Agile Software Development Risk Assessment

2018 ICEAA Annual Conference

By John McCrillis

INTRODUCTION

Agile software development for Information Technology (IT) programs has challenged traditional cost estimating with its claim that cost and schedule are fixed and scope is variable. If true, then Agile programs are level of effort--it will produce whatever it can for the cost and schedule provided. But experience suggests few programs have been given such latitude. There just aren't many examples of people willing to fund projects that can't describe what they intend to deliver. But if a program can define its delivery then scope isn't variable as claimed and there is a risk of not getting what was defined for the cost and time agreed. Thus, Agile risk is no different any other software development[1] even though it claims fixed cost.

But messaging cost risk to a program that claims its fixed is not received well and is a distraction to discussing requirements and productivity. *Instead of capturing risk in a classic cost S-curve, a capability S-curve is more effective communicating risk to Agile programs*.

This paper investigates the Agile claim of fixed cost by discussing the two methods for estimating cost and the associated risk with each. It concludes that individual increments may be fixed cost but what is unknown is the number of increments it will take to deliver the intended capability. For assessing development risk, it is irrelevant the process used, be it Agile, Waterfall or any other method as long as a program capability is defined and captured in a metric.

CAPABILITY S-CURVE; A BETTER WAY OF ASSESSING RISK FOR AGILE PROGRAMS

Similar to a cost risk curve, a capability S-curve plots the confidence level for a capability as captured in a sizing metric[2]. A sizing metric like lines of code, function points, or other relates directly to cost; more capability equates to more cost. Agile programs that fix their cost to the point estimate will indeed have variable scope. The traditional cost S-curve can be inverted to a capability S-curve as shown in the notional confidence curve figure below.

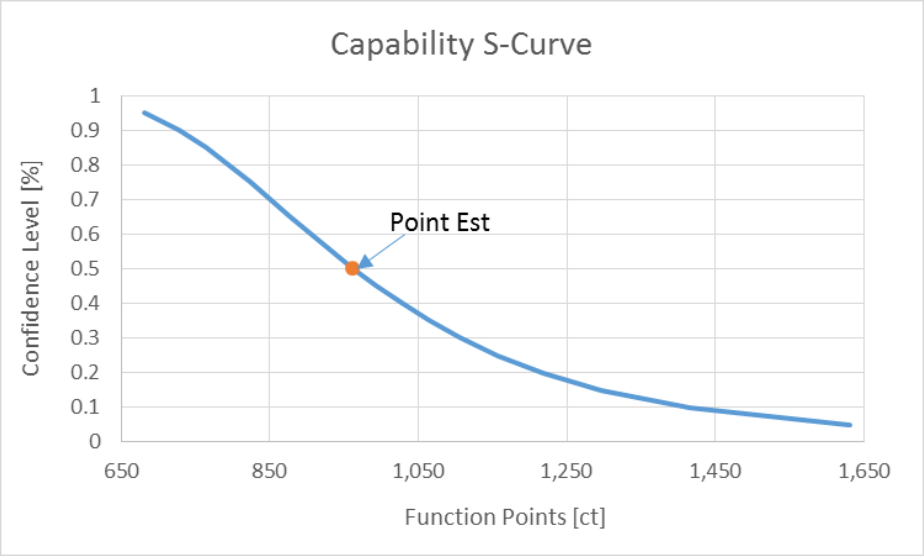


Figure 1 Capability S-Curve

Although it looks different than a cost S-curve, it has the same outcome – more capability equates to more cost. The same risk assessment in terms of cost is shown below. Converting cost to capability is tailoring the message to the user and has been more successful communicating risk than cost. Decision makers are receptive to this approach as well because it is aligned with the Agile objective of fixed cost and schedule.

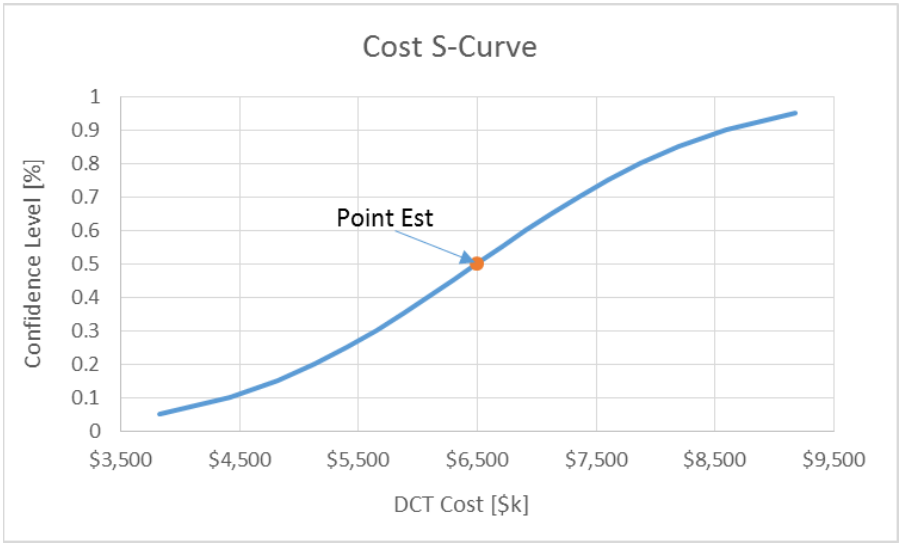


Figure 2 Traditional Cost S-Curve

BACKGROUND

Agile has adopted the "project management triangle" theory from the 1950s. Today, it is known as the "iron triangle" and consists of three legs; time, cost, and scope. The iron triangle, as implemented by Agile, can best be described as a theory that project management can only

control two of the three legs leaving the third leg to accommodate the unknown. Traditionally, programs fixed scope and varied time and cost. Agile contends that cost and schedule are fixed and scope varies as shown in the diagram below. Contractually, Agile programs are defining themselves as Time and Materials (T&M), yet few are. So what's going on?

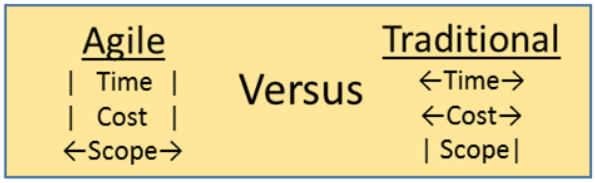


Figure 3 Agile software development "Iron Triangle" boundaries

Agile programs are producing requirements documents prior to, or shortly after kick-off that encompasses final delivery. They are producing Mission Needs Statement (MNS), Test and Evaluation Master Plan (TEMP), Concept of Operations (CONOPS), Operational Requirements Document (ORD) and other documents that specify what the final product "looks" like. Detailed planning is deferred to the development cycle but the end objective is fairly well defined in these various documents. The apparent inconstancy with the iron triangle is resolved by allowing for an undefined number of development cycles. In other words, *what's really variable is not program scope but the number of development cycles to get to a fixed scope.* This concept is shown in the figure below.

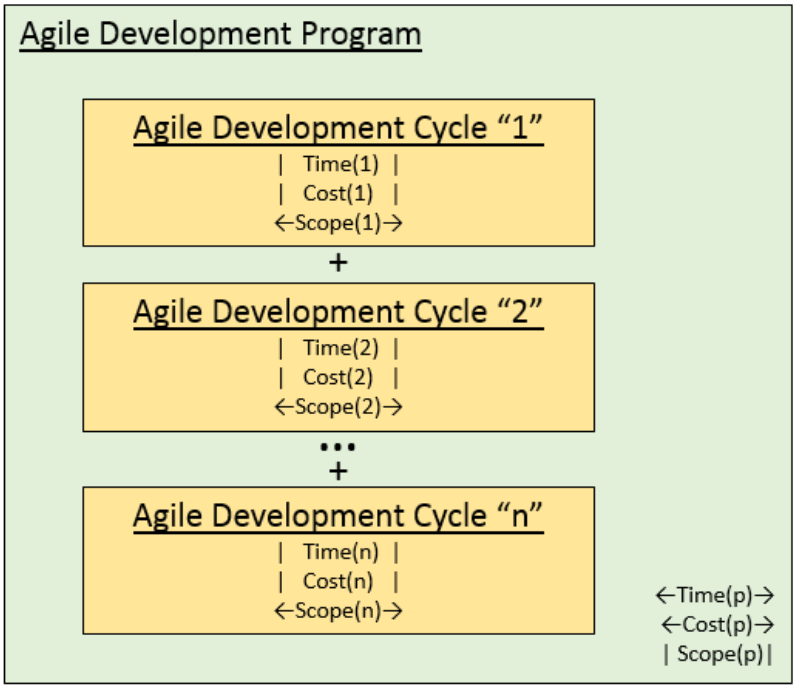


Figure 4 Reconciling scope to unknown development cycles

Agile has some great features; user involvement, early delivery of capability, and evolving solutions (how it's done, not what's done). But these features can be detractors as well. Successful programs produce the intended capability by controlling requirements, not allowing

them to float. How a program achieves the objective can certainly vary, but what it delivers should stay relatively constant. In traditional terms, the Material Design Solution (MDS) specifies the final solution for the desired function and this is what cost estimators are used to working with to produce a cost estimate.

Agile programs are not defining the MDS, but they are likely defining program functionally. For example, if the objective is to increase throughput of loan applications at a bank, one MDS might be hiring more people to process loans. Another MDS might be buying more printers while another could be developing software to automatically pull credit scores. Each MDS will have a unique cost and thus the desire for its specification. Agile programs likely don't know the MDS, but they mostly know the functionality. While improving throughput is a quality requirement, and the MDS is a technical requirement, identifying an applicant's credit score is a functional requirement. A cost estimate can be done using functional requirements as much as a MDS, however the variance will be higher to account for the variability in how the functionality is achieved. The same functionality can be achieved by multiple MDSs as depicted in the figure below that shows four paths to the same functional outcome. *Using a functional description instead of a MDS is a paradigm shift for many in the cost estimating community that needs to be made to accommodate Agile development*.

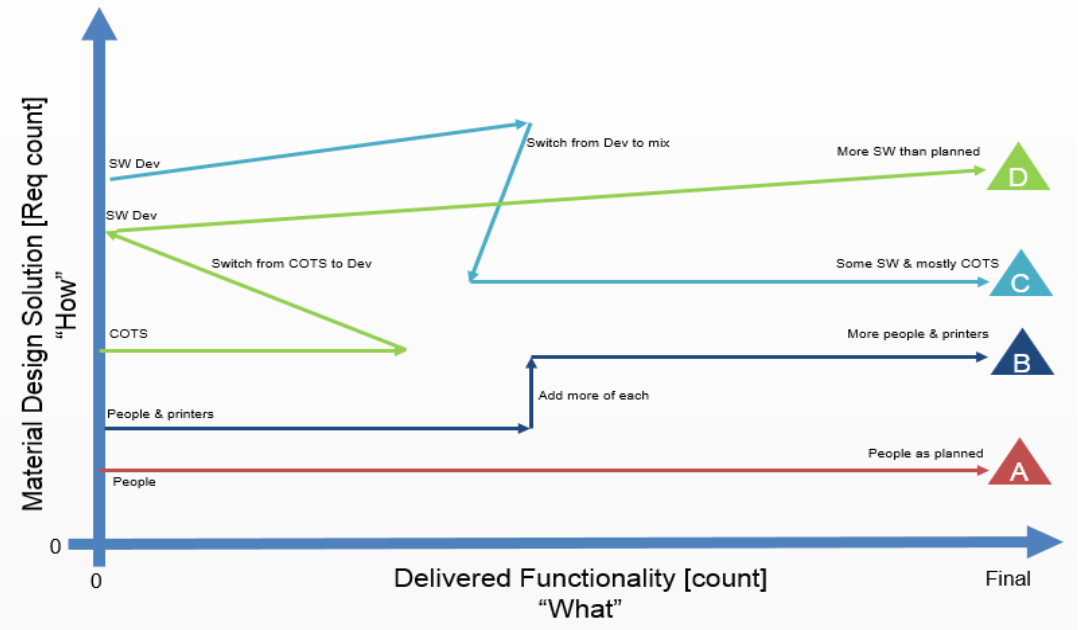


Figure 5 Functional Capability vs Material Design Solution

GOVERNING EQUATIONS

The cost estimating community uses one of two methods for costing software Design, Code, and Test (DCT) cost[3]: capability-based or staffing-based equations as shown below.

Capability-Based Equation	Staffing-Based Equation
DCT ($) = Size(unit) x Productivity ($/unit)	DCT ($) = Burn Rate ($/Time) x Duration (time)

Figure 6 Software Development Governing Equations

How the software is produced, waterfall, agile, or other is independent of the governing equations. Data may show differences in productivity by development method, but it doesn't change the governing equations. Which equation is used is dependent on the size and productivity being estimated up front. If size and productivity can be estimated, then the capability-based equation is used and the program has a defined completion. Risk is estimated in a standard Monte Carlo simulation. If scope is variable, then the staffing-based equation must be used and the program is T&M with little or no risk. *Whether Agile programs are estimated by capability or staffing is based on the availability of a sizing metric derived from the program's final functionality*.

WHAT SHOULD A SIZING METRIC LOOK LIKE?

A sizing metric should be independent of the implementation method, measurable by a third party from existing program documentation, and be consistent, repeatable, and verifiable. Additionally, it is desirable that size is auto-counted from delivered code[4] to establish an "actual" that could be used to verify the size estimation process and calculate actual productivity.

There are two types of numbers, ordinal (sequential; like small, medium, or large) and cardinal (measurable and repeatable). Ordinal numbers are problematic for measuring size because they can't be used to compare different programs nor should you make calculations with them like regressions. When used consistently in a program, they are reasonable predictors of future work, but offer no insight until an historical record has been established. This means ordinal numbers are less useful at program kick off because the metric has not been calibrated to the program's actual. This is the case for story points, feature points, releases, epics and many other metrics used in Agile.

There are many different sizing metrics[5] from which to choose and the table below summarizes some of the different criteria for selecting an appropriate metric. The reason for using one over another will be dependent on program specifics. *What's important is not so much the metric but that size is quantified; without this, capability can't be costed and risk cannot be assessed*.

Comparison of Software Metrics for Agile Development																	
	IFPUG FP	Mk II FP	Simple FP	FP Pattern Matching	NESMA FP	FiSMA FP	COSMIC FP	IFPUG SNAP FP	RICE objects	Goal Question FP	Lines of Code	Logical Statements	Backfiring	Fetaure Points	Story Points	Use-Case	Classes/Objects
Criteria	12	10	10	9	9	9	9	8	7	7	7	6	6	5	4	4	4
Derived from final software, auto count											x						x
Large body of published data	x			x						x	x						
ISO Standard	x	x			x	x	x										
Formal users groups	x						x	x		x				x	x		
Cost-effective and fast to apply			x	x					x		x	x	x		x		
Conversion rules to other metrics	x	x	x	x	x	x						x	x				
Unambiguous implentation method	x	x	x	x	x	x	x	x						x			
Derived from final software, manual count	x	x	x		x	x		x	x		x						x
Standardized counting method available	x	x	x		x	x	x	x			x	x	x				
Derived from requirments doc	x	x	x	x	x	x	x	x	x	x				x		x	
Support all sizes of applications	x	x	x	x			x		x	x	x	x	x	x	x	x	x
Support new & reused applications	x	x	x	x	x	x	x	x	x	x		x	x				
Supports all languages	x	x	x	x	x	x	x	x	x	x				x	x	x	
Supports all types of domain	x	x	x	x	x	x	x	x	x	x	x	x	x			x	x

Figure 7 Sizing Metrics Comparison

STORY POINTS

Story points[6] are time estimates for completing a feature. One story point represents a standard unit of effort for one development team, but only that team. Other teams will have different levels of effort associated with the same value. Story points are ordinal numbers and cannot be equated with capability. *Story points are really just a different form of the staffing-based equation and not representative capability*.

SOURCE LINES OF CODE (SLOC)

Source Lines of Code (SLOC) and its cousin Equivalent SLOC (ESLOC) have been used extensively as a sizing metric for many years. Although SLOC is a cardinal number (ESLOC is not), it can only be measured after it is delivered (ESLOC can never be measured). SLOC estimates at program kickoff come from Subject Matter Experts (SMEs) based on analogies. There is no reliable predictor of SLOC based on design documents. This is partly true because *SLOC is not deterministic; it is dependent on how a feature is coded*. One software feature coded by two developers will likely produce two different SLOC counts even when using the same language. Change the language and the difference will be even greater. But SLOC was used extensively because it is perceived as unambiguous; just like counting the lines of text in a book. But there are many issues with using SLOC or ESLOC as a sizing metric; paramount is an inability to accurately predict final SLOC at the beginning of the program.

FUNCTION POINTS (FP)

Function Points (FP) were first developed by engineers at IBM back in the 1970s. They use standardized rules to count transactions within elementary processes from a user's perspective. It is a count of inputs, outputs, outputs that need calculation, data internal and data external as shown in the figure below. The count is adjusted for complexity based on the number of each type of transactions, making this an ordinal number. The expected accuracy of a FP count is on the order of 5-10% even though it is ordinal. A function point count can be made at any time during the development cycle. It can also be used to "measure" a programs size using design documents at program kickoff and is independent how the feature is implemented be it waterfall, Agile or other. And FPs are independent of coding language; the FP count stays the same for Java, FORTRAN, C or any other language. Productivity may vary for different languages, but the sizing metric stays the same. This makes FPs an excellent choice for sizing the effort even with its limitation as an ordinal number.

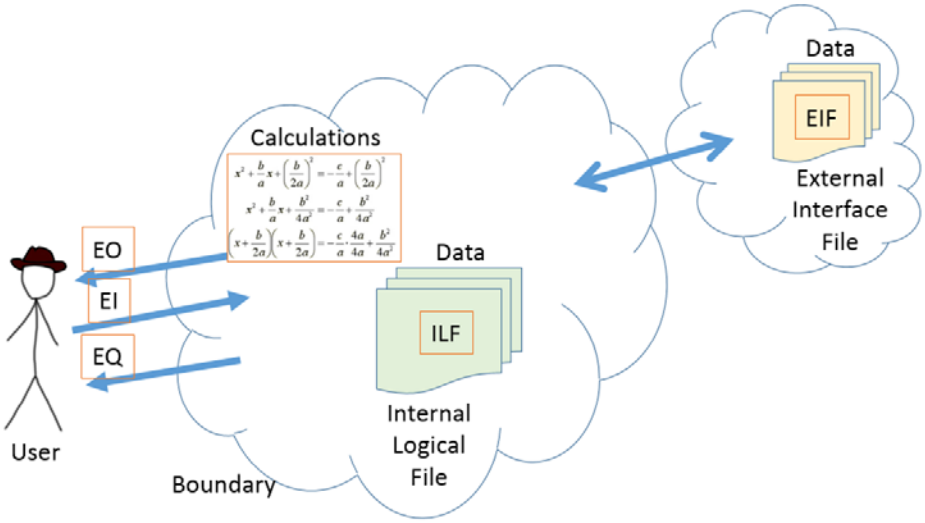
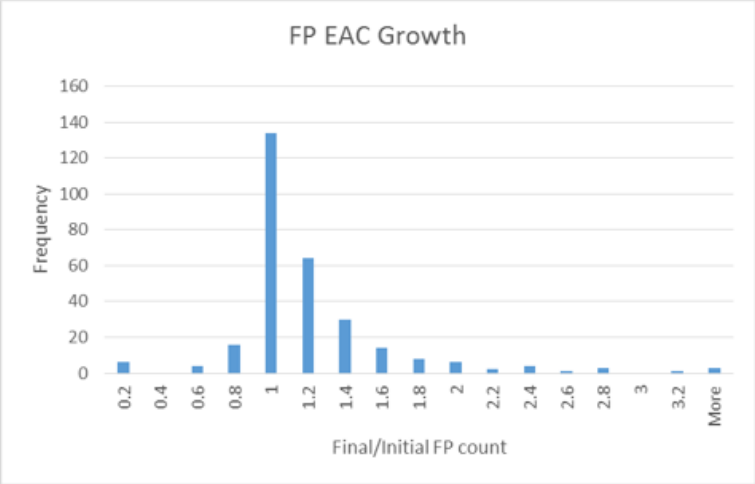


Figure 8 Function Point Transactions

FP counts can be performed by qualified FP counters independent the program at virtually any time in the development life cycle[7]. Separating the count from program management oversight removes the risk of a bias towards some program goal that has not been brought to light. But this doesn't mean the program shouldn't participate in the count, quit the contrary. The programs buy-in to the FP count is critical to the success of the program since it is the basis of the cost estimate. *It is recommended that FP counts be conducted by independent third parties throughout the life of the program.*

FUNCTION POINT ACCURACY

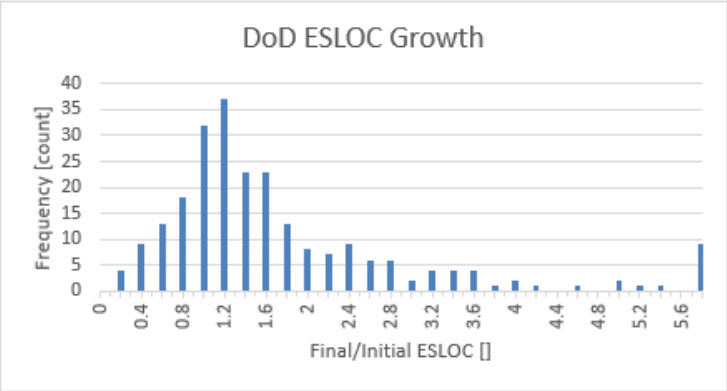
To test the premise of using FPs for sizing programs, FP counts from completed programs were compared to the initial estimates. Assuming the requirements stay constant, the accuracy of the estimate can be measured by dividing the final by the initial. This was done using commercially available FP data from the International Software Benchmarking Standards Group (ISBSG) as shown in the figure below. The growth is modest, something less than 14%, suggesting FPs are a viable sizing metric. However, the ISBSG data set is biased towards small programs which are easier to size than large complex programs. Whether the accuracy holds true for larger programs has not been established. Accepting the data as representative sizing growth, the histogram and variance can readily be implemented in a Monte Carlo risk assessment using Probability Density Function (PDF).



	Final/ Initial FP
Mean	1.14
Standard Error	0.03
Median	1.00
Mode	1.00
Standard Deviation	0.48
Sample Variance	0.23
CV	42%
Skewness	3.05
Range	4.89
Minimum	0.01
Maximum	4.90
Count	296.00

Figure 9 Function Point Growth

For comparison, ESLOC size estimates are notoriously inaccurate as shown in DoD Software Resources Data Report (SRDR) data below. This implies FPs are more precise at estimating size than SLOC estimates. Interestingly, there are significant number of programs that delivered less ESLOC than initially planned unlike the FP data. This variance is likely a manifestation of relying on SME opinion to identify analogies and the subsequent padding that comes with these estimates.



DoD ESLOC Growth	
Mean	1.85
Standard Error	0.14
Median	1.30
Mode	1.00
Standard Deviation	2.17
Sample Variance	4.73
CV	1.18
Skewness	4.60
Range	18.71
Minimum	0.10
Maximum	18.82
Count	240

Figure 10 ESLOC Growth

WHAT DOCUMENTATION CAN BE USED FOR FUNCTION POINT SIZING?

There are three fundamental requirement types; functional, technical, and quality. *Only functional requirements can be counted with FP methods*. In general, design documents need to be specific and measurable. Quality requirements like "improve" or "faster" are not useful. The fundamental concept that needs to be captured in the documentation used for function point counting is the identification of Elementary Processes (EPs)[8][9]. An EP is an end user task that after completion, a "save" is executed or the user moves on to another EP. Some EPs are simply requesting a display of data while others might be updating existing data. An EP may also be identified as a "feature" or "what" the application is intended to do. In terms of software coding, an EP is something that needs to be tested for acceptance.

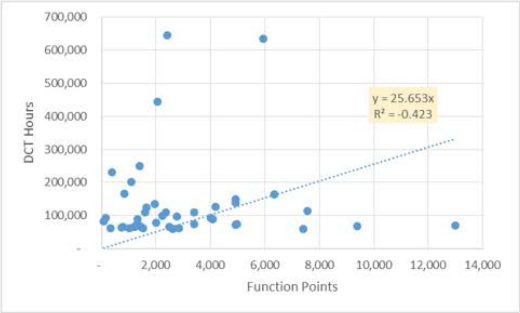
In practical terms, an EP is identified with a verb-object pair like in the following example; "the analyst uploads the arrest record into the central database." In this situation, the action is "upload" and the object is "arrest record". Assuming this feature is being done in software, this would count as an EP. There are certainly details below this task, like adding meta data and associating it with a court case, but there won't be an EP within it. Any program documentation that provides user scenarios tends to be written with verb-object phrases that can be used to identify EPs. Within the DoD framework this is the scenario section of the CONOPS[10][11]. There are other details necessary for conducting a FP count which are left to the standard selected. But the conclusion is the same, *whether Agile or other development method, size can be measured from functional descriptions.* The remaining challenge before generating a capability S-curve is identifying productivity.

PRODUCTIVITY CONSIDERATIONS

The data used to estimate productivity will depend on when the estimate is being developed within the program's development life cycle. There are three general classes of data; program, organization, and commercial[12]. Program data is collected throughout the life of the

program. Typically, program productivity starts out higher than it ends because of integration, which mostly occur at the end and takes effort without adding code. Because of this, in-process program productivity is an unreliable predictor of final productivity and it is better to use organization data. Program data becomes organization data at completion of the program. As an organization matures, its collection of actual productivity becomes the best source for estimating what is expected of a new program. Until an organization has its own data to establish initial estimates, there are commercial data sets available to identify benchmarks.

As an example, the ISBSG data set previously discussed was analyzed for programs with more than 30 FTEs. The scatter plot and histogram are shown below:



Programs great than 30 FTEs	
Mean	112.1
Standard Error	29.7
Median	46.0
Mode	#N/A
Standard Deviation	196.8
CV	176%
Kurtosis	15.3
Skewness	3.7
Range	1,092.0
Minimum	5.4
Maximum	1,097.4
Count	44

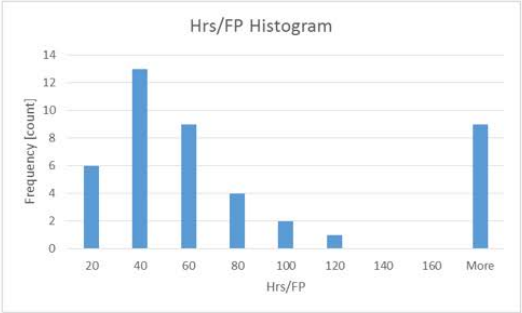
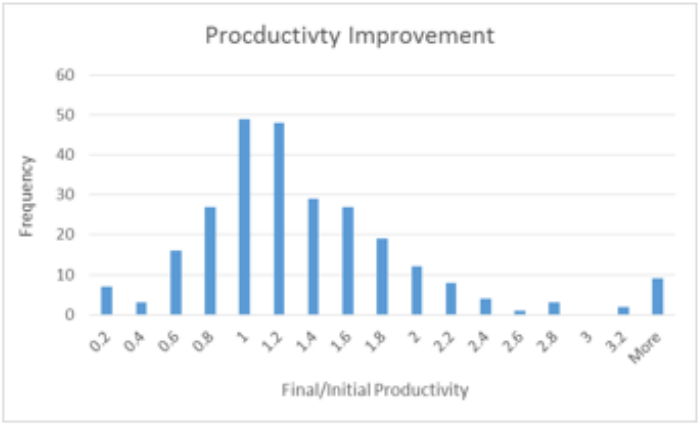


Figure 11 ISBSG Productivity Data for Programs with More than 30 FTEs

The hours represent only Design, Code, and Test (DCT). There is a significant variance with 9 programs having productivity greater than 140 hours/FP skewing the mean right and making it unrepresentative. The scatter plot regression should not be used either, as evidenced by the negative R^2. As a result, the median at 46 hours/FP is more representative the data than the average or regression. This is significantly lower productivity than all the ISBSG data which is on the order of 10 hours/FP. Intuitively, this makes sense because bigger programs need more coordination between more teams and this reduces overall productivity.

No relationships were identified between language, measurement methods, size, or other Meta data and productivity including development method. The latter is an interesting result -- *productivity is insensitive to the development process*. This supports the premise the FPs can be used to size Agile or any other development method used by a program.

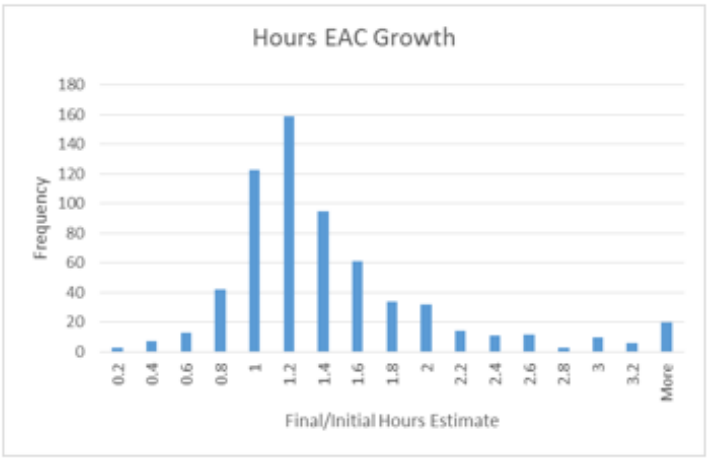
Like size, productivity was assessed for growth, which can be used in the Monte Carlo risk assessment for a productivity PDF. The results are shown in the figure below. As expected, the initial productivity estimate is higher than the actual with a 11% median. This is acceptable growth for most initial estimates.



	Final/ Initial productivity
Mean	1.34
Standard Error	0.07
Median	1.11
Mode	1.00
Standard Deviation	1.09
Sample Variance	1.18
CV	81%
Skewness	4.91
Range	10.90
Minimum	0.04
Maximum	10.95
Count	264

Figure 12 FP Productivity Growth

As a cross-check to productivity growth, hours growth which is the result of size and productivity growth was analyzed as shown in the figure below. The median, at 16% is comparable the product of the size and productivity growth.



Final/ Initial hrs	*Alldata*	Agile only
Mean	1.50	1.16
Standard Error	0.06	0.16
Median	1.16	1.06
Mode	1.00	#N/A
Standard Deviation	1.59	0.60
Sample Variance	2.54	0.35
CV	107%	51%
Skewness	7.60	2.16
Range	18.97	2.39
Minimum	0.03	0.52
Maximum	19.00	2.91
Count	645	14

Figure 13 Hours Growth Cross Check

Agile hours growth was assessed individually as shown below and found to be comparable to the larger data set for all development types shown above. With only 14 data points, it is still too early to draw conclusions, but it would appear that Agile is not going to be significantly different in terms of hours growth than waterfall programs.

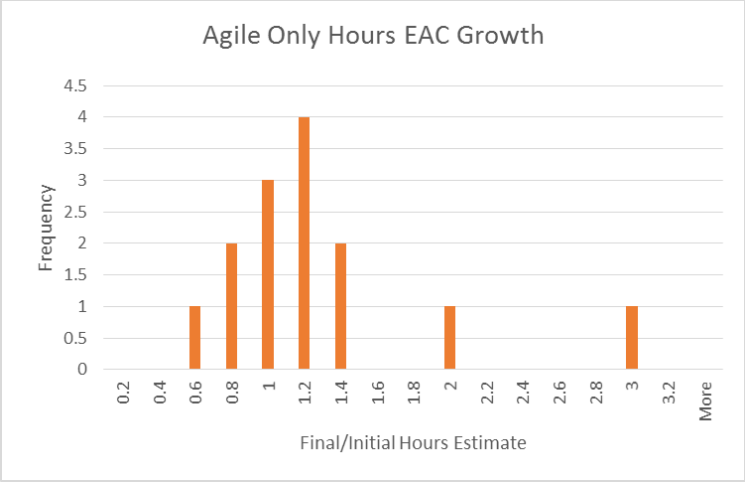


Figure 14 Agile Only Hours Growth

AGILE ACQUISTION STRATEGY

The Agile acquisition strategy begins with program planning and specification for the intended functionality. The functionality is used to create an Estimate at Completion (EAC) for cost and capability as previously discussed. The total cost and functionality are "chunked" into Minimally Viable Products (MVP) per increment of fixed cost and schedule. Each increment has its own detailed planning and oversight before it is released for development. Since the increment is a viable product by itself, the oversight process can cancel the program at any time or even incrementally fund the program. Cost for the increment is "fixed" and any content that can't be produced is pushed to the next increment. Thus the increment scope is variable and the time and schedule fixed like in the "Iron Triangle" concept. After development of an increment is complete, it is deployed and actual cost and final requirements are used to update the program plan as shown in the figure below. The number of increments it takes to deliver the program capability is dependent the actual productivity achieved.

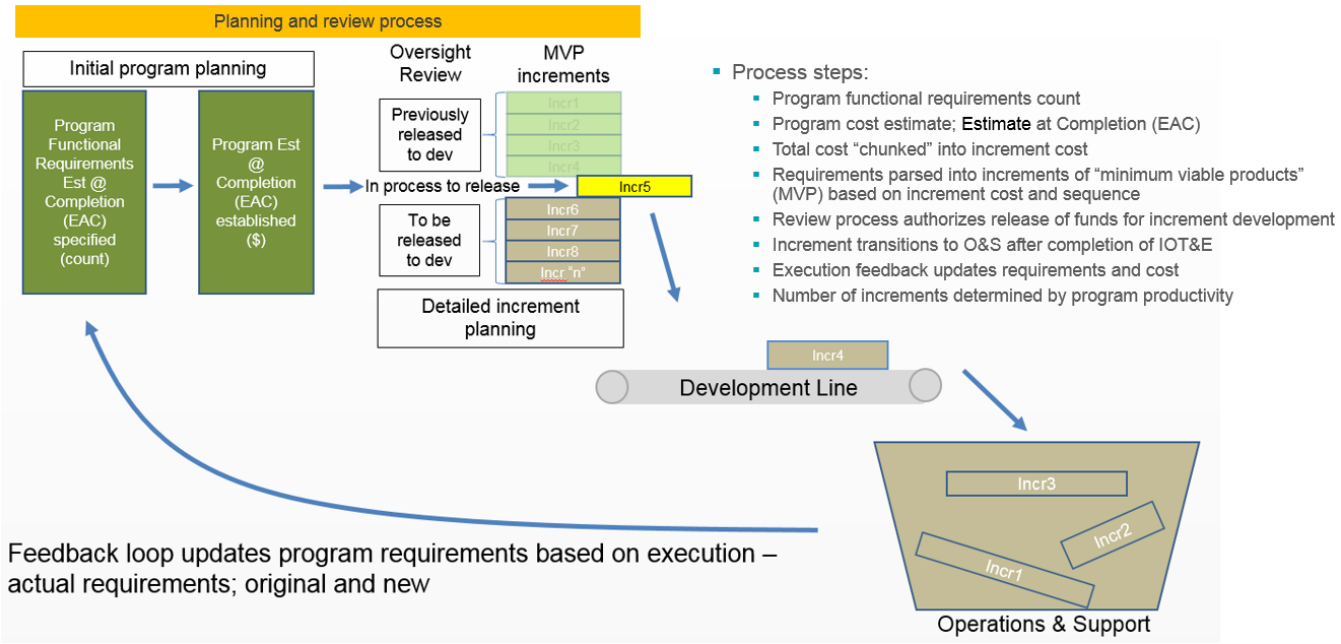


Figure 15 Agile Acquisition Process

Adopting an Agile acquisition strategy like the one described facilitates using a progress chart like the one below. Planned versus actual cost and capability are compared and earned value analysis techniques applied to predict completion. The target EAC for cost and capability are revised based on increment feedback. The progress chart is one of the few tools available to assess progress towards completion[13]. It also documents requirements change over time[14] which is useful for explaining cost growth should a program have this issue.

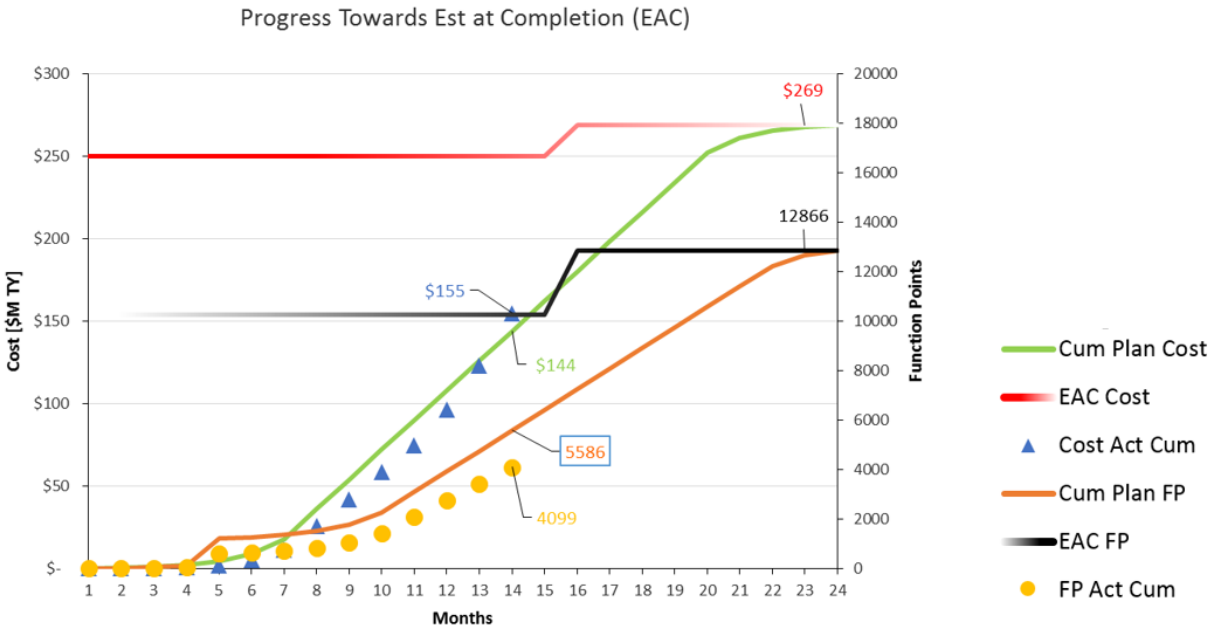


Figure 16 Program Progress Chart

PUTTING IT ALL TOGETHER AND THE CONE OF UNCERTAINTY

With size and productivity identified, it's possible to estimate the program cost using traditional methods. The Prime Mission Product (PMP) is the DCT estimated using the capability governing equation. Size is derived from functional descriptions and productivity organizational or program data. Other cost like integration, independent testing, systems engineering, program management, and training development may be estimated as a factor of DCT cost in many instances. Other cost like hardware and software purchases are added to generate a total program estimate. Risk is specified with a PDF for each element as appropriate.

As a program matures, the variance on both scope and productivity narrows, as notionally shown in the cone of uncertainty figure below. Similarly, requirements and their corresponding documents also become more refined as the product is better understood. Sometime during the "approved product" phase or "requirement specification" phase a reasonable first estimate can be produced because the functional description is available. During the "initial concept" phase a Rough Order of Magnitude (ROM) is the best that can be expected and will likely be based on analogies.

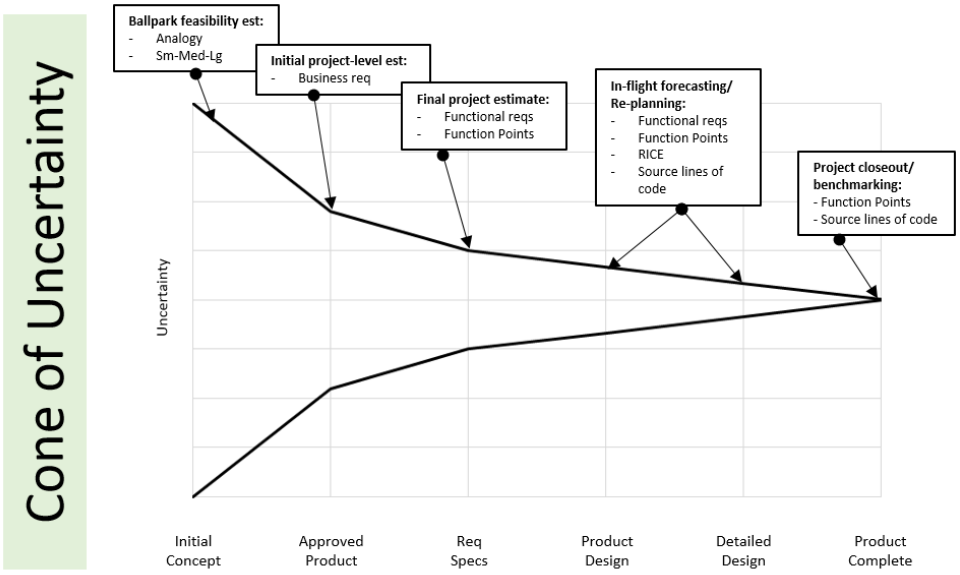


Figure 17 Cone of Uncertainty

WHERE'S THE RISK IN AGILE SOFTWARE DEVELOPMENT?

Assuming program scope has been specified in a sizing metric, the cost risk for Agile software development is the same as any other program; *not delivering what was intended for the agreed upon cost and schedule*. Risk makes no distinction about the development method. Using the capability equation, there is variance in both the sizing and productivity variables as previously shown and captured in PDFs.

There are only two variables to assess for PMP risk: size and productivity. There are two primary reasons why the size may not be known initially: poor requirements definition and poor sizing metric measurement. There are numerous sources of error for productivity: unskilled labor, inefficient use of labor, poor requirements definition, poor communication, churn, personnel turnover, delayed start, slow spool-up, new language learning curve effect, and a host of other reasons. Depending on the details, PDFs can be used to model the error for any one or all of these sources.

HOW TO CONVEY COST RISK TO AN AGILE PROGRAM WITH FIXED COST

For cost estimators working with Agile programs, conveying the issues surrounding cost risk can be challenging. The Agile principle that scope is variable is an opportunity to convey cost risk in terms of capability. To accomplish this, the cost s-curve, which was calculated from equation 1 below, can be readily reformatted from cost to scope using equations 2 and 3.

$$Eq1)\ Cost\ f(CDF) = \sum_{i=0}^{n} WBS_i\ f\left(var_1, f(PDF_1), \dots\right)$$

$$Eq2)\ Productivity\ f(CDF) = \frac{Cost\ f(CDF)}{Size}$$

$$Eq3)\ Size\ f(CDF) = \frac{Cost_{EAC}}{Productivty\ f(CDF)}$$

Where: PDF is the probability density function

CDF is Cumulative Distribution Function

$WBS_i\ f\left(var_1, f(PDF_1), \dots\right)$ is the functionality of WBS element *i* in terms of variables their probability density functions

$Cost_{EAC}$ is the Cost Estimate At Completion

Figure 18 Equations for Converting Cost CDF to Size

Equation 1 summarizes the detailed cost estimate for the program using a Monte Carlo simulation. Variables with known risk are modeled using PDFs to generate a cost Cumulative Distribution Function (CDF). Using the capability governing equation, the cost CDF is converted to a productivity CDF using the fixed size estimate as shown in equation 2. Similarly, a size CDF is back calculated from the productivity CDF using the fixed cost EAC of equation 3. Applying the programs cost EAC to productivity as a function of CDF reveals size as a function of CDF. As shown below, when the resulting s-curve is flipped, the probability decreases as the number of FP increases. This plot visualizes what is intuitive-- that getting more capability for a

fixed cost is less likely than getting less capability for that same cost. Or said differently, you are more likely to get less capability, not more for a fixed cost.

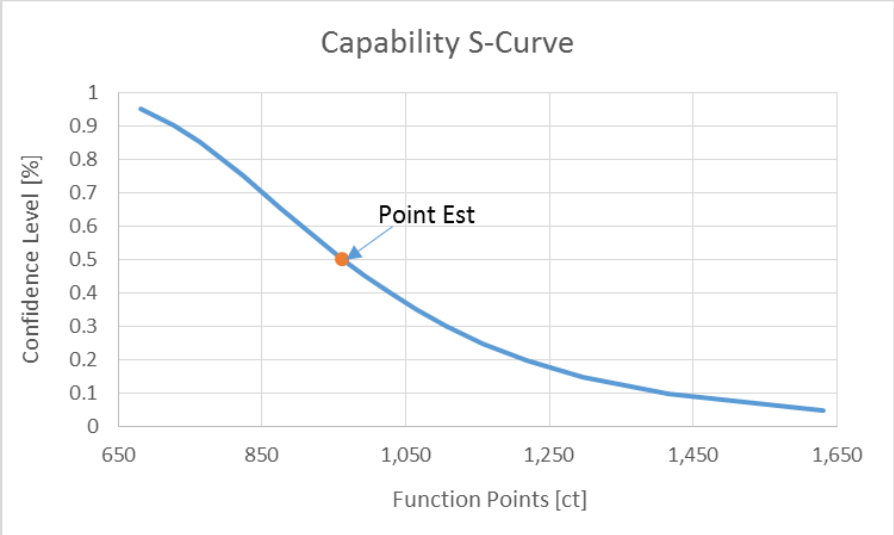


Figure 19 Risk Distribution on Scope

Like a traditional S-curve, a scope-based S-curve, can be an invaluable tool to facilitate discussion and informed decision-making. Its primary purpose is to focus management attention on requirements growth and productivity variance. Avoiding an Agile programs dismissal of cost growth, estimators and program managers can now have data-driven discussions on a wide variety of risk topics, including:

- How much risk is the program willing to assume?
- What are the implications of delivering less than the desired full scope?
- What mitigation strategies can be used to reduce the uncertainty in the underlying variables?
- What cost elements are most contributing to the overall program risk?

Perhaps most importantly, a scope-based S-curve conveys the concept that agile programs carry just as much risk as traditional software development. The way in which this risk manifests itself may differ, as does the terminology but the underlying risk remains. This understanding, and the data surrounding the analysis, is critical to giving program managers the best opportunity to manage and mitigate risk.

CONCLUSIONS AND RECOMMENDATIONS

There are only two governing equations to choose from when estimating software development cost: capability-based or staffing-based. The development method is independent the governing equations. Selecting which equation to use depends on whether the size of the scope is known. To estimate capability, the scope must be described sufficiently such that a sizing metric, like Function Points (FP), can be measured. Whether it's Agile, waterfall or any

other method, size is required for a capability-based estimates. With a sizing estimate, the Agile development cost and risk assessment are no different any other software development method.

Historically, cost estimators required programs to provide the sizing metric based on a MDS. It was believed that only the program could accurately define size, and that the software size wasn't deterministic from requirements. Function Point counting breaks this paradigm and has been shown to be applicable Agile programs as well as any other development method. Function Points have been shown to be more accurate for IT programs than SLOC even though it is based on functional requirements and not the MDS. It is recommended that function point counts be used to estimate the DCT size and that those counts be conducted by a third party to remove any potential for a bias. Having a sizing estimate, a capability cost estimate can be performed and the risk assessed with a traditional s-curve. Without a size estimate, cost can only be estimated using the less preferred time and material (T&M) methodology, which has little or no risk.

The true risk for an Agile program is similar any other program--not achieving the anticipated capability for the agreed upon time and cost. Agile's claim that cost is fixed is only true for a given iteration; the risk is in the number of iterations it will take to achieve the intended capability. Using the traditional S-curve for cost, the data can be reformatted into Agile terminology by plotting the confidence of achieving the intended capability. Presenting risk in terms of capability breaks the communication barrier with Agile programs into terms a program is more likely to relate.

GLOSSARY OF TERMS

Agile Method	A software development process that evolves a product in a series of rapid iterations
Application	A set of software instructions that comprise a logical grouping of capabilities, features, or capabilities.
Capability	In terms of software it is the function performed by the application
EAC	Estimate at Completion is the best guess estimate of the final value in terms of a single number. EAC can be for cost, size or other measures.
Effort	The *what* that needs to be achieved by the development team as measured in terms of labor, i.e. story points, hours, teams, or other
Feature	A *prominent* characteristic or capability
Functionality	The purpose or range of operations that the application was designed to fulfill
S-Curve	The cumulative probability of the likelihood that the independent parameter occurs
Scope	The application range; the functions performed by the application
Size	Refers to how *big* the software is as measured by some metric like lines of code, function points, or other.

[1] "Agile Estimating: Straightforward and Simple", by Donald J. Reifer, Reifer Consultants LLC,October 2014
[2] "Software Sizing, Estimation, and Risk Management; When Performance is Measured Performance Improves" by Daniel Galorath and Michael Evans, Auerbach Publications 2006
[3] "Estimating Software Costs; Bringing Realism to Estimating" second edition, by Capers Jones, McGraw-Hill 2007
[4] "Automated Function Points, A Game-Changer in Software Sizing" by David Herron, DCG, Consortium for IT Software Quality, November 2017
[5] "Function Point Counting Practices Manual, release 4.3.1", The International Function Point User Group, 2010
[6] "Agile and Earned Value Management: A Program Manger's Desk Guide" by John McGregor, OUSD AT&L (PARCA), 3 March 2016
[7] "A Cost Model for Early Cost Calculation of Agile Deliveries", Eric van der Vliet, ICEAA Workshop 2017
[8] "Early & Quick Function Point Method, An Empirical Validation Experiment" by Roberto Meli, Data Processing Organization Srl, April 2015
[9] "Simple Function Point Functional Size Measurement Method, Reference Manual" Simple Function Point Association, SiFP-01.00-RM-EN-01.01, March 2014
[10] "How Should We Estimate Agile Software Development Projects and What Data Do We Need?" by Glen Alleman and Thomas Coonce, ICEAA 2017
[11] "Integrating Agile with EVM", by Glen Alleman, Niwot Ridge Consulting, LLC, September 2013
[12] "Heuristic Risk Assessment Using Cost Factor" by Raymond J. Madachy, Litton Data Systems and University of Southern California, 1997 IEEE Software 0740-7459/97
[13] "Guide to Earned Value Management for Agile Projects: Industry Best Practice", National Defense Industrial Association Integrated Program Management Division, 7 December 2015

[14] "Fundamental of Function Point Analysis" by David Longstreet