Technomics
Better Decisions Faster

Software Metrics

Measuring Growth in an Agile Environment

Aubrey Dial, adial@technomics.net

Benjamin Truskin, btruskin@technomics.net

Ken Rhodes, krhodes@technomics.net

Peter Braxton, pbraxton@technomics.net

International Cost Estimating and Analysis Association

2024 Professional Development Workshop

Abstract

Agile software development practices, while designed to deliver value sooner and accommodate changing requirements, are not intended to mitigate cost growth. Nevertheless, Program Managers must navigate this paradigm and control risk while ensuring stakeholder requirements are fully met. Metrics traditionally used to measure growth (e.g., SLOC counts, productivity factors, requirements sell-off) are likely unavailable in Agile projects, and while recent DoD policy recognizes the need for metrics, agile metrics are not standardized, and using them for independent estimation is uncommon.

This paper discusses real-world experience balancing leadership's goals for independent analysis with the realities of an Agile environment. It will show the value of utilizing program-specific metrics and calculating useful measures such as Change Traffic and Feature (in)Efficiency for producing defensible estimates, enabling better program outcomes, and providing insights for others to use themselves.

Keywords: Agile, Data-Driven, Government, Modeling, Performance Management, Program Management, Software

Table of Contents

Table Of Figures

Table of Tables

Table of Equations

1. Introduction

1.1. Agile – Get with the Program

The shift to agile software development has profoundly changed cost estimators' involvement with the software development process as compared to traditional work done under paradigms like Waterfall, Incremental, and Spiral. Agile development, along with changes in technology, contracting approaches, and government culture has forced cost estimators to make changes to traditional approaches to analysis to remain with the times. There are a multitude of reasons why agile became the predominant software development methodology (Dekkers & French, 2018):

- Satisfies the customer with early delivery of valuable software[1]
- Welcomes changing requirements
- Delivers updated working software frequently [2]

Studies show agile projects have consistently higher success rates (primarily based on customer satisfaction) than waterfall development. Statistics range from nearly one-and-a-half (1.5) times more successful than waterfall (Southworth, Hunt, Lucas, & Sanchez, 2023) to the three-and-a-half (3.5) times higher success rate shown in Figure 1 below. However, these statistics are typically not representative of government acquisitions, especially large software-intensive, mission-critical programs. These complex software development efforts challenge the agile principle that cost and schedule are fixed parameters while scope is the variable measure. These are the programs whose growth puts the most pressure on government budgets and warrant consistent, rigorous evaluation, including independent cost estimation.

[1] The oft used term Minimum Viable Product (MVP) comes to mind but doesn't fully encapsulate the agile prioritization process.
[2] This is in part by definition from #1, stripping products to their minimum components, but also via newer software development practices such as Continuous Integration/Continuous Delivery (CI/CD).

SIZE	METHOD	SUCCESSFUL	CHALLENGED	FAILED
All Size Projects	Agile	39%	52%	9%
	Waterfall	11%	60%	29%
Large Size Projects	Agile	18%	59%	23%
	Waterfall	3%	55%	42%
Medium Size Projects	Agile	27%	62%	11%
	Waterfall	7%	68%	25%
Small Size Projects	Agile	58%	38%	4%
	Waterfall	44%	45%	11%

Figure 1: Comparison of Project outcomes by Development Type and Size (Standish 2015)

From previous research (Long, 2022), agile programs studied experienced 57% schedule growth and 24% scope growth[3]. On average, agile programs do not complete 23% of their planned scope[4], furthering the position that realistic program plans should include expected under execution in software development (Smallwood, 2018).

Despite numerous studies showing that agile is not the cure for all that ails software development, the government continues to push agile in the interest of continuous delivery of technical capabilities. Unfortunately, this often goes hand in hand with reduced oversight and a failure to deliver useful metrics and to capture actual cost and schedule data to accompany technical authoritative sources of truth (ASoTs). The fact that agile is

[3] This means that programs experienced 27% schedule growth even when accounting for scope growth. An important point to consider later in our trip around the Iron Triangle and the implications of agile programs focus on near-term planning over total scope.

[4] At the risk of stealing our own thunder, this figure will be shown to be eerily close to the complement of the Efficiency Factor values we've observed.

here to stay for the foreseeable future *and* government-run agile program performance is generally poor requires an analytical leap forward. ***This paper presents a novel method for the cost estimator to model growth and monitor developer performance in an agile environment. It also documents lessons learned for implementing the enabling metrics collection and analysis that underlie the new method.***

1.2. Agile – The Dilemma

Currently, more than 71% of U.S companies predominantly use agile as their software development approach (Southworth, Hunt, Lucas, & Sanchez, 2023), which means that the workforce supporting government development is caught in the same sea change. Agile software development puts emphasis on feedback, responsiveness, prioritization, and replanning. Unfortunately, continual replanning (i.e., rebaselining) is typically not accompanied by continual evaluation of measured sprint-to-sprint (or increment-to-increment) performance.

This represents a serious problem! Why? ***Because the credibility of capacity-based projections for future sprints and increments necessarily depends on thoughtful, systematic use of recent measured performance data.*** In other words, the preferred approach to continual agile project estimating and planning is Extrapolation from Actuals (International Cost Estimating and Analysis Association, 2020).

As logical as Extrapolation from Actuals sounds, the United States Government (USG) lags in implementing policy and practices to ensure that the “right” recent performance data is captured for its agile projects and available to analysts for estimating the subsequent sprints and increments. Not to mention that even if policy were keeping pace with the rapid adoption of agile, many current Department of Defense (DoD) and other major USG contracts extend years into the future and are delivering whatever data were agreed to at the time of award without incentive for contractors to change.

There is some movement on DoD policy, specifically the Software Acquisition Pathway (SWAP) described in DoD Instruction (DoDI) 5000.87, that is partially meant to address this. However, adoption is slow and high-level guidance does not mean the world is suddenly replete with agile metrics contributing to more realistic cost estimates. The

Software Resources Data Report (SRDR), for example, has been adapted for use with agile programs, but it was designed as an annual (or less frequent) Contract Data Requirements List (CDRL) and risks lagging when data are needed for estimating purposes.

In the meantime, and making matters worse, historical software development datasets and existing parametric software estimating tools continue to reflect a disproportionate number of waterfall development efforts, which may be of limited value in estimating agile projects. Lacking metrics, we tend to fall back on subject matter experts (SMEs), which risks weakening the credibility of our estimates. In the next section, we further explore the agile environment and how it informs the approach of estimators.

1.3. Agile Estimating Framework

Agile software development, with its higher emphasis on requirements evolution and user feedback, requires each iteration of development to provide the user with mature sub-elements of the overall capability. This development process creates solutions and products through the collaboration and alignment of cross-functional teams. Each team leverages concepts such as those from the Scaled Agile Framework (SAFe), a predominant approach to Agile that we will reference for common terminology in this paper (Hoffman, Davis, Ashwood, Smith, & Lane, 2019). This framework is designed to help in streamlining the process to enable the enterprise to make quicker decisions, communicate effectively, simplify operations, and remain focused on the customer needs. (Scaled Agile, Inc., 2024). In much the same way that SAFe requires developers and customers to focus on near-term needs and quicker decisions, estimators must adjust their *Weltanschauung* to enable an improved level of support to, and engagement with, these programs.

Agile acquisitions de-emphasize total program scope, the main scalar that cost estimators look to when estimating most projects, in favor of pre-defined capacity and time blocks that force near-term planning and deployment of resources to meet user needs more rapidly. Figure 2 contrasts this approach with Waterfall where scope is planned to meet requirements and resources are estimated and then phased accordingly. While long-term activity definition is not the focus of agile programs, requirements decomposition (scope)

and incremental planning (capacity) give insight into the overall scale of the project. That does not mean estimators are completely in the dark…many have begun working to crack this proverbial nut and estimate scale for this new development approach.

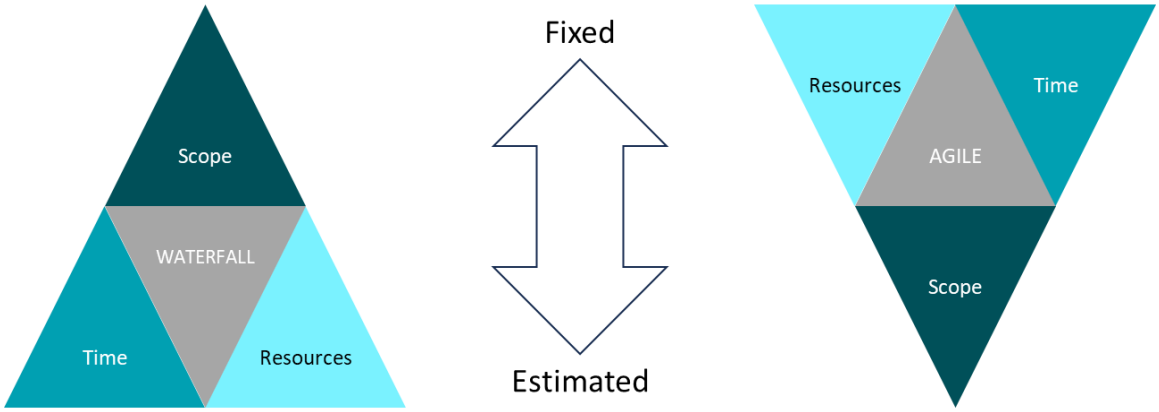

Figure 2: Priority shifts in Software Development Approaches

Several methods exist for estimating future agile software development effort; however most require SMEs to forecast the volume of scope planned to be accomplished via T-Shirt Sizes (ranges of hours), Story Points (estimated hours per requirement), or Scrum Teams (people). All of these are tantamount to direct estimation of hours, which is the acid test for Expert Opinion, frowned upon as a primary estimating technique (International Cost Estimating and Analysis Association, 2020). While these sizing sources have the benefit of collecting information closer to the source of the development activity, they have been shown to be highly variable, subject to SME input bias, and challenging to standardize. One example is T-Shirt Sizing, where in SMEs attempt to size SW by picking from a list of T-shirt 'sizes' where each size is double the preceding and half the succeeding. Prior research (Braxton P. J., Brown, Rhodes, & Wekluk, 2022) shows that when including technical uncertainty of the baseline, the unintended consequences of such a simple sounding sizing approach include implicitly adding anywhere between 6% and 33% growth and a resultant coefficient of variation on the technical baseline of 36%-66%.

It is important to note that the challenges laid out above exist both at the micro level (i.e., across teams or time within a project) as well as the macro level (i.e., across projects in a portfolio or between software development focused organizations). No matter what the

purview of an individual estimator, in the ever-evolving world of agile it is as important as ever to collect as many of the technical measures as possible and study homogeneity across time.

One last hurdle that must be navigated by cost estimators looking to adapt to today's landscape of agile development is the rapid adoption of fixed price contracting alongside agile development. Because agile focuses on constraining capacity and schedule, while varying scope (Figure 2 above), a firm fixed price (FFP) contracting approach is often attractive, as it is believed the execution of work is "lower risk." Ultimately what tends to happen from a cost estimator point of view is that the challenges with agile estimation get compounded with contracting woes of lesser (or no) data deliveries.[5] DoD's Cost and Software Data Reporting (CSDR) process has long fought against the misnomer that FFP contracts must entail reduced CDRL requirements. In fact, CSDR remains a statutory requirement across all contract types.

1.4. Performance Measurement and Metrics

Alongside the rapid adoption of Agile as the predominant software development method, government acquisition offices have been pushing to increase data driven program management through the collection of metrics. It is increasingly common for government agencies to have a Chief Data Officer (CDO), or an office dedicated to data and analytics. While these CDOs may not be specific to software, the cost community should attempt to ride the wave and leverage these types of organizational investments. Not leveraging these data offices and officers unnecessarily puts estimators at odds with key stakeholders in their respective organizations, and risks cost estimators' being viewed as a resistive force to modernization.

[5] Stay tuned for Section 3.3 where the impact of reduced data deliver on a fixed price type project forced 'creative' data collection approaches and Section 4.2, where we explore a case study and how the interplay between cost and schedule can be viewed by analysts.

One popular set of metrics has been advanced by DevOps Research and Assessment (DORA) centers on the following focus areas (Dr. Nicole Forsgren, 2019):

- Deployment Frequency: How often new releases are pushed into operations
- Lead Time for Changes: Average time to deliver a Feature into operations
- Time to Restore Service: Time to recover from a service incident
- Change Failure Rate: Percentage of changes that result in degraded service
- Availability: Percentage of the time the system is available to users

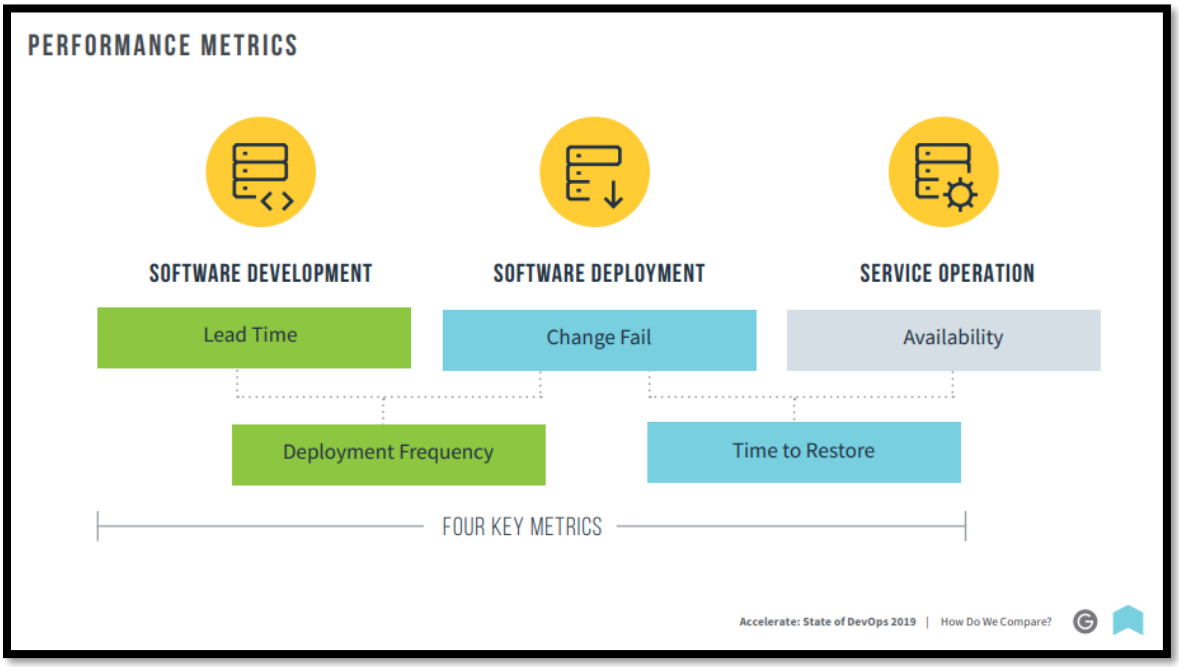

Figure 3: DORA Metrics

Our team observes that these metrics tend to focus on schedule as primary and technical scope as secondary. Cost, the third leg of the so-called Iron Triangle, is barely addressed while capacity (i.e., investment required for a given throughput) and plan versus execution are rarely touched upon. One way of seeing that Cost is important to consider is the *reductio ad absurdum* to which pure Schedule metrics are prey. Take the first bullet above, deployment frequency, for example. Developers can “excel” by delivering smaller and smaller releases more frequently, but what is the total value delivered (per time)? This starts to sound an awful lot like integral calculus: what is the “area under the curve”

as the number of releases approaches infinity but the content of each release approaches zero?!

Agile metrics tend to focus solely on actuals and not comparison to plan[6]. Also, the prevalence of dashboards tends to make us prisoners of the moment, showing current status but neglecting trends over a substantive period of time (at least a year or two). This paper highlights the importance of establishing plans of varying time horizons, tracking progress against them, and of examining trends – are we getting better or worse over time in delivering value to our customers?

2. Traditional Software Measurement: Inputs and Outputs

2.1. Sizing Estimates, Counting, and the Progression of Time

For a long time, cost estimators were able to build software estimating techniques around a couple of key features of the waterfall development paradigm.

First, a complete baseline provided the framework for measuring progress within a program or comparing a current program to a past program. Everyone knew how much scope was planned and how that aligned to the projected schedule and estimated level of effort (or cost). While the final deliverable product may not have been exactly what the customer wanted or needed (a significant reason agile was introduced), there was little debate about what the plan was or how it was modified over time.

Second, most waterfall programs were required to collect metrics related to size or scope (e.g., Source Lines of Code, Function Points) *and* provide counts to decision authorities as part of the approval process to pass from one gate to the next of the acquisition process or development cycle. There has been much justified discussion about how to

[6] Stay tuned for the discussion in Section 4.1 and 4.2 of the importance of measuring both completed work and the baseline plan is important!

measure size, including: 1) was it best assumed to be an absolute or relative measure, 2) which measure of size is most appropriate, and 3) how accurate sizing approaches were (ranging from analogy to judgment). Even so, the consistency of delivery of these measures amongst enough of the industry gave estimators the consistent technical drivers needed to develop strong methods and estimate development costs. Substantive amounts of reported data were consolidated into repositories of such as NAVAIR's Software Database, derived from DoD's collection of SRDRs across the past two decades and available in the Endorsed Datasets, Tools and Models Hub of CADE Data and Analytics app (Draheim, 2022). Such data sets were purpose-built expressly to support these methods.

These features of waterfall development have supported the practice of traditional cost analysis, and more broadly, program management techniques such as Earned Value Management (EVM), would generally remain valid. Being creative and adapting these foundational approaches to measure the scale of agile over time is what cost estimators of today must do to produce realistic estimates.

2.2. Software Productivity

At their most basic, software CERs come down to an expected productivity: how much software a given development team can produce in a given amount of time. More precisely, most CERs calculate a reciprocal productivity, as in hours per line of code.

The typical Inputs-based Risk approach begins with an initial Size estimate and applies expected growth, as summarized in the previous section. Then, the risk-adjusted value is the input to an Effort Estimating Relationship (EER) or CER to produce an estimate of Effort or Cost, respectively (working the outside of the diagram in Figure 4 below). An alternative to this Inputs-based approach is to develop EERs or CERs that implicitly include expected growth, which is shown as the cutting diagonal in Figure 4. These would be derived from Final Cost as a function of Initial Size. This approach is less prevalent for several reasons:

1. It violates the general regression assumption that the independent variables should be deterministic and not stochastic.

2. It requires a family of CERs, each based on a set of historical programs with a comparable degree of maturity of the technical baseline.
3. It masks interactions between optimistic forecasts of size (we needed to develop more code than we thought) and productivity (we were less efficient in developing code than we thought).

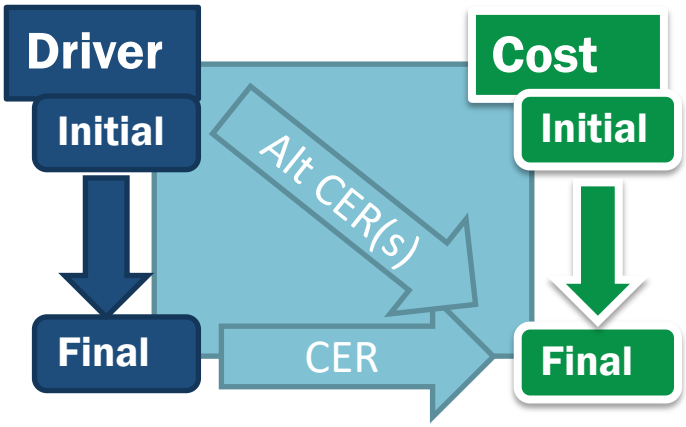

Figure 4: CERs with Explicit and Implicit Growth

While agile programs may use different terminology (velocity, feature count, etc.), they fundamentally can convert their development effort into the same sorts of measures and be estimated with the same cost estimating approaches (not specific CERs though!) as traditional development paradigms. However, many organizations have not made it over the gulf of collecting enough Final Costs to build agile-specific CERs and do not have data-driven methods to provide support to programs currently undergoing development. This dearth of data and lack of consistent productivity metrics has led our team down the path of the research presented here, focused initially on contract-level productivity measures with the goal of further expansion.

3. Agile Performance Measurement

3.1. Rolling Wave Planning in Agile

All agile development approaches require an incremental approach to project development, with a fixed period of time for each cycle and some standard milestones for entry and exit. Typically, these fixed periods are referred to as Increments or Program

Increments (PI's) and in our experience span around three months on average. Our paper will maintain the naming convention 'PI' to distinguish from the Increments of a major acquisition program and the (overlapping) increments of the traditional Incremental software development approach.

A regular part of the Agile approach is the PI planning stage, where the team reviews the status of the development effort coming out of PI *N-1* and finalizes the plan for PI *N*. Some common names for these quarterly events include Program Increment Planning Event (PIPE), Rolling Wave Planning Event (RWPE), and Program Increment Summary Event (PISE). Next is the execution stage, usually split into *K* smaller development sprints of 2-4 weeks where PI *N* is actually worked. Lastly there is the sell-off stage, where PI *N* features are formally delivered and baselined for integration into the software baseline, and the backlog for PI *N+1* is finalized. For operational systems, this integration may happen either in cycle with a PI (e.g., PI 10 will be a software integration heavy increment) or happen out of cycle and maybe even on an entirely different contract[7] (as in Figure 5 below).

Like a quality cost estimate, the Agile development OPTEMPO has "no gaps or overlaps." Planning for PI *N* must be accomplished near the end of PI *N-1*, before delivery is complete, or near the beginning of PI *N*, after development has already started. (In our experience, the former is more common.) Similarly, the retrospective for PI *N-1* is conducted after development for PI *N* has commenced. The phrase "no rest for the weary" might come to mind, except that Agile teams are supposed to be sustainable and not rely on heroics to deliver any specific piece of software. "Slow and steady wins the race" is more like it. (OK, maybe not "slow" so much as measured!)

[7] Note: The inconsistency of delivery of features to end-users across contracts or within a contract but across time directly contributes to the challenges of estimators using common agile metrics discussed in Section 1.4.

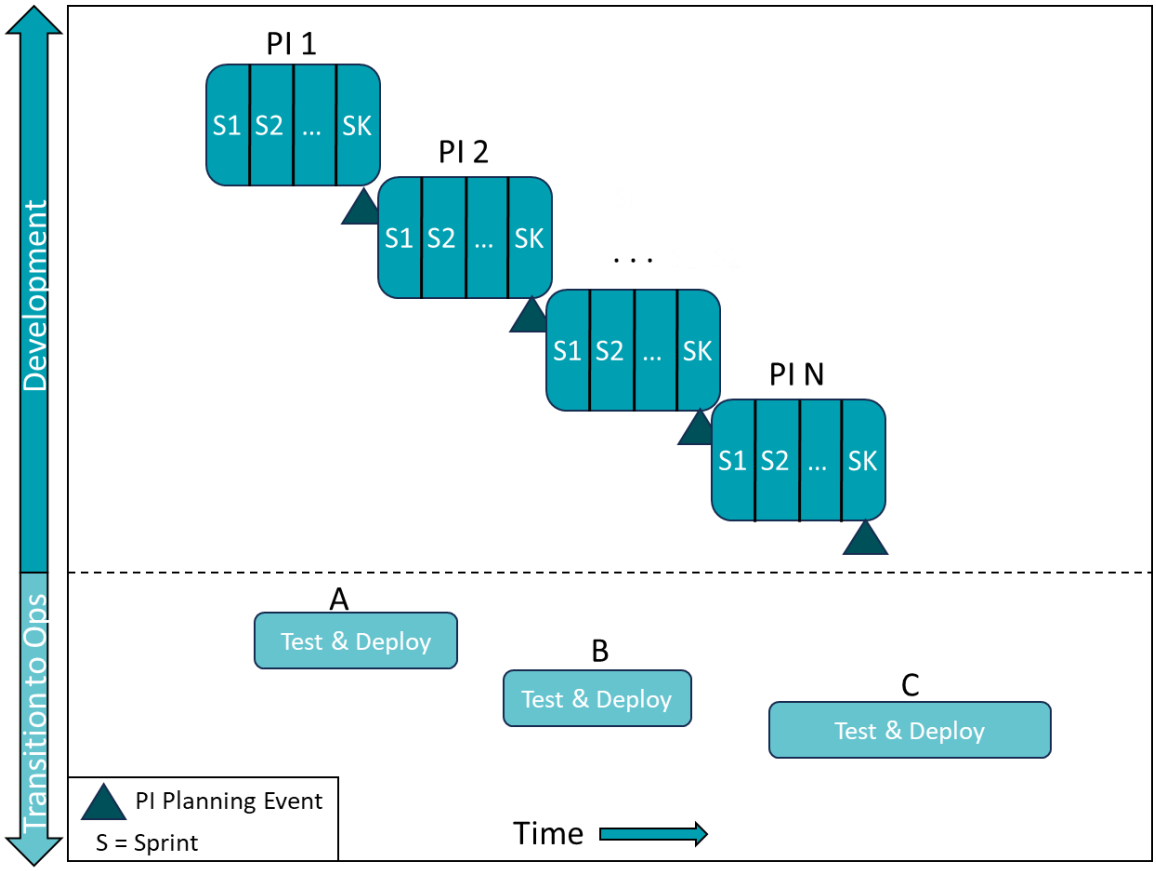

Figure 5: Program Incremental Planning Process Depiction

A feature of rolling-wave planning is that development can be more responsive to a dynamic environment or changing customer needs. While this responsiveness is essential to the agile development manifesto[8], that does not mean that a baseline (albeit more limited in scope than traditional software development paradigms) cannot be laid out and documented. Few who preach agile as their preferred software development approach suggest that doing so means that no measurement of performance can be done, and many more simply specify that the means for doing so must change. For example, DoDI 5000.87, which defines the SWAP, requires annual updating of the

[8] For those who may have forgotten, the final of the four stanzas of the manifesto reads as follows (emphasis from source): "**Responding to change** over following a plan."

technical baseline and cost estimate (Office of the Under Secretary of Defense for Acquisition and Sustainment, 2020).

Rolling Wave is an approach usually associated with EVM, wherein Work Packages define detail-planned short-term work and Planning Packages reflect rougher definition of long-term work. These concepts apply quite nicely to Agile, where PI's N and $N+1$ might be at the Work Package level of detail; the next several PIs in the Road Map might be at a Planning Package level of detail; and multi-year program budget might be based on an even less well-defined strategic portion of the Road Map.

Our foundational hypothesis is that ***agile programs are not immune to the inefficiencies related to replanning that plague more traditional software development approaches***, but because of the shorter baselines and focus on delivering value, the full picture of development cost is overlooked.

3.2. Features and Their Central Role

While the incremental approach within the Agile Framework provides bounds for development effort against schedule (and thus cost), it does not negate the need to measure planned development effort via some standardized means. Features are one measure amongst many to do this, but the one that we focus on in this paper. Features are a Goldilocks ("just right") size measure more granular than Epics but less granular than Stories.

Feature-Oriented Software Development (FOSD) is an approach for agile software development typically used on large software systems, similar to story points or function points, but focusing on decomposing the system to the features and value it provides to end users (Apel & Kästner, 2009). The primary unit of measure, a feature, *is a unit of functionality of a software system that satisfies a requirement, represents a design decision, and provides a potential configuration option.*

In short, a feature is a standard measure of capability small enough to be completed within a PI. As in any agile approach, it is understood that broad user requirements are defined and fleshed out in parallel with software development. Features that are well defined and have significant user impact are then planned into PIs and the sprints within

them, and then executed upon. While not all Features are created equal, but they are sufficiently consistent to use for our central construct of Planned vs. Actuals.

The funnel graphic in Figure 6 is a typical depiction of the scope planning process one would find when working with an agile program. However, our team made an important addition to the graphic to depict a more realistic view of how agile programs operate…what happens when things stray from the plan?

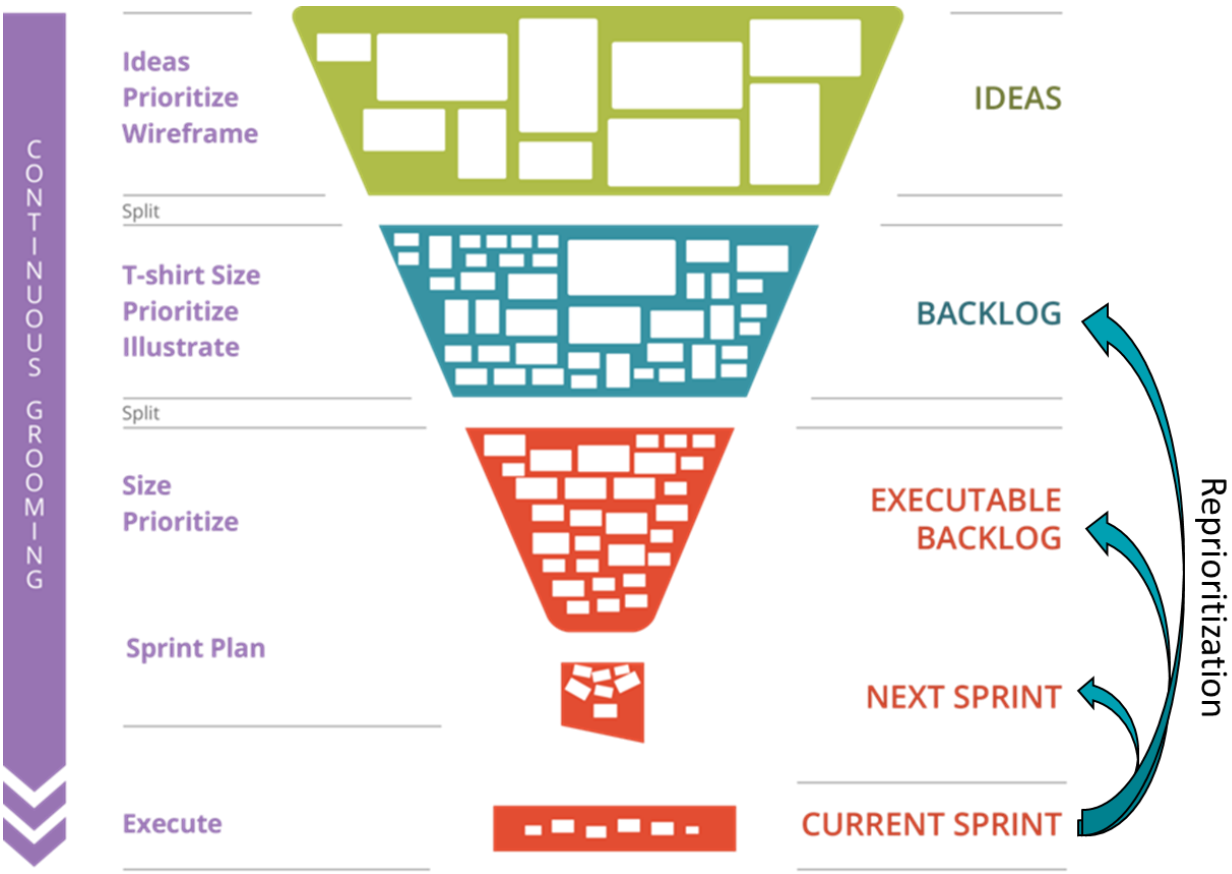

Figure 6: The Agile Funnel, Backlog to Sprint Planning (credit: ITX)

Cost estimators do not have to be Heraclitus to deduce that change is the only constant in software development. In execution of a sprint or PI, it is a certainty that development will not go to plan. Whether it is an emergent requirement that demands immediate attention, a feature that needs to be split into smaller pieces, or something that is in the immediate plan and is determined to be no longer necessary, the baseline will change. That recognition forms a large portion of the entire premise of why agile software development was needed in the first place!

Those unmet or modified requirements must be re-prioritized and the knock-on effects to related features must be considered. Therefore, a more realistic depiction of actual software development backlog maturation includes the addition of the feedback loops on the right side of Figure 6 from the bottom of the funnel up to at least the preceding three layers. Closing this feedback loop is the added engineer, architect, and customer time spent now reintegrating those repatriated features back into the backlog flow and ensuring they are ready for development when a given feature reaches the bottom of the funnel again.

3.3. Efficiency Metrics and Applications for Cost

Features measurement and PI planning is analogous to classical EVM, just tailored to agile developer terminology. Like EVM, estimators can account for actual behavior of their systems the same way as Cost Performance Index (CPI) and To-Complete Cost Performance Index (TCPI) are used in classical EVM space. For more background on EVM, see Module 15 of CEBoK (International Cost Estimating and Analysis Association, 2020).

During execution of a development effort there is a baseline of features planned, and those features can reach one of a few states by the end of a PI:

Completed – Feature successfully developed and ready for integration into software baseline

Added – Feature not originally in PI plan added for development (and completed) in current PI

Deleted – Feature slated for development in current PI is determined to be no longer relevant and removed from the development queue[9]

Moved – Feature moved from current PI to a later PI for future development

[9] There may be both “hard” deletes, when there is consensus that the feature will never be relevant again, and “soft” deletes, where it is relegated to deep within the product backlog.

Split – Feature is split across multiple PIs, with some work done during the current PI on the feature and some work pushing to future PIs, usually the succeeding one.[10]

These categories are representative of what we observed on various programs, but terms and definitions may vary across the broader software development community. It is incumbent on the cost estimator to hold Technical Exchange Meetings (TEMs) with the developer(s) and product owner(s) to gain better insight into the local execution of the agile process.

With basic definitions for agile software development covered, it is pertinent to provide some background on the Technomics' team experience working with the acquisition offices to understand their agile software metrics. At each PIPE, the developer(s) and USG program office review the outcomes of the prior increment as well as the implications and the plan for future increments. The focus tends to lean on the count of features completed or planned and released to users (see lower thread of Figure 5), but when looking backwards, there is very little in the way of historical metrics discussed. As this trend continued from one planning event to the next, our team added value by beginning to track each program's metrics EVM-style, based on what was briefed, to better understand the immediate PI performance and longer-term trends. The simplest to explain and the one that gained the most traction with Project Management Offices (PMOs) was Feature Efficiency (Equation 1). Those familiar with EVM may recognize feature efficiency as the agile software developer's version of performance index, though whether it's most analogous to CPI or Schedule Performance Index (SPI) depends heavily on the contract structure.

Equation 1: Feature Efficiency Definition

$$Feature\ Efficiency = \frac{Completed\ Features}{Planned\ Features}$$

[10] In the subsequent equations, the feature change types will be abbreviated *Add*, *Split*, *Move*, and *Delete*.

The corollary metric that is being tracked as the hypothesized cause to degrading Feature Efficiency is Change Traffic. This is in part a nod to our previous description of agile developments as a "'12-lane highway," where work is occurring in parallel on several different related efforts (Braxton P. , Brown, Rhodes, & Wekluk). For those familiar with rush hour traffic, the amount of change in traffic on a highway (by vehicles merging on and off at an exit for example) directly affects the progress of all the other vehicles involved, especially those just trying to pass through and continue on their way. Crude analogies aside, this physical world exemplar forms the foundation of our hypothesis in this paper that ***the magnitude of changing traffic relative to planned throughput is inversely related to how much of the baseline gets completed.*** Our analysis tests the assumption that a correlation exists between the number of features changing from a baseline and the inability to execute to the planned workload.

Equation 2: Change Traffic Definition

$$Change\ Traffic = \frac{Added\ Features\ +\ Removed\ Features}{Planned\ Features}$$

$$Change\ Traffic = \frac{\sum Features\ (Add,\ Move,\ Split,\ Delete)}{Planned\ Features}$$

There are two large assumptions made in Equation 2 above that are worth further exploring. First, the assumption about what to use as the denominator in our scaling approach: Planned Features (analogous to the Return on Sales, or *ROS*, formula for profit, wherein the numerator is included in the denominator) or Actual Features (analogous to the Return on Cost, or *ROC*, formula for profit, wherein the numerator is excluded from the denominator). Our hypothesis would indicate that focusing on the ROC approach would result in higher correlation because of the implied relationship between numerator and denominator. However, since "ROC" is retrospective (we can only know actual completed features for prior periods) the paper will focus on the "ROS" approach (we can know what the developer is proposing at least for PI N) for more forward-looking predictive analysis in Section 4. Figure 7 below proves out the above assumption of our hypothesis in an exemplar dataset, showing a higher correlation between feature efficiency and ROC-style change traffic over the ROS approach. It is not necessary to know the precise definitions of ROC and ROS to appreciate the analogy, but they can be

found in CEBoK Module 14 “Contract Pricing” for those interested (International Cost Estimating and Analysis Association, 2020).

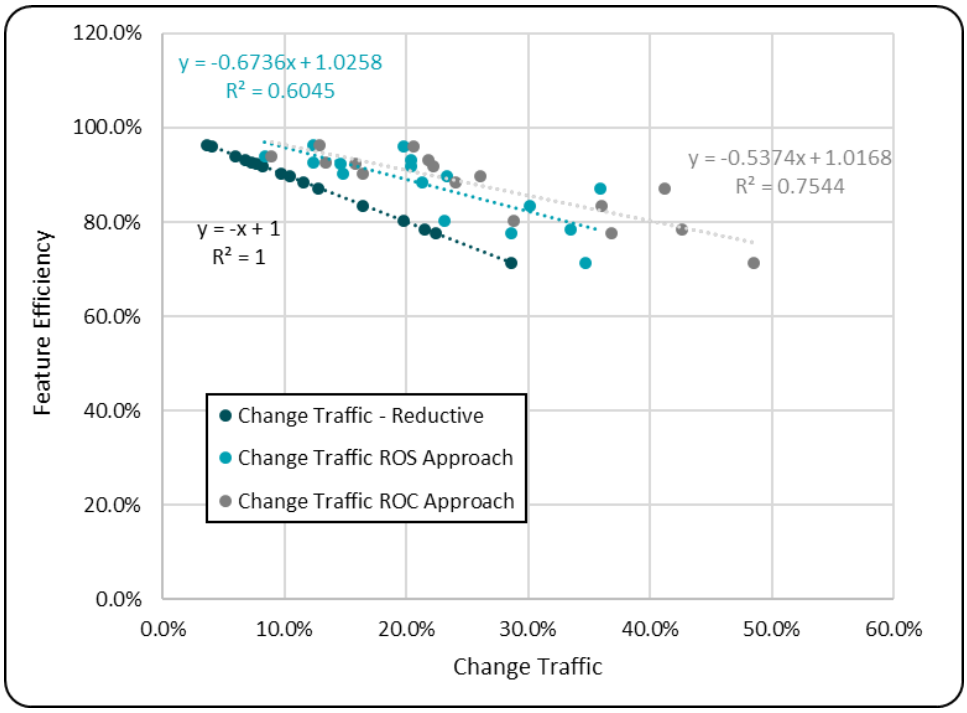

Figure 7: Comparison of Change Traffic Computation Metrics

The second assumption is the decision to sum the added and removed features in the numerator rather than just look at the net shortfall to the planned feature count. (This is tantamount to the absolute value of all feature change types as opposed to signed values.) While the latter sounds interesting, making the numerator the delta between actual and plan, and then dividing by one of those two same measurements, results in something very reductive. For example, that would assume an increment with 0 added or removed features behaves the same as one with exactly 100 added and 100 removed features. This reductive measure is further borne out with the R^2=1 trendline shown in Figure 7 above, as the shortfall perfectly predicts the shortfall. (This problematic finding is further explored in Section 9.1.) For that reason, we will stick with the change traffic calculation as depicted in Equation 2.

At this point, our team acknowledges that while the preceding discussion in this section appears logical and straightforward, the analytical path to publishing this paper was quite circuitous. The team originally began looking at the data and coming up with the

'reductive' line in Figure 7 above, which really is not much of a finding. After nearly getting lost navigating the proverbial Heywood Woods and missing the forest for the trees, it was not until months later and continued performance challenges in the monitored programs that we fully recognized our error. We shifted our focus to the magnitude of traffic rather than direction of changes alone. This enabled us to make deductive reasoning about likely outcomes, but the time trending and predictive analysis required took an additional year of collected data, as we were only getting updated frames in the movie on quarterly basis. Let this be a lesson about the benefits of persistence and patience in analysis!

3.4. The Peril of Circular Regression

Even if we avoid the tautological case, any same-period regression between Feature Efficiency and Change Traffic remains problematic, since the same feature counts feed both metrics, creating a direct relationship that runs counter to the spirit of regression. We will continue to use it as a baseline for comparison for this paper, but keep in mind that it represents a "too good to be true" case. Even if the regression were kosher, it would not be practically useful, since we will not know the feature counts needed to compute Change Traffic metrics until the PI is over, in which case we no longer need to predict Feature Efficiency, we can measure it directly. (This is known as "salting the bird's tail.")

Before we make adjustments that allow us to use Change Traffic as a leading indicator, not a lagging one, we delve a little more into a conceptual understanding of drivers.

3.5. The Philosophy of (Cost) Drivers

Knowing that inefficiency exists is not the same as understanding the underlying drivers. Nobody plans on changing requirements on day 1, yet every software acquisition does it somewhere along the way. By now, there should be a common understanding between the customer and developer that changing requirements will happen and plans for how change will be handled must be established. Regardless, limitations in prediction do not mean we have to ignore the effect (we also don't know the exact weather in December, but we know it is generally a bad time to garden). This is no different than when cost estimators use proxy measures that we know correlate well to cost (or whatever dependent variable we are trying to predict) such as tonnage for ships or mass for space

vehicles. While these proxy variables have some level of explanatory power (e.g., more mass means more material), the logic and reasonableness of their impacts breaks down when pushed to lower levels of granularity or for more specificity (e.g., an extra ton of hull is not the same cost impact as a ton of guidance and navigation equipment). However, with the appropriate guidelines and caveats cost estimators can do their job and the world keeps turning (or at least budgets keep being executed).

4. How Does Inefficiency Impact the Iron Triangle?

4.1. Exemplar Problem[11]

For this paper, our team is leveraging a notional dataset modeled after observed behavior from data collected for actual acquisition programs. The construction of the dataset values are representative, with behavior typical of surveyed trends from recent acquisitions. Our team collected and analyzed data from programs across significant periods of performance, over the course of several years, with software development progress being measured in approximately quarterly increments. The increments often comprise four sprints of three to four weeks each, though there is variation at the lower level depending on specific software development activities and calendar effects (e.g., holidays/vacations).

The software development effort is composed of several swim lanes of effort, though for purposes of our analysis we are not showing results at that level of granularity. In our exemplar problem we measure the plan and actual delivery for features by increment, as well as the count at the feature level for the categories laid out in Section 3.3.

So how do things look in our exemplar problem? Figure 8 shows the program delivering approximately 225 features per PI to start and increasing delivery increment over

[11] Disclaimer: While we are attempting to show typical results, correlations and CERs shown should not be used as is. The goal is to describe a process that the reader can faithfully replicate with their own data.

increment consistently. Wouldn't any project manager be happy with this? What's the problem?

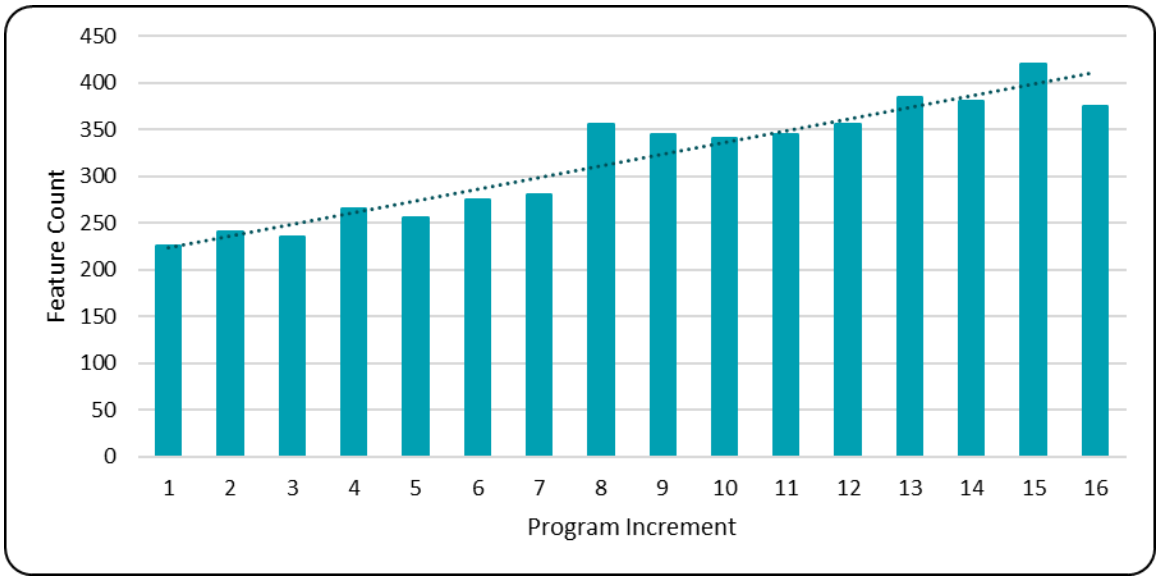

Figure 8: Delivered Feature Count by Increment

While it is clear in Figure 8 that the count of features is going up over time, Figure 9 provides a more complete picture. This graphic shows both the actual completed features by increment, but also the shortfall as it relates to the planned features for each increment. While the shortfall is relatively small early on, it is clear there is a bow wave effect building of uncompleted work. Independent analysts would be right to question the sustainability of the upward trend in features delivered, or start looking to other dimensions (e.g., financial, staffing) to see the measurable effects of this growing backlog.

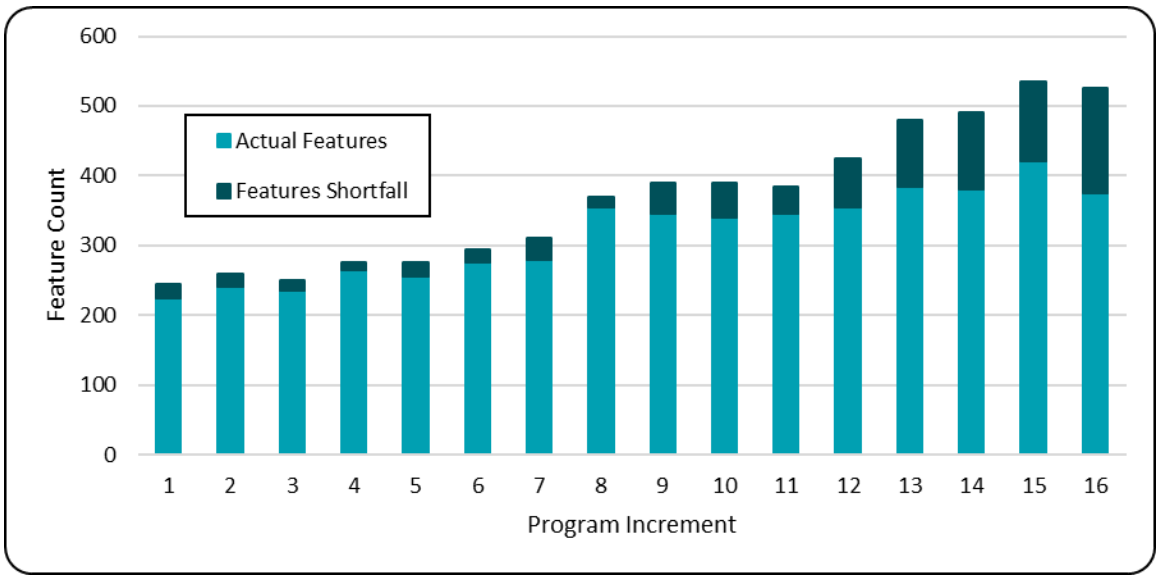

Figure 9: Planned Feature Count by Increment

Keep in mind that "All risk is relative." Our central Feature Efficiency metric is essentially a risk benchmark, and there are two symmetric contributors to an undesirable value: poor planning and poor performance. If the contractor is biting off more than they can proverbially chew, this could be a result of *aggressive planning* (committed to more features than is reasonable at the beginning of the PI) or *poor performance* (not achieving a reasonable feature throughput for the PI) or *both*. This will constitute the focus for the remainder of Section 4.

4.2. Iron Triangle Components – Cost, Schedule, and Technical

Our first stop around the iron triangle is **cost**. In the world of FFP contracting and agile development, cost is often substituted for content delivery. By utilizing the Feature Efficiency and Change Traffic equations covered in Section 3.3 and applying them to our exemplar dataset we can see how our newfound measures apply to this problem (Figure 10).

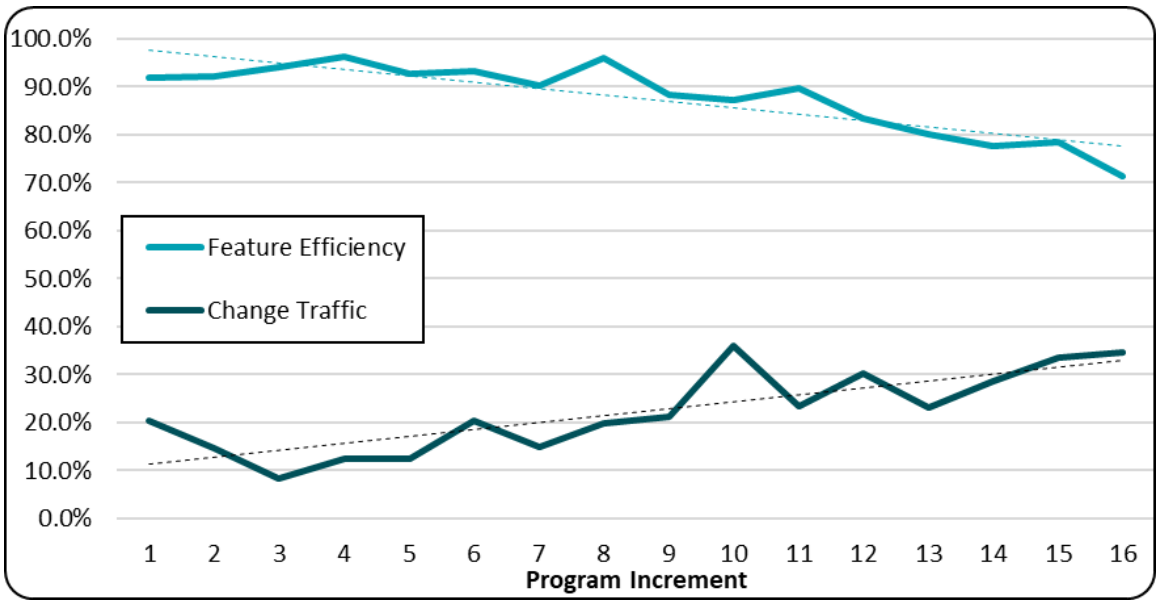

Figure 10: Relationship of Change Traffic to Feature Efficiency

There is an apparent inverse but steady relationship that as Change Traffic increases, Feature Efficiency decreases, and the same-period correlation can be calculated as -0.77. Contrast this with the "increasing capacity" show in Figure 8. The strength of the relationship between these two variables is further backed by measuring the significance of the slope of the ROS feature efficiency measure presented in Figure 7 (based on the same dataset), resulting in a p-value of 3.9×10^{-5}, well below the 5% threshold of significance. The trend is undeniable, so the honest question becomes "Is the PMO alright with paying a significant premium per delivered feature, if those features are the most important portion of the backlog?".

The second stop on the trip around the iron triangle is **schedule**. While for the first eight PIs the contractor averaged delivery of 93.3% of planned features, the contractor steadily shows degrading performance, to the point that the contractor is only delivering on 71% of their plan in the latest increment. Clearly the contractor cannot deliver anywhere near their planned content on schedule…So what gives?

One explanation for the worrisome trend may lie in the natural but ill-advised reaction to slipping scope…overcorrection. If one looks at the planned features (the total stacked bar in Figure 9), it is easy to see that after a couple of strong increments (PIs 7-9), the

developer pushed planned capacity[12] higher in PI 10 and saw a change around that time where their delivery to the plan has degraded and not come back[13]. If the developer had paid more attention to the total magnitude of changes (change traffic count, the numerator of Equation 2) instead of just directionality (shortfall), someone may have seen this coming. While the PI 10 shortfall does not appear odd in Figure 11 below, PI 10s change traffic stands out as the inflection point to degraded performance. In fact, it is easier to see that the contractor should have been paying more attention to change traffic a year earlier, as a spike in change traffic in PI 6 also had immediate impacts on both feature efficiency discussed earlier, and the shortfall shown below.

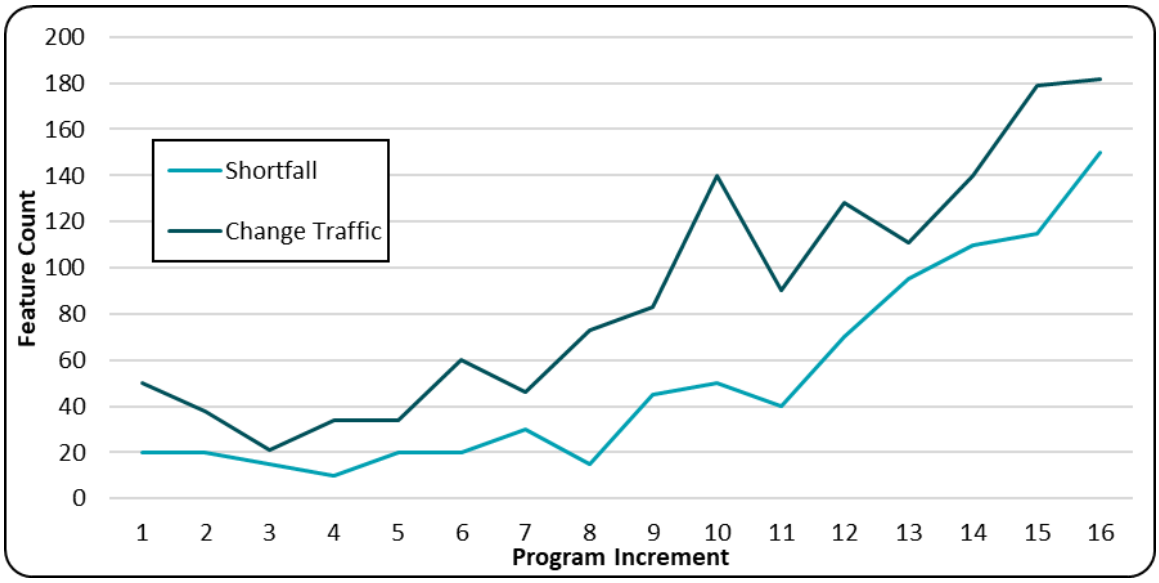

Figure 11: Shortfall vs. Change Traffic by PI

The bow wave of deferred work, which is shown in Figure 9 at the conclusion of Section 4.1 (and can be inferred from Figure 11), has several possible ramifications:

- Delivered software is not as capable as planned at a given point in time relative to original planned.

[12] Capacity is directly proportion to both total team size and average productivity. In our notional scenario, we're assuming that team size is relatively constant.
[13] Readers already familiar with EVM approaches will find this very similar to other cases of PMO optimism where CPI and TCPI begin diverging due to an unrealistic BAC, EAC or LRE.

- Schedule extensions will be required to complete development or original scope.
- Attempts to curtail schedule growth may require development "crashing" to complete key functionality, which may exacerbate cost challenges (and is antithetical to the agile philosophy to boot).

Note, while velocity typically increases throughout a development effort, it is more likely due to the requirements decomposition (i.e., more/smaller story points) and not due to the team's ability to complete more complex requirements more efficiently).

While development schedule challenges might be the portion of the iron triangle most observed in support of Continuous Integration / Continuous Delivery (CI/CD) pipelines focused on the end-user, developer performance metrics and overall contract monitoring are critical components of program management that cost estimators and Program Managers cannot ignore. Given so many agile programs leverage LOE style contracting or have limited mechanisms to hold developers accountable for lack of delivery, schedule slips tend to translate to cost growth. Even worse schedule slips mean a longer delay between definition of a need and development to satisfy that need. This may lead to scope being developed that is outdated compared to user needs or at risk of being done less efficiently, for instance by being pushed to a follow-on contract executed by a new developer. The new developer might have less familiarity with the program and be less efficient in development.

The last of the vertices on the iron triangle is **technical**. For our simplified dataset there are limited avenues of insight to consider as compared to all the ways development effort may be categorized in the real world with more complex systems. One way to approach the problem is by looking at the evolution in the contributors to change traffic over time (Figure 12).

While our team's first instinct when reviewing the degrading performance of real-world data assumed that there must be a growing amount of government intervention via late breaking additions, this is not the main driver for the growing shortfall. Review of our exemplar dataset in Figure 12 shows similar behavior, with little growth in added features over time, and thus limited impact of that portion of change traffic on the growing backlog.

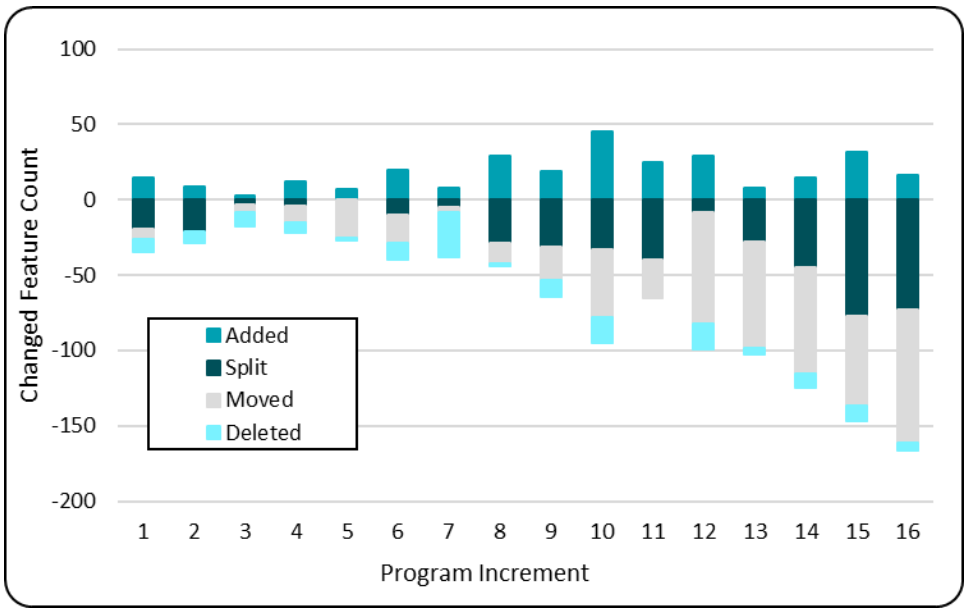

Figure 12: Evolution in Change Traffic by Component

What appear to be the primary contributors to the growth in change traffic are the features being *moved* and *split*. After getting past the initial instinct of additions to the baseline as the most impactful, this makes a lot of sense. Moved features require better definition, reinsertion in the program backlog, before another attempt at development can be tried. This results in a lot of unplanned non-development time and effort. Split features are even worse, requiring contractors to determine how to break apart what are supposed to be indivisible units of scope, and then immediately prioritize the portion of scope not worked in PI *N* into the succeeding PI *N+1*. Contrast this with the addition of a feature to a PI, which likely means that it is of great importance and probably has had the resources applied to define it well enough for immediate development. (We will test this a priori reasoning later by exploring weights for the different change types, which we have thus far naïvely[14] assumed to be equal.)

On a real program, we might look to the individual swim lanes for insight into competing priorities and root cause analysis, though we should always beware the slippery slope of

[14] Naïve in the mathematical sense, meaning we start with simple counts until there is a preponderance of evidence that we should deviate from them.

rationalization. True Root Cause Analysis (RCA) may require technical detail and expert testimony, for which the *de rigueur* agile retrospectives and postmortems may be an excellent source.

Finally, from a technical perspective, our team looked at whether there were economies or diseconomies of scale as it related to execution of PIs. Should the expected feature efficiency depend not only on change traffic, but also on the relative amount of development to be accomplished?

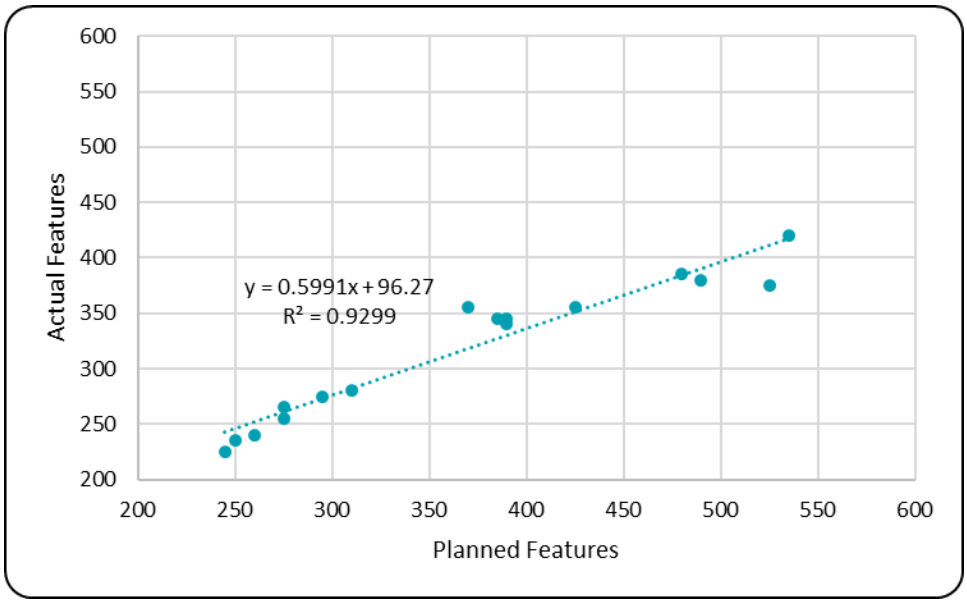

Figure 13 : Planned vs. Actual Feature Deliveries

To answer this question, our team plotted how the planned and actual features relate to one another in Figure 13 above. It is clear that based on the slope of the line that each additional planned feature is not resulting in a completed feature on average within the range of data from PIs 1-16. Also, looking at the shape of the residuals from the linear trendline one could see that there may be a diseconomies of scale effect, where the trendline follows more of a power or logarithmic function whose slope flattens even more than the linear trendline shown. While not plotted in Figure 13 (given the difficulty to discern from the linear line), the logarithmic and power functions both show a slightly tighter fit and higher R^2 with our exemplar data. Additional data collection for future increments, particularly if the contractor continues to push higher planned feature counts, would clarify what relationship shape may be most explanatory.

What can be gleaned from a technical management aspect? Look for the near-term impacts of technical changes (i.e., moves and splits) as the drivers of poor performance. If a developer's plan shows an outsized amount of this activity, be skeptical of how cost and schedule will be affected. Also look for realism in diseconomies of scale, and if a developer plans to just "do more," they might accomplish some more but not as much as they hope. At a minimum collect data on how developers scale so analysts can make informed judgments when plans change.

4.3. Forecasting Application

Whose job is it to manage the iron triangle? The program manager! Ultimately, all the descriptive statistics and effects above have limited impact if they cannot be leveraged to forecast future effects. Our team proceeded to the next step to look at how prior period behavior could be used to project future feature efficiency. This type of analysis is important because any analyst applying this to an ongoing program cannot know the future, and thus would at best have data on the just-completed increment to plan one, two, or three increments into the future.

4.3.1. Correlation with Prior Periods

One way our team looked at predicting feature efficiency is by analyzing the correlation of prior increments change traffic to future increments' feature efficiency. Our exemplar data shows that the predictive power of change traffic remains strong through three increments prior (Table 1).

Table 1: Correlation of Feature Efficiency to Change Traffic from N-X Increments Prior

	N-0	N-1	N-2	N-3
R^2	0.604	0.561	0.468	0.458
Correlation	-0.78	-0.75	-0.68	-0.68

Another way to look at using prior history to project into the future is by evaluating moving averages (Figure 14). This seems to have stronger predictive power than just analyzing a single prior increment. This is likely due to the smoothing effect negating the yo-yo effects between increments partially masking an underlying trend of consistently degrading performance.

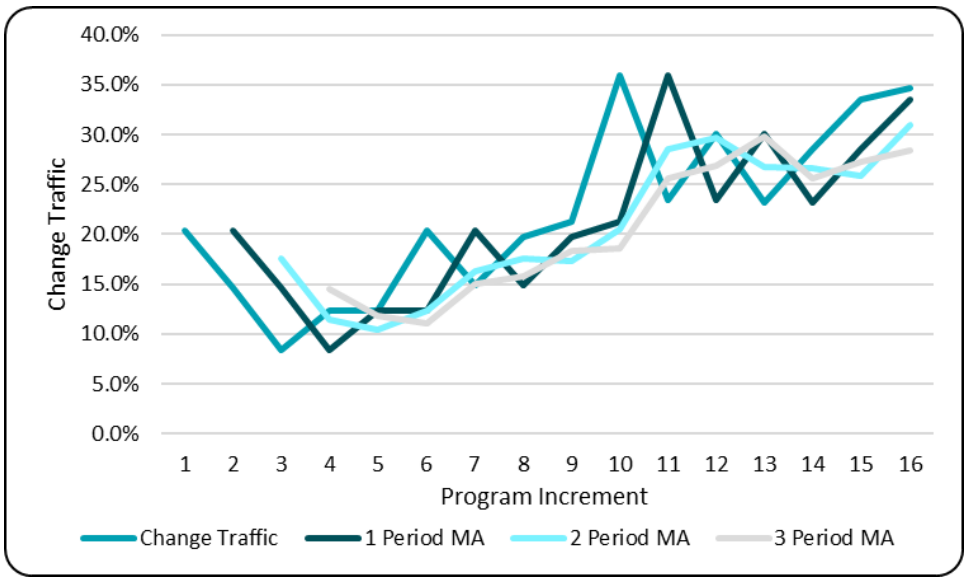

Figure 14: Change Traffic Moving Averages

The R^2 and correlation between the moving averages of prior increments is shown in Table 2 below. Comparison with straight prior period information (Table 1) shows over a 40% improvement in the two-period and three-period moving average R^2 values as compared to just the straight period look.

Table 2: Correlation of Moving Average of Change Traffic to Feature Efficiency

	0 Period	1 Period	2 Period	3 Period
R^2	0.604	0.561	0.661	0.706
Correlation	-0.78	-0.75	-0.81	-0.84

(Note that the first two columns are identical to the previous table. The '0 Period' column is the problematic same-period correlation, for benchmarking purposes only. The '1 Period' column shows that correlation with the previous period is tantamount to a one-period moving average.)

Our team also looked at whether improvements could be realized by utilizing optimized weighted averages instead of straight moving averages (Figure 15 and Table 3). Our team did this optimization by solving for the weights such that the Sum Squared Error (SSE) in the estimated residuals of Feature Efficiency is minimized. In this exemplar dataset, there is limited improvement in overall prediction abilities from straight moving averages, with only a 0.01 increase in the correlation coefficient for both the two- and

three-period moving averages. Figure 15 shows how closely the weighted and straight moving hew to each other in this dataset. This may be an artifact of this exemplar dataset and could be better refined with the collection of additional real-world data.

Table 3: Optimized Weighted Moving Average Coefficients

	0 Period	1 Period	2 Period	3 Period
N-3 Weight	N/A	N/A	N/A	0.294
N-2 Weight	N/A	N/A	0.430	0.353
N-1 Weight	N/A	N/A	0.570	0.353
R^2	0.604	0.561	0.665	0.707
Correlation	-0.78	-0.75	-0.82	-0.84

These moving average approaches could be used as inputs to a future regression model with less sensitivity to the turbulence of increment-to-increment performance. Also, moving averages may make for better dashboard-like health metrics for a CDO or data analytics group to track, and the estimating team can curate the one that has been shown to best portend development challenges. (A legitimate alternative is the MA of Feature Efficiency itself, but we are focused here on exploring drivers instead.)

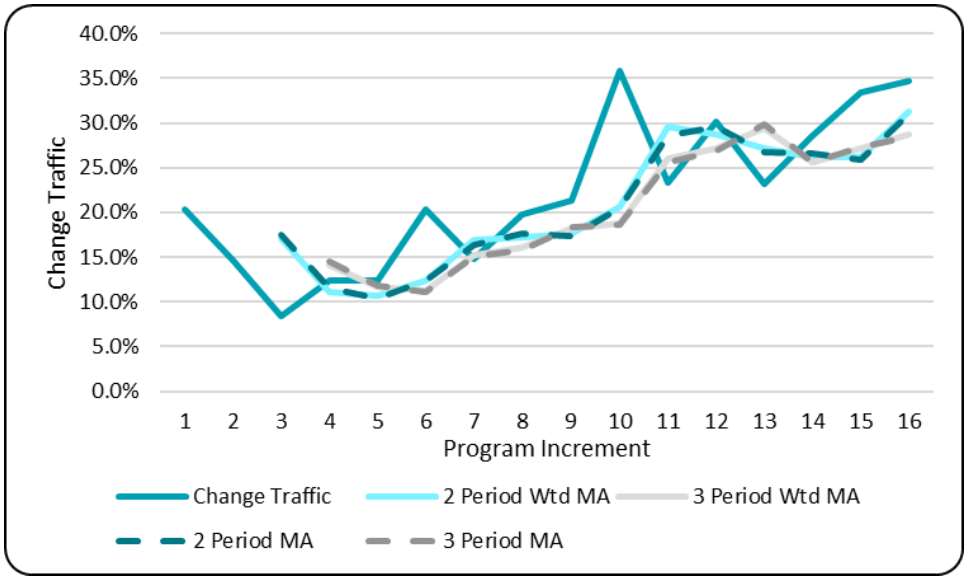

Figure 15: Change Traffic Weighted Moving Averages

4.3.2. Not all Changes are Created Equal

In addition, our team analyzed the impact of various types of change traffic on feature efficiency. For our exemplar dataset, this was done by looking at the relationship of the various types of feature changes to feature efficiency, and then making decisions about how to adjust change traffic to account for these relative effects. Recall in Equation 2 the formulation of change traffic presented, where all absolute changes were counted equally (the aforementioned naïve weighting). This was a slight oversimplification to introduce the topic, and one could consider the more complete version presented in Equation 3 below.

Equation 3: Change Traffic with Weighting

$$Change\ Traffic = \frac{\sum Features\ (\alpha \cdot Add,\ \beta \cdot Move, \gamma \cdot Split, \delta \cdot Delete)}{Planned\ Features}$$

Equation 3 removes a simplifying assumption from the introduction of feature efficiency, which implied that all feature changes had equal weight on change traffic. This is an arbitrary restriction, and it is easy to envision that some types of changes may prove more impactful on eventual development completion than others. While in actual program data there are other ways to optimize the computation of change traffic to better predict feature efficiency (e.g., sub-projects, Contract Line Item Numbers (CLINs), scrum teams), in our simplified paper and example we'll only go into the possibility of weighting feature change types[15].

To go about solving for the weights of the feature types, and deciding which should be weighted independently or not, there are two initial considerations. The correlation of each category to the eventual dependent variable (feature efficiency in this case), and the correlation between independent variables, also known as multicollinearity. Multicollinearity is important to consider because it can wreak havoc on the significance of regressed models in which multiple dependent variables carry the same explanatory

[15] This is very much like established practice in calculating Effective Source Lines of Code (ESLOC), where certain types of code (e.g., autogenerated, re-tested, re-implemented, etc.) are weighted less heavily than a wholly new developed line of code.

power. Keep in mind with time series data that this correlation analysis can take more effort, as all dimensions can be considered as well as prior periods (e.g., comparisons for the same variable for period N-1, N-2, N-3, etc.).

Table 4: Cross-Correlation of Feature Change Categories

	Add	Split	Move	Delete
Add	100%			
Split	42%	100%		
Move	31%	62%	100%	
Delete	8%	-24%	-8%	100%

It makes sense that Split and Move are most highly correlated. We hypothesized earlier that they would have the biggest disruptive effect on development, and they are also arguably to the two recourses the development team has at their disposal when needing to triage more troublesome features. Adds are likely coming at urgent customer requests and Deletes would conversely need to be confirmed as no longer needed or otherwise OBE. Moving all or Splitting part of a feature to the next PI would presumably both be used heavily when there is pressure on the capacity of the current PI.

Table 5: Select Weighted Feature Model Regression Statistics

#	Name	Functional Form	R^2	SSE	Model F
1	Unweighted Change Traffic (N-0)	FE(N) = a*CT(N)+b	0.604	0.049	21.396
2	Unweighted Change Traffic (N-1)	FE(N) =a*CT(N-1)+b	0.561	0.045	16.612
3	Unweighted Change Traffic (N-2)	FE(N) =a*CT(N-2)+b	0.468	0.036	10.559
4	Add/Remove Wtd. Change Traffic (N-1)	FE(N)=(α*Add+β*(Split+Move+Delete))/Planned+b	0.734	0.058	16.564
5	Add/Remove Wtd. Change Traffic (N-2)	FE(N)=(α*Add+β*(Split+Move+Delete))/Planned+b	0.748	0.058	16.337
6	Delete Wtd. Change Traffic (N-1)	FE(N)=(α*Delete+β*(Split+Move+Add))/Planned+b	0.672	0.054	12.317
7	Move Wtd. Change Traffic (N-1)	FE(N)=(α*Move+β*(Split+Add+Delete))/Planned+b	0.683	0.054	12.903

Based on the multicollinearity present between many of the variables in Table 4, it is expected that a simple linear regression to solve for coefficients α through δ would result in unstable coefficients and a less significant model. This was borne out by our analysis and not worth showing the results of in this paper. However, just because one option doesn't work, that doesn't mean alternatives to improve predictability cannot be found. Table 5 above shows a select set of combinations of regression models our team attempted to highlight potential ways to improve forecasting and key statistics.

While no model achieves as strong performance as Model 1 (which assumes perfect knowledge of current period change traffic), Models 4 and 5 provide superior R^2 and F scores over Model 2. However, Models 6 and 7, which attempted to weight one of the removal feature change types different than the other two, resulted in less significant models and appear less likely to be candidates for a better predictive model. This is likely attributed to those previously discussed multicollinearity effects and is being investigated further in real program data.

Figure 16 below shows a graphical comparison of Models 2 and 4 from Table 5 and how their predictions of feature efficiency vary over the range. One can quickly see the tighter dispersion in predictions, especially at the lower end of the range from Model 4 as compared to Model 2. This improved predictive model cuts the SSE by almost 50% and increases the R^2 from 0.56 to 0.74, with just one extra regressed variable.

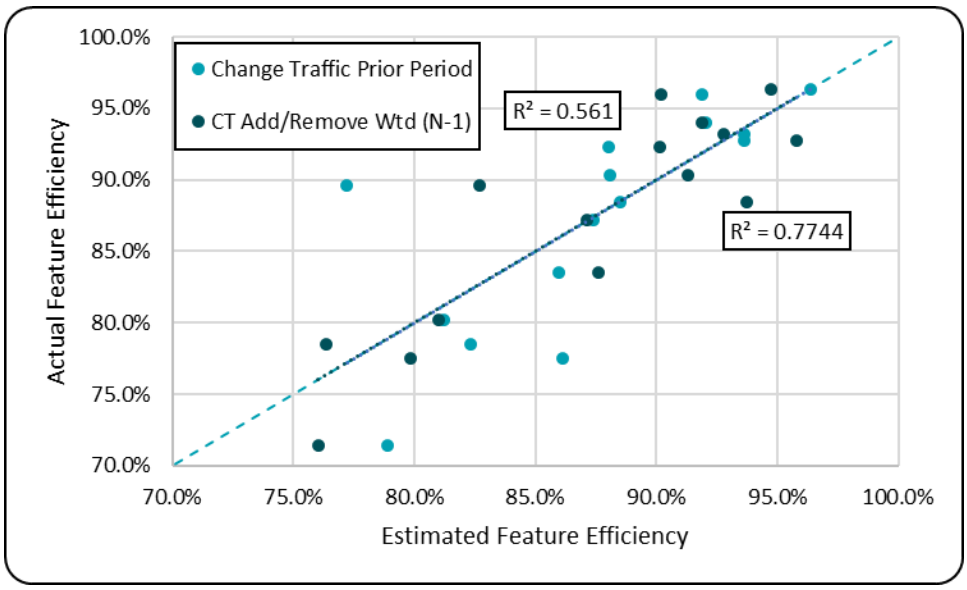

Figure 16: Comparison of Actual vs. Estimated Feature Efficiency

While our team is unable to comment on which exact models prove most applicable to our actual program data, the comparisons like those shown in the Table 5 above and decision tradeoffs discussed in this section are the same as in the real world. There are no one-size-fits-all models to solve this problem and all comparative models come with tradeoffs in data recency and considerations about applicability. Ultimately the best method is supported by statistics, aligned with SME judgment, and able to be confidently brought in front of the PMO.

4.4. Change Traffic as a Driver

After showing the detailed analysis of the ability for Change Traffic to predict Feature Efficiency in our exemplar dataset, it makes sense to zoom back out and talk about how others can leverage Change Traffic for their own purposes. Ultimately readers should be left with an understanding of how Change Traffic, given our careful treatment of it, is eminently suitable as a parametric driver for Feature Efficiency because it is:

Knowable: While the Change Traffic for the increment being estimated (PI *N*) is not known until after the fact, the metric can easily be calculated for all previous increments, with the preceding increment (PI *N-1*) being just-in-time.

Measurable: Change Traffic is an objective measure, dependent entirely on feature counts. Some refinements may include expert-based or statistically-derived weight coefficients, but it still traces directly back to observable quantities.

Controllable: While Change Traffic cannot be precisely specified in advance, there are elements of it under the control of the PMO and the developer working in concert. The former may work to reduce (or at least track) disruptive Adds and Deletes, reserving them for requirements that are urgent or OBE, respectively. The latter may work to optimize productivity and capacity to minimize the number of Splits and Moves. Not only will these lead to tautological improvements in Feature Efficiency in the current period, but evidence shows they will yield dividends in curtailing or reversing the bow wave across future PIs as well.

Immediate: Unlike SLOC for software scope, for example, Change Traffic is not a distant proxy but rather a direct reflection of the hypothesized causal variable, the amount of "churn" injected into the development process during a PI.

Intuitive: In fact, the volatility of external requirements on programs is what originally led us to the general notion of churn and the specific parameter of Change Traffic to capture it. Even where lanes are fairly well defined, there are continually new lanes entering the highway and existing lanes being widened or narrowed, or exiting altogether.

5. Conclusion

5.1. Application of Results

Our results offer a comprehensive framework and practical insights that allow for cost estimators to address multiple analytic needs within their organization. For those with a broader perspective, analyzing general trends of change traffic across a portfolio of development efforts allows for the accurate assessment of budgets and schedules based on real-world behavior. If budget constraints exist, analysts can use feature inefficiency to project a lengthening schedule, while constraints on schedule can prompt data-driven assessments of additional resources needed to get back on track. There are also portfolio

health considerations that could be modeled if metrics are tracked consistently across an organization.

At the more detailed level this analysis can be replicated within a PMO or smaller portfolio to directly support the PMO in the quarterly PI evaluations. This analysis enables tracking project health over time and challenges developer performance and proposals.

Key takeaways from this analysis include:

There is a clear relationship between change traffic and feature efficiency that persists over time. Similar to EVM, rectifying poor performance in a sprint is not an immediate process, and analysis of both current and cumulative metrics should be considered.

Do not discard the foundations of program management. Baselining projections to a plan remains crucial, and the program discussed in this paper could have had a more realistic baseline if this recommendation had been heeded. While an agile development effort may be willing to adjust priorities of development to meet user needs, the expectation is not that there is a significant decrease in overall productivity from the baseline plan.

An ounce of prevention is worth a pound of cure. Manage the program proactively into the future. It is clear the more changes that occur during an increment, the larger the negative impact on productivity going forward. This underscores the need for developers and customers to spend more time scrutinizing and re-prioritizing requirements for at least increment N+1, and maybe further out depending on how stable the program is.

Understand the impact of changes. Customers should be skeptical of developers' incorporating changes to development plans without margin or tradeoffs in the development schedule/scope elsewhere. Developers should understand how their planning and execution metrics are affected by the consideration of feature efficiency.

5.2. The Importance of Policy

While the results shown in the paper focus on exemplar data that should not be difficult to “collect,” much work must be undertaken to normalize the collected technical data and align the financial and technical baselines in order to manage agile programs effectively. These challenges in collection are directly antithetical to current DoD policy on implementation and promotion of metrics for all to see (Office of the Under Secretary of Defense for Acquisition and Sustainment, 2020). The technical data, which is the focus of this paper, often is not even a required deliverable in any analytic format, much less something to be delivered to independent estimators!

Our team first identified potential data sources by attending PIPE sessions and reviewing PowerPoint slides – transcribing information by hand from various static sources into one cohesive dataset. With the utility of the data established, our team was able to find friendlier data sources, spanning government and contractor agile management systems (e.g., Jira) and technical data CDRLs. Ultimately, cost estimators remain at the mercy of PMOs’ deciding what data to collect from developers. For example, in many instances, previous EVM reporting requirements have been waived. When Cost and EVM CDRLs are lacking, we often have to rely on invoices, contract data (at the CLIN level), or staffing levels reported in management briefs. While better than nothing, these sources often lack essential features such as bucketing of costs via a standard work breakdown structure (WBS).

This highlights the vital importance of overarching data collection policies. Leaving data sharing decisions to PMs leads to gaps in data when independent estimates and the associated data sharing may not be required, reduced ability to comprehend data due to inconsistencies or interpretability when data sharing is required, and data inconsistencies that reduce utility and hinder future analysis across a larger population or the development of parametric methods.

If policy dictated concurrent deliverables and required allocation/alignment of costs by technical scope on the developer side, this would result in data that are more readily useful for analysis and more programs’ worth of data with which to make more general pronouncements and methods with broader applications.

From our team's engagement with PMOs, ***there is a negative feedback loop when it comes to the perceived value of data collection requirements and the ability to inform project management activities.*** When there is no data, it's nigh impossible to demonstrate its utility, which in turn makes it hard to overcome the inertia of not engaging in data collection. An exacerbating factor is that many project management offices do not appear to have the staff in-house to review and provided decision-worthy analysis like that which is presented here. Without that demand signal or understanding of potential, it's easy to see why program offices might be enticed to remove reporting requirements or reticent to add them in the first place. It takes sustained investment from a centralized group with analytical skills to "kick in some doors," and willingness from within the PMO to listen, brief the metrics, and make the progress shown in the analysis of this paper. ***With a determined effort, we can turn the negative feedback loop into a positive one: demonstrated value whets the appetite for more data collection, and more robust data supports more insightful, credible, and actionable analysis.***

5.3. Untapped Potential

The authors do not want to present this analysis as any sort of "victory lap," and we acknowledge there is much more work to be done (see Section 6). This includes both the depth of analysis for those programs leaning into collecting the data and metrics discussed in Section 4 and the breadth of application across a larger population of software acquisitions.

There is also the missing link between the production of relevant analysis that provides insights into the health and the effect on project planning. The authors believe that the metrics tracking and analysis presented here provides similar benefits to well-established EVM practices and metrics but better conforms to the framework of agile software development. Benefits of actively monitoring and acting upon our analysis include:

- Ability to inform enterprise-wide metrics collections and visualizations via CDOs.
- More effective planning of project scope. Less bow wave, less time spent replanning.
- Inform PMOs about the need to plan further out than just one increment at a time.

- Inform project management about the potential of not fully subscribing development hours and minimizing the deferred work (adding to the backlog).

We also want to acknowledge that there is an opportunity for cost estimators to leverage, rather than eschew, the dashboards many agile development focused program offices are implementing. While generally these dashboards are ephemeral and focused on user delivery and not focused enough on development progress, that does not mean the information cannot be tied to the needs of estimators at the program office or independent level. By writing scripts to scrape key information from the dashboards and save to a repository as a "track record" of sorts, estimators can mitigate the burying of past data with the most recent. Analysts can also use this historical track record to look for correlations (or lack thereof) between user metrics and development behavior, which may allow for estimates to resonate further with the PMO. Even when metrics of interest aren't shown on the dashboards themselves, estimators can often directly access the back-end data systems used to feed the dashboards instead, via export or predetermined interface.

Lastly, our team recognizes the potential of mitigating negative trends in the first place by implementing contracting strategies to incentivize contractors to plan accordingly and providing guard rails to protect budgets from consistent growth. The general belief that the agile software development budget is fixed is not always reflected in contract space (Kosmakos & Brown, 2022), making it more likely that the government will be on the hook for cost growth when inevitable technical execution challenges arise.

6. Next Steps and Future Research

Many of the conclusions in this paper come from analysis across a few programs, and not a broader population of acquisitions. Further monitoring and research is needed to see how much of the behavior exhibited in this notional example is seen in the broader development community and whether there are any lurking variables yet to be studied.

Our team is also interested in disentangling the conundrum of whether poor productivity drives change traffic, vice versa, or some of both. We believe this challenge could be resolved with more active tracking and measurement of development progress within increments. Weekly level sprint information and better documented context about why

content was being added/adjusted would inform the source versus symptoms argument. Technomics has begun analyzing the PIPE notes in our actual program data to do this. These notes document *which* features in our real-world data are added/changed/deleted/split, but not *why*, and it is not yet clear if current PMO processes provide enough of a description to inform an analytical determination about the source versus symptom.

Another dimension of analysis yet to be fully explored with our real-world datasets are what kinds of features are changing most often or if there's certain type of software development that is most affected. Our programs comprise numerous "lanes" of effort, and there may be more ability to home in on a certain capability, piece of technical scope, or specific interface to be actively managed within or across them. Further research would be needed to understand if there are more challenging functionality to plan from a capability perspective and/or areas to just expect more change management. This could also extend to the purpose of changes. It's one thing to say you added X or deleted Y… but *why* were those things done? Was a feature added because a customer driven change in requirements, or by poor planning in managing the scale of a given story? Understanding the "why" is key to making management decisions to mitigate the inefficiency discussed in this paper, vice just providing headroom to account for the effects.

There is a myriad of other questions that our team raised when performing this research, and a few further areas for considered study are listed below:

- Could more advanced regression techniques (i.e., ridge regression) be used to resolve the multicollinearity conundrum explored in Section 4.3.2.
- Do developers on other acquisitions already do this sort of tracking? Do they use other terminology than what we've ascribed?
- Is there a primary or secondary correlation of team size or number of agile teams that helps explain time trending. Is it inevitable that larger teams are just more inefficient?
- Is this research extensible to other agile software effort measurement types (Function Points, Simplified Function Points, Interface Counts, etc.)?

- Could SME-derived weights be used to inform a more accurate change traffic measure? This would save regression degrees of freedom and possibly receive more buy-in from PMO staff.

We look forward to presenting this work and discussing it with the community at the workshop to glean new ideas and perspectives.

7. References

Abumrad, J., & Krulwich, R. (2018). Smarty plants. [Audio podcast episode]. In Radiolab. Retrieved from https://www.wnycstudios.org/podcasts/radiolab/articles/smarty-plants

(2023). *AGILE ASSESSMENT GUIDE Best Practices for Agile Adoption and Implementation.* Retrieved from https://www.gao.gov/assets/d24105506.pdf

Apel, S., & Kästner, C. (2009). An Overview of Feature-Oriented Software. *Journal of Object Technology*, 1-36.

Australian Bureau of Statistics. (2017). Retrieved from Childhood education and care (No. 4402.0): https://www.abs.gov.au/AUSSTATS/abs@.nsf/Lookup/4402.0Main+Features1June%202017?OpenDocument

Brady, S. P. (2021, October 6). *Let's Talk Agile: Take 2 with Sean Brady – Software Pathway Policy Changes.* Retrieved from Defense Acquisition University (DAU): https://www.dau.edu/events/lets-talk-agile-take-2-sean-brady-software-pathway-policy-changes

Braxton, P. J., Brown, D. H., Rhodes, K. S., & Wekluk, R. A. (2022). Uncertainty of Expert Judgment in Agile Software Sizing. *ICEAA Professional Development & Training Workshop.* Pittsburgh, PA.

Braxton, P., Brown, D., Rhodes, K., & Wekluk, A. (n.d.). *Bridging the Gap: Leveraging Micro Agile Data in Macro Planning Estimates.* Joint IT/Software Cost Forum.

Dekkers, C., & French, D. B. (2018). Using Function Points to Manage Agile Product Backlog: Fact vs. Fiction. *ICEAA Professional Development & Training Workshop.*

Dr. Nicole Forsgren, D. D. (2019). *State of DevOps 2019.* DevOps Research and Assessment (DORA).

Draheim, P. H. (2022, December). *NAVAIR SRDR DB: NAVAIR Software Data Compilation.* Retrieved from Cost Assessment Data Enterprise (CADE): https://cade.osd.mil/

Hoffman, M., Davis, K., Ashwood, E., Smith, A., & Lane, W. (2019). Case Study: Agile Execution Environment Analysis (AE2A) in AF DCGS. *ICEAA Professional Development & Training Workshop.*

International Cost Estimating and Analysis Association. (2020, March 17). CEBoK 2.0. Retrieved from Cost Estimating Body of Knowledge 2.0 Wiki: https://wikidev.iceaaonline.com/wiki/Main_Page

Kosmakos, C., & Brown, D. (2022). *What Does Agile Software Development Need? Predictable Cost or Predictable Outcomes?* Retrieved from https://www.iceaaonline.com/wp-content/uploads/2022/06/SA03-Kosmakos-What-Does-Agile-Software-Development-Need.pdf

Krempasky, R., Rhodes, K., & Wekluk, A. (n.d.). *Cloud Cost Estimating & Optimization Framework.* Joint IT Software Cost Forum.

Long, C. D. (2022). *Software Development Phasing & Schedule Growth Analysis.* ICEAA. Retrieved from https://www.iceaaonline.com/wp-content/uploads/2022/06/SA10-Long-Software-Phasing-Schedule-Growth.pdf

McQuade, J. M., Murray, R. M., Louie, G., Medin, M., Pahlka, J., & Stephens, T. (2019). *Software Is Never Done: Refactoring the Acquisition Code for Competitive Advantage.* Defense Innovation Board.

Office of the Under Secretary of Defense for Acquisition and Sustainment. (2020). *DOD INSTRUCTION 5000.87 OPERATION OF THE SOFTWARE ACQUISITION PATHWAY.*

Scaled Agile, Inc. (2024, January 24). *How the Scaled Agile Framework® Benefits Organizations*. Retrieved from Scaled Agile Framework: https://scaledagile.com/what-is-safe/scaled-agile-benefits/

Smallwood, B. (2018). *Agile Development Cost Factors Case Study.* ICEAA. Retrieved from https://www.iceaaonline.com/wp-content/uploads/2018/07/AG01-Agile-Software-Development-Cost-Factors-Smallwood.pdf

Southworth, R., Hunt, B., Lucas, C., & Sanchez, E. (2023). *Applying Simple Function Point Analysis to an 804 Rapid Acquisition Program Cost.* ICEAA. Retrieved from https://www.iceaaonline.com/wp-content/uploads/2023/06/IT02-Hunt-Applying-Simple-Function-Point-Analysis-to-an-804-ppt.pdf

The Standish Group. (2015). *2015 CHAOS Report*. Retrieved from https://standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf

8. Exemplar Dataset

PI	Planned	Actuals	Shortfall	Added	Removed	Split	Moved	Deleted	Change Traffic
1	245	225	20	15	35	19	7	9	50
2	260	240	20	9	29	21	0	8	38
3	250	235	15	3	18	3	5	10	21
4	275	265	10	12	22	4	11	7	34
5	275	255	20	7	27	0	25	2	34
6	295	275	20	20	40	10	19	11	60
7	310	280	30	8	38	5	3	30	46
8	370	355	15	29	44	29	13	2	73
9	390	345	45	19	64	31	22	11	83
10	390	340	50	45	95	33	45	17	140
11	385	345	40	25	65	40	25	0	90
12	425	355	70	29	99	8	74	17	128
13	480	385	95	8	103	28	70	5	111
14	490	380	110	15	125	45	70	10	140
15	535	420	115	32	147	77	60	10	179
16	525	375	150	16	166	73	88	5	182

9. Derivations

These two short sections illustrate some relationships between our metrics of interest and the features counts that feed them.

9.1. Tautological (Recursive) Regression

First, we can define Shortfall as the difference between Planned and Actuals, so that

$$Actual\ Features = Planned\ Features - Shortfall$$

where

$$Shortfall = Removed\ Features - Added\ Features$$

We are generally assuming that Shortfall is positive (which is true across our notional data set), in which case we deliver fewer Actual Features than Planned. This also entails that the number of Removed Features (via Splits, Moves, or Deletes) exceed the number of Added Features. However, all results are still valid if Shortfall is zero (we hit our Plan exactly) or negative (we deliver a surplus of features).

Dividing both sides of the first equation by Planned Features, we have

$$\frac{Actual\ Features}{Planned\ Features} = 1 - \frac{Removed\ Features - Added\ Features}{Planned\ Features}$$

But the left-hand side is just our definition of Feature Efficiency, and the fraction on the right-hand side is how we might define Net Change Traffic.

$$Feature\ Efficiency = 1 - Net\ Change\ Traffic$$

This is the tautological regression shown in Figure 7. Recall that this Change Traffic definition was unsatisfactory, since in practice large numbers of Adds and Deletes don't "cancel out," they each perturb the system in their own way.

9.2. Change Traffic Counts and PI Shortfall

Expanding the components of Deleted Features from above, we have

$$Shortfall = Remove - Add = (Split + Delete + Move - Add)$$

Again, the Shortfall is a signed difference, but Change Traffic is an absolute sum.

$$Change\ Traffic\ Count\ =\ Split\ +\ Delete\ +\ Move\ +\ Add$$

This unweighted sum is the numerator of our naïve Change Traffic formulation.

Subtracting the two, we have

$$\begin{aligned} &Change\ Traffic\ Count\ -\ Shortfall \\ &\quad = (\ Split + Delete + Move + Add) - (Split + Delete + Move - Add) \\ &\quad = \ 2 \cdot Add \end{aligned}$$

Thus, the gap between the Change Traffic and Shortfall series in Figure 11 is precisely twice the number of Added Features in each PI.

10. Glossary of Terms

Term	Definition
Change Traffic	The measure of magnitude of changing requirements, including both those added and deleted, as a ratio of the total plan or actuals completed.
Diseconomies of scale	Diseconomies of scale occurs when a company or business grows so large that the costs per unit increase.
Extrapolation from Actuals	Extrapolation from actuals uses actual costs from past or current items to predict future costs for the same item.
Feature	A unit of functionality of a software system that satisfies a requirement, represents a design decision, and provides a potential configuration option
Feature Efficiency	The agile software developer's version of performance index. (Include equation?)
Function Point	Function points measure the size of system based on the functional view . Size is determined by counting the number of inputs, outputs, queries, internal files and external files in the system and adjusting that total for the functional complexity of the system.
Incremental	The incremental development (methodology) is a refinement to the waterfall process in that software is built in increments of functionality, in overlapping series.
Iron Triangle	Refers to the three key constraints that can affect a project: cost, schedule, technical.
Spiral	A SW development method combining elements of Waterfall and Agile. Includes version releases and concurrent documentation and testing. However, like the

	agile method, it provides incremental, staged improvements in functionality.
T-shirt Sizing	A SW sizing approach primarily used in Agile wherein SMEs estimate the amount of effort a given item will take by picking pre-defined 'sizes' of T-shirt, often where each size doubles from the previous in effort.
Waterfall	The waterfall process is the traditional lifecycle development (methodology) that models a conventional software engineering cycle. A single effort.

11. List of Acronyms

Acronym	Expansion
ASoTs	Actual Authoritative Sources of Truth
CDO	Chief Data Officer
CDRL	Contract Data Requirements List
CER	Cost Estimating Relationship
CLIN	Contract Line-Item Numbers
CPI	Cost Performance Index
CSDR	Cost and Software Data Reporting
DoD	Department of Defense
DoDI	Department of Defense Instruction
DORA	DevOps Research Association
EER	Effort Estimating Relationship
EVM	Earned Value Management
FFP	Firm Fixed Price
FOSD	Feature-oriented Software Development
LOE	Level of Effort
MA	Moving Average
OBE	Overcome By Events
OPTEMPO	Operational Tempo
PI	Program Increment
PIPE	Program Increment Planning Event
PISE	Program Increment Summary Event
PMO	Program Manager (or Management)
PMO	Program Management Office
RCA	Root Cause Analysis
ROC	Return on Cost
ROS	Return on Sales
RWPE	Rolling Wave Planning Event
SAFe	Scaled Agile Framework
SEITPM	Systems Engineering, Integration and Test, and Program Management

SLOC	Source Lines of Code
SME	Subject Matter Expert
SRDR	Software Resources Data Reports
SSE	Sum Squared Error
SW	Software
SWAP	Software Acquisition Pathway
TCPI	To-complete Cost Performance Index
TEM	Technical Exchange Meeting
USG	United States Government
WBS	Work Breakdown Structure

12. Author Biographies

12.1. Aubrey Dial

Aubrey Dial is an ICEAA CCE/A certified analyst and Employee Owner at Technomics, Inc. She has over 6 years of experience performing cost analysis, data manipulation, estimate development, acquisition decision support, and program execution support across several DoD customers. Currently, Ms. Dial is responsible for developing life-cycle cost estimates and developing data workflows/analysis for software and IT programs. Aubrey is a 2023 ICEAA Washington Chapter Junior Analyst of the Year winner. She holds a BS in Mathematics and a MS in Data Analysis and Applied Statistics from Virginia Tech.

12.2. Benjamin Truskin

Ben is an ICEAA CCE/A Service Area Manager and Employee Owner at Technomics, Inc. with 11 years' experience providing data-driven decisions to leaders. He has provided support in the cost community to Defense, Civilian, and Intelligence customers across a range of commodities. Ben's experience includes: cost estimation and phasing, data policy development, data collection and normalization, methods development, source selection support, industry interface, database requirements development, data visualization. He received his Master's and Bachelor's in Aerospace Engineering from Penn State University.

12.3. Ken Rhodes

Ken Rhodes is an ICEAA CCE/A, Project Manager, and Employee Owner at Technomics, Inc. He has over sixteen years of experience performing cost analysis and acquisition decision support for DoD customers. Currently, Mr. Rhodes develops life-cycle cost estimates, cost / price assessments, and data visualization products for software and IT programs. He holds a BS in Industrial and Systems Engineering and MS in Systems Engineering from Virginia Tech.

12.4. Peter Braxton

Peter Braxton is a Subject Matter Expert and Employee Owner at Technomics, Inc. He has over 20 years of experience performing cost and risk analysis and delivering associated training for a broad spectrum of federal government clients. The inaugural VP for Professional Development and a multiple ICEAA Educator of the Year winner, he has shown a long-standing commitment to knowledge sharing within the community. He holds an AB in Mathematics from Princeton and an MS in Operations Research from the College of William and Mary. He is a semi-retired game show contestant and avid cruciverbalist.