process	Library Use Level	Tool Use Level	Feature Category	Feature Description	Person Months	Duration
System Y	N000-C-2017-yyyy	Air Vehicle-Unmanned	VP-Pointing, Command, & Control Interface	Agile	Heavy	Moderate	Flight Control	Accept pilot missions via web client		
System Y	N000-C-2017-yyyy	Air Vehicle-Unmanned	VC-Flight Control	Agile	Heavy	Moderate	Flight Control	Transmit pilot mission to UAV		
System Y	N000-C-2017-yyyy	Air Vehicle-Unmanned	VC-Attitude Control System	Agile	Heavy	Moderate	GN&C	Record aircraft orientation, altitude and vector data		
System Y	N000-C-2017-yyyy	Air Vehicle-Unmanned	VC-Flight Control	Agile	Heavy	Moderate	Flight Control	Transmit aircraft orientation, altitude and heading data to GCS		
System Y	N000-C-2017-yyyy	Air Vehicle-Unmanned	VP-Pointing, Command, & Control Interface	Agile				Allow GCS pilot to input control commants		
System Y	N000-C-2017-yyyy	Air Vehicle-Unmanned	VC-Flight Control	Agile	Heavy	Moderate	Flight Control	Transmit real time pilot flight control commants to UAV		
System Y	N000-C-2017-yyyy	Air Vehicle-Unmanned	VC-Flight Control	Agile	Heavy	Moderate	Flight Control	Execute GSC pilot control commands real time		
System Y	N000-C-2017-yyyy	Air Vehicle-Unmanned	VC-Flight Control	Agile	Heavy	Moderate	Flight Control	Collect & store real time camera video		
System Y	N000-C-2017-yyyy	Air Vehicle-Unmanned	VC-Flight Control	Agile	Heavy	Moderate	Flight Control	Transmit real time video to GCS pilot		
System Y	N000-C-2017-yyyy	Air Vehicle-Unmanned	VC-Flight Control	Agile	Heavy	Moderate	Flight Control	Allow pilot to view UAV real time camera		
System Y	N000-C-2017-yyyy	Air Vehicle-Unmanned	VC-Flight Control	Agile	Heavy	Moderate	Flight Control	Accept Pilot GCS mission commands		
System Y	N000-C-2017-yyyy	Air Vehicle-Unmanned	VC-Flight Control	Agile	Heavy	Moderate	Flight Control	Autonomously execute pilot mission		
System Y	N000-C-2017-yyyy	Air Vehicle-Unmanned	VP-Reconnaissance Payload	Agile	Heavy	Moderate	ISR	Allow GCS personnel to request SAR images		
System Y	N000-C-2017-yyyy	Air Vehicle-Unmanned	VP-Reconnaissance Payload	Agile	Heavy	Moderate	ISR	Record & store SAR image data		
System Y	N000-C-2017-yyyy	Air Vehicle-Unmanned	VP-Reconnaissance Payload	Agile	Heavy	Moderate	ISR	Allow GCS personnel to request SAR images		
System Y	N000-C-2017-yyyy	Air Vehicle-Unmanned	VP-Reconnaissance Payload	Agile	Heavy	Moderate	ISR	Transmit GCS-requested SAR image data to GCS		
System Y	N000-C-2017-yyyy	Air Vehicle-Unmanned	VP-Reconnaissance Payload	Agile	Heavy	Moderate	ISR	Allow GCS Personnel to view SAR data		
System Y	N000-C-2017-yyyy	Air Vehicle-Unmanned	VP-Reconnaissance Payload	Agile	Heavy	Moderate	ISR	Record & store EOIR data		
System Y	N000-C-2017-yyyy	Air Vehicle-Unmanned	VP-Reconnaissance Payload	Agile	Heavy	Moderate	ISR	Transmit EOIR data to GCS		
System Y	N000-C-2017-yyyy	Air Vehicle-Unmanned	VP-Reconnaissance Payload	Agile	Heavy	Moderate	ISR	Allow GCS personnel to view EOIR data		
System Y	N000-C-2017-yyyy	Air Vehicle-Unmanned	VC-Executive	Agile	Heavy	Moderate	Status Monitoring	Monitor Status of on-board systesm		
System Y	N000-C-2017-yyyy	Air Vehicle-Unmanned	VC-Executive	Agile	Heavy	Moderate	Status Monitoring	Transmit Status of on-board systesm to GCS		
System Y	N000-C-2017-yyyy	Air Vehicle-Unmanned	VC-Executive	Agile	Heavy	Moderate	Status Monitoring	Log & store mission statistics		
System Y	N000-C-2017-yyyy	Air Vehicle-Unmanned	VC-Executive	Agile	Heavy	Moderate	Status Monitoring	Transmit mission statistics to GCS		

Table 4.0. Sample Attributes Needed in Feature Repository for UAVs

Conclusions and Recommendations

The traditional parametric estimating tools are not effective in helping a DoD analyst estimate the effort and duration of an evolutionary software project because detailed requirement are not available to generate software sizing inputs. The authors have shown that estimating a planned agile developed project can be done by using a nearest neighbor analogy technique built upon the actual development effort and durations and attributes of completed similar software features. To do this requires the following actions:

1. The DoD customers must produce a clear set of capabilities and desired features in the CONOPS.
2. Development contractors must submit the following data about each feature developed on CAPE's SRDR
    a. Brief Feature Description;
    b. An indication of the level of standard library and automated tool use;
    c. Categorization of standardized Operating Environment and Domain and Subdomains;
    d. The type of development method (Agile, Hybrid Agile, Spiral, RAD or Waterfall); and
    e. Actual development hours and duration
3. The CAPE must verify the submitted data and make it available to the authorized estimating community, and
4. The estimating community must develop a standardized Feature Breakdown Structure from the collected software feature repository.

It is the authors' understanding the all projects are required to develop a CONOPs that describe the end-state vision, capabilities and features so item 1 above is already being done. The CAPE's SRDR already contains most of the data elements needed and CAPE is currently vetting the submitted data and making the data available to authorized estimators. Only items 2 (a) and (b) would need to be added to the SRDR. Once the data is collected, the estimating community would surely rise to the occasion to create a standardized feature breakdown structure.

References

[1] https://dap.dau.mil/acquipedia/Pages/ArticleDetails.aspx?aid=8e8f5bf3-f517-4e98-8c07-80bb8670a830

[2] Jorgensen, Michel J., and Molokken, Kjetil. 2003. "A review of Studies on Expert Estimation of Software Development Effort". *International Symposium on Empirical Software Engineering*, 2003

[3] Nelson, E. A. 1966. "Management Handbook for the Estimation of Computer Programming Costs", AD-648750, Systems Development Corp

[4] Clark, Brad, and Madachy, Raymond, 2015. "Software Cost Estimation Metrics Manual for Defense Systems", Software Metrics, Inc.

[5] https://en.wikipedia.org/wiki/Software_development_process

[6] Elliott, Geoffrey, 2004. "Global Business Information Technology: an integrated systems approach". *Pearson Education* .87

[7] http://www.acqnotes.com/acqnote/careerfields/software-development-approaches

[8] DoD-STD-2167A, Defense System Software Development, 4 June 1985

[9] https://acc.dau.mil/CommunityBrowser.aspx?id=518290

[10] Royce, Winston. August 1970 "Managing the Development of Large Software Systems," *Proceedings IEEE WESCON*, 1-9.

[11] https://en.wikipedia.org/wiki/Waterfall_model

[12] Larman, Craig and Basili, Victor. 2003. "Iterative and Incremental Development: A Brief History". *IEEE Computer (June ed.).*

[13]http://myprojects.kostigoff.net/methodology/development_models/development_models.htm

[14] http://www.acqnotes.com/acqnote/careerfields/software-development-approaches

[15] http://www.testingexcellence.com/spiral-model-sdlc/

[16] Welby, Stephen, November 21, 2013. Deputy Assistant Secretary of Defense for Systems Engineering, "Thinking About Agile in DoD", *AFEI Agile for Government Summit*

[17]http://www.marcorsyscom.marines.mil/Portals/105/FRO/The%20Point%20Archive/The%20Point%20January%202013.pdf.pdf

[18] Wrubel, E., Gross, J., August 2015. "Contracting for Agile Software Development in the Department of Defense: An Introduction", *Software Engineering Institute, TECHNICAL NOTE CMU/SEI-2015-TN-006,*

[19] McGregor, John. 03 March 2016 . "Agile and Earned Value Management: A Program Manager's Desk Guide", *OUSD AT&L (PARCA)*

[20] ERP SRDR Form 3026-3 dated 2017January19