D E P A R T M E N T O F D E F E N S E

Agile Metrics Guide

Strategy Considerations and Sample Metrics for Agile Development Solutions

Version 1.2

11 November 2020

OUSD(A&S)

This page intentionally left blank.

Table of Contents

List of Figures

List of Tables

No table of figures entries found.

1 Executive Summary

The Office of the Under Secretary of Defense for Acquisition and Sustainment (OUSD(A&S)) provides policy and governance for the Department of Defense (DoD) and is responsible for executing FY18 873 and 874 Agile pilots. In support of this mission OUSD(A&S) is releasing a series of documents related to Agile adoption that address the challenges and complexities faced by programs that are transitioning from traditional, waterfall project management and execution to Agile practices. This guide is one of the documents OUSD(A&S) developed.

This document offers guidance and recommendations related to actionable metrics for Agile products and services. It provides Agile development teams, testing teams, and program management teams with an understanding of how Agile metrics differ from metrics used in traditional waterfall approaches, and identifies key Agile metrics, their benefits, and drawbacks (noted as challenges). Specifically, the document presents guidance on the types of metrics available when executing Agile projects, the benefits of each metric, and actions that can be taken as a result of the data being shown. It covers the following types of Agile metrics:

- Process Metrics – Related to efficiency of processes
- Quality Metrics – Related to quality of work performed
- Agile Capability Delivery Metrics – Related to value delivery to users
- DevSecOps Metrics – Related to efficiency of the value delivery pipeline
- Cost Metrics – Related to Agile cost measures
- Value Assessment Metrics – Related to evaluating the impact of the work.

Programs should consider this guide as a starting point, and tailor the metrics for the unique considerations of the program. Programs should also identify stakeholders for each metric and the decisions that the metric supports.

Although most of the guidance can still apply to embedded systems and hardware implementations using Agile, the document focuses primarily on software development efforts.

2 Purpose

The DoD is working with Government and industry leaders to modify how DoD plans and delivers software products and services. Most projects today continue to use a waterfall delivery approach. The increasing complexity of systems, decentralization of tools and technologies, and rapid pace of change pose challenges in delivering meaningful, secure, and modern capabilities that meet user expectations on time and within budget. Addressing these issues requires a significant change in approach to planning and delivery of capability to the warfighter: specifically, a shift to Agile project management and delivery.

Transitioning from a waterfall to an Agile approach represents a true paradigm shift. It has impacts at all levels of the organization, including programs, projects, and enabling technologies. Measurements are key to providing the visibility and transparency of information necessary to determine whether the program is moving in the right direction. Programs must have visibility into the work being performed and the results of the work. The increased visibility will allow the Program to make decisions based on data.

This guide provides Agile teams and Project/Program Managers (PMs) with a collection of Agile metrics to measure progress and support decisions in the areas of process, quality, product, DevSecOps, cost and value delivery. Each area (or category) contains a number of metrics to consider.

Programs should tailor the metrics to fit their needs, based on their experience with Agile and DevSecOps practices and toolsets, the Agile framework, and project-specific factors (e.g., data availability for reporting). Programs should ensure that each selected metric is aligned to a stakeholder to provide the information necessary to support specific decisions.

This guide assumes that the reader has a general understanding of Agile practices – specifically Agile Scrum and Kanban practices. For additional information on Agile, see the Agile 101 - Agile Primer also published by OUSD(A&S).

3 Scope

This guide presents a curated list of core Agile metrics for Agile teams and PMs. It explains each metric's purpose, benefits, and challenges. It also provides context for each metric and offers some examples to clarify specific points related to visualization of data as it pertains to the metric.

Teams and PMs should understand the principles upon which Agile is based, as some of the metrics are related to Lean concepts such as flow, feedback and continuous improvement/ learning. The appendices to this document contain additional background on popular Agile frameworks and terminology.

As noted previously, this guide is not intended to be used as a monolithic template for all Agile projects. Every program is different, so not all metrics will apply to every program. The metrics should provide the program with usable information to support decision-making. Given that each metric takes time and resources to frame, collect, observe, analyze, and report, the program should consider and weigh the cost and benefit of each metric. The key to a successful metrics strategy is to identify the right set of metrics that provide the information necessary to act, while minimizing the collection and analysis burden on the team. Ultimately, the program should seek to automate metrics collection as much as possible.

4 The Importance of Flow

Flow, measured primarily by three metrics (Work in Progress (WIP), cycle time, and throughput), is a Lean concept that permeates this guide. Flow helps project teams understand process predictability. This is important because process predictability provides project teams with the ability to measure the time it takes to complete work (cycle time). A lack of process predictability can manifest as bottlenecks in the work taking place (or WIP). These bottlenecks can lead to a buildup of queues somewhere in the process and a lack of predictability as to when users can expect work to be completed.

Key indicators of a lack of predictability include:

- Starting work before previous work is completed (context switching)
- Inserting expedited new requests that interrupt the flow of work
- Adding new scope or acceptance criteria to existing work (maybe because it is easier to insert new work than to define a new story) [1]
- Lack of a clear and continuous elaboration process for defining work, which leads to incomplete work definition (in Scrum, for example, this manifests as work that is not clearly defined with a definition of done and acceptance criteria)
- Continual overtime and heroics that lead to burnout and quality issues.

[1] Source: Vacanti, D. (2015). *Actionable agile metrics for predictability*. Daniel S. Vacanti, Inc.

Ultimately, these indicators manifest as bottlenecks, which lead to increased and less predictable cycle time and, as a result, impact throughput – or the metric that explains how much work can be done in a given amount of time.

The key flow metrics are defined as:

- WIP – Total number of items currently being worked on. Kanban boards are designed to provide transparency and visibility into WIP. The Agile 101 – Agile Primer document published by OUSD(A&S) contains more information on Kanban boards.
- Cycle Time – The flow metric that represents the amount of time it takes for work to complete.
- Throughput – The flow metric that represents how much work completes per unit of time.

Flow metrics can be used on multiple units of measure. For example, flow metrics can measure the throughput, cycle time, or WIP of specific business value units such as requirements, tasks, epics, features, capabilities, and releases. They can also be used to measure level of effort units such as story points and hours of work.

Why Are Flow Metrics Important?

Flow metrics use the language of the customer.[2] It is easy for users (representing the business) to understand the language of flow:

- WIP – How many tasks is my team currently working on?
- Cycle Time – How long will it take to complete a task?
- Throughput – How much work can my team perform at any given moment in time?

More important, reports showing standard Agile metrics involving story points, velocity, and other related metrics do not necessarily convey their meaning well to users. Standard Agile metrics are intended mostly to help the development team and Product Owner to plan and measure work. The user, on the other hand, might not understand how to interpret the results of standard Agile metrics such as velocity. Ultimately, users are interested in knowing how long it will take to receive the capabilities or features they desire. For these reasons, when reporting performance results, consider translating the results into *flow-based language* to simplify communication of the pace of progress. This guide identifies some of the common Agile process metrics but whenever possible makes connections to flow (the user-friendly explanation of how long it will take to deliver work products).

[2] Source: Vacanti, D. (2015). *Actionable agile metrics for predictability*. Daniel S. Vacanti, Inc.

5 Agile Metrics

5.1 Agile Process Metrics

Agile process metrics measure process performance, or how well planning, execution, and delivery activities are performing. These flow-based metrics can be analyzed to uncover flow-related issues such as bottlenecks in the value delivery process.

5.1.1 *Story Points*

The Agile team uses story points to perform relative sizing of stories. Story points measure the complexity of a story and are explained in the next subsection. The developer assigned to a story is responsible for identifying how much effort is required to complete the work in a given sprint. Based on the team's sprint cycle length, minus overhead and time off, the team builds an understanding of the number of story points it can complete in a given sprint. Over time efficiencies are developed among the team in estimation of story points and estimation tends to improve.

Benefits
Story points are the building block units of measure that the team uses to estimate the complexity (or size) of stories in a sprint. The sizing allows the team to determine the total amount of work that it can accomplish in a given sprint or release.

Challenges
Story points cannot be compared across teams because every team level-sets its own relative sizes of work based on the capabilities of the team.

Additional Context
Sizing can be performed using multiple approaches (e.g., Planning poker; t-shirt sizing), and multiple sizing models (e.g., Fibonacci numbering; straight numbering). The important takeaway is that the team must agree on a single approach and sizing model for consistency.

5.1.2 *Velocity*

Velocity measures the amount of work (usually in story points, although other measurement units can be used as well, such as hours) that the team completes in a given sprint. It is derived by summing the total story points of all the completed user stories in a given sprint. As an example, Figure 1 depicts planned and completed story points across a seven-sprint window of time.

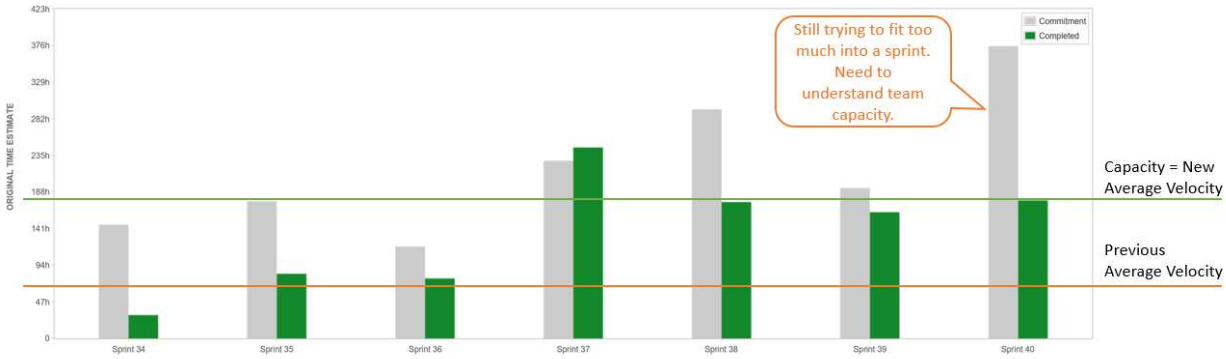

Figure 1 - Velocity Chart of Planned vs. Completed Story Points

Benefits

Velocity is a useful tool for Agile teams to determine how much work they can complete in a sprint. It serves as a benchmark for estimating and planning work in future sprints by using previous sprint velocity to predict the capacity of future sprints.

Challenges

As noted, most velocity and velocity-based metrics use story points. Since story points cannot be transferred or compared across teams, neither can velocity or velocity-based metrics. Additionally, to prevent team padding of story points, the program should not use metrics that are based on story points as a measure of team performance.

Additional Context

Planned velocity (or capacity; see below) is usually based on historic precedents or previously completed sprints. In the first sprint, the Agile team agrees on an estimated number of story points as a starting point based on what the team thinks it can accomplish. After the first sprint is complete and velocity metrics are collected, the team can adjust based on actual velocity data. Over time, the team converges on an average velocity, assuming there are no major flow disruptions that could cause wide variations in velocity (velocity variance; see Section 5.1.3).

Variations

- Capacity – Refers to the total number of story points that can be completed per sprint. It is usually based on previous team velocity metrics. This is an important planning tool at the team level. If the team, over time, understands that it is completing 30 story points per sprint on average, then the team can plan its work accordingly in future sprints.
- Planned Velocity – Is also synonymous with capacity in some frameworks. This is the team's velocity limit for a given sprint, or how much work the team estimates it can complete.

5.1.3 *Velocity Variance*

Velocity variance is the standard deviation from average velocity – or the difference from the mean velocity. As an example, Figure 2 depicts velocity variance over a seven-sprint window of time.

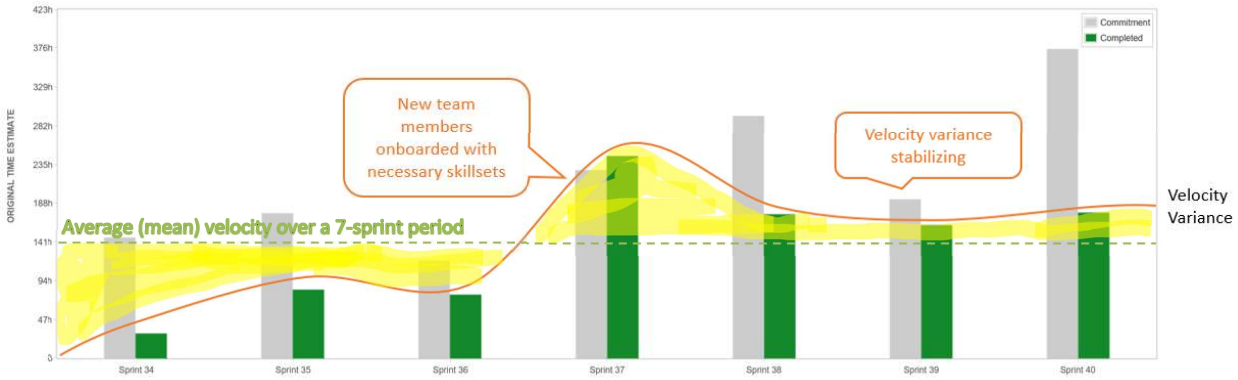

Figure 2 - Velocity Variance over a Seven-sprint Window of Time

Benefits

Velocity variance provides insight into predictability and data points to inform conversations on root causes of the fluctuations. As noted earlier, predictability is a flow objective because it enables the Agile team to generate accurate estimates of lead time (or the time it takes to deliver a feature or capability). Ultimately, users want to know how long it will take to see their requested change(s) in production, and predictability will help the team provide better estimates to the users.

Challenges

Velocity variance is based on story points and, therefore, cannot be compared across projects or teams. If the variance metrics are converted to percentage variance metrics, comparisons across projects and teams might be possible.

Additional Context

Consider reviewing velocity variance as part of each retrospective. Outlier sprints that exceed the acceptable variance thresholds should be reviewed for root cause to understand what caused the variation.

Variations

- Release variance is similar in concept to velocity variance, except the scope is at the release level.

5.1.4 *Velocity Predictability*

Velocity predictability is measured as the difference between planned and completed velocity, which is the difference between the total planned and completed story points for a given sprint. Figure 3 depicts velocity predictability and the associated gap.

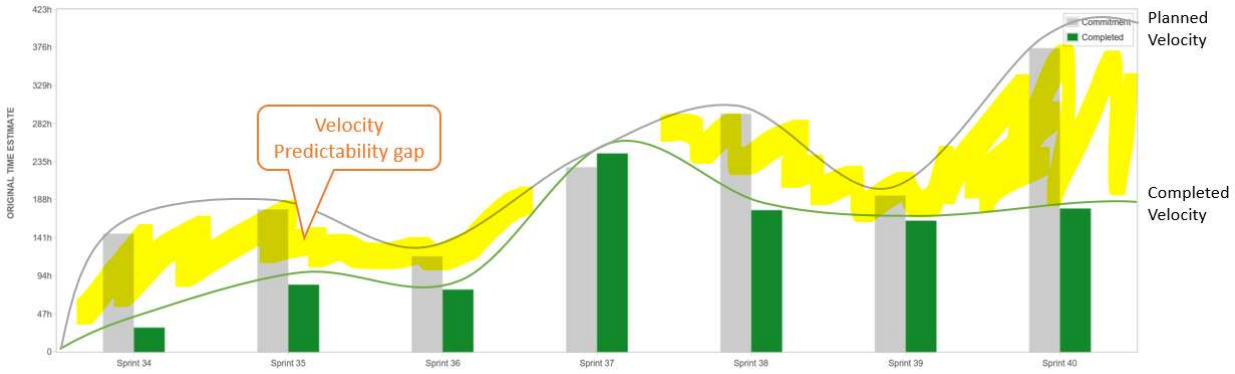

Figure 3 - Velocity Predictability and the Velocity Predictability Gap

Benefits

Like velocity variance, velocity predictability is also an indicator of process stability. Recall from Section 0 that process predictability is a key goal the project strives to achieve. Predictable flow is an indicator of minimal bottlenecks or interference in the flow of work being accomplished. When the flow of work is interrupted by new work priorities, it can lead to delays in planned work and bottlenecks. This can lead to fluctuations in the time needed to complete planned work, which in turn leads to fluctuations in velocity. Continual and regular fluctuations in velocity are a sign that processes used to deliver work are experiencing interruptions in flow that lead to bottlenecks, or that the work is continually interrupted by other "high-priority" work that stops the first-in-first-out (FIFO) flow of work.

Challenges

Velocity variance is based on story points and, therefore, cannot be compared across projects or teams. If the variances are converted from story points to percentage variances, comparisons across projects and teams might be possible.

Additional Context

When trying to identify root causes of predictability issues, look for activities that can disrupt the flow of work through its regular value delivery flow. Track work using a Kanban board to visualize the flow of work, measure WIP, and visualize FIFO flow interruptions by emergency or higher priority activities that cause context switching on a regular basis.

5.1.5 *Story Completion Rate*

Story completion rate describes the number of stories completed in a given sprint or release.

Benefits

Tracking the number of stories completed can be a good way of communicating progress on user requirements to the Product Owner and users, as opposed to story points, which can be vague or difficult for users to map to actual progress against requirements.

Challenges

Story sizes vary, and as a result the number of stories can vary across sprints or releases. The team could have completed a set of larger stories in a given sprint, but from a user story perspective it might appear as though less work was performed.

Additional Context
Epic, feature, and capability delivery can also be measured using a similar approach.

Variations

- Story completion ratio describes the number of stories completed in a given sprint or release as a ratio:

$$Story\ Completion\ Ratio = \frac{Count\ of\ Stories\ Completed}{Count\ of\ Stories\ Planned}$$

- Story point completion ratio is similar to story completion ratio but is measured in units of story points. As previously noted, story point metrics may not be useful when communicating work planned or completed to users, and the team should reserve them for planning purposes.

5.1.6 *Sprint Burndown Chart*

Teams implementing sprints use a sprint burndown chart, depicted in Figure 4, to estimate their pace of work accomplished daily. The pace is usually measured in hours of work, although there is no specific rule that limits the team from measuring its pace in story points. In Figure 4, the gray line is a tool-generated line that depicts the estimated completion date of the sprint based on the pace of work being completed.

Figure 4 - Sprint Burndown Chart

Benefits

The pace of work completed helps inform the team on whether it will be able to complete the work scheduled within the sprint. The red line in Figure 4 depicts the remaining hours of work.

Challenges
The sprint burndown chart is intended for use by the Agile team. Sharing the chart outside of the team has minimal benefits at best.

Additional Context
Figure 4 indicates that at the end of the four-week sprint a few items remained and were most likely placed back into the product backlog for re-prioritization. Although commonly used for measuring sprint progress, burndown charts can be used for additional types of timespans and cadences, such as for a release or an entire product effort.

5.1.7 *Release Burnup*

Release burnup charts measure the amount of work completed for a given release based on the total amount of work planned for the release. Usually, story points are used as the unit of measure to show planned and completed work.

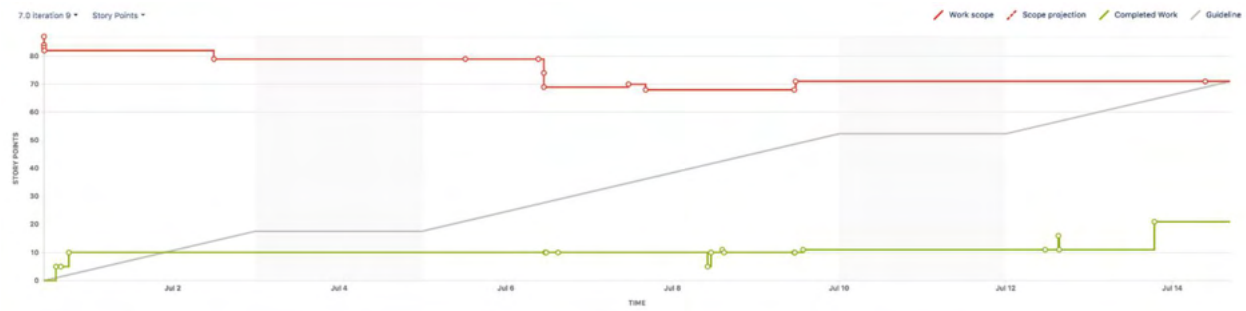

Figure 5 - Burnup Chart[3]

Benefits
The release burnup metric is a planning tool that allows the Agile team to estimate whether the team is on track to complete the items in the release. For example, if the team is completing 100 story points per sprint, and the release backlog contains 500 story points, then the team could estimate that it will take about five sprints to complete the work in the product backlog.

Challenges
The release burnup is a point-in-time assessment and should only be used as one data point among many. Specifically, the challenge with the release burnup is that if the team is following Lean-Agile principles, it is applying emergent and iterative design patterns. Given this, the team will not have defined every item in the backlog at the same level of detail. The items closer to execution will be defined in greater detail than the items planned for work later. The team does not assume it knows all requirements or levels of effort upfront. As a result, it is risky and most likely inaccurate to use the total product backlog story points as

[3] Source: Atlassian.com: https://confluence.atlassian.com/jirasoftwarecloud/burnup-chart-945124716.html

the basis for forecasting completion time. Expecting some level of variance in estimates, as in any estimate, is prudent, especially when using Agile planning techniques.

Additional Context
Conceptually, release burnup could be measured using requirements or user stories as the unit of measure as well. From the user perspective, understanding how many requirements are completed and how many remain might be a better way of communicating progress than story points. Additionally, like burndown charts, burnup charts can be applied to other scopes of work beyond releases (e.g., sprint burnup and product burnup).

Variations

- The number of requirements completed provides insight to users on requirements completed and requirements remaining.
- The number of user stories completed is similar in concept to the metric showing the number of requirements completed.

5.1.8 *Cumulative Flow Diagram*

A Cumulative Flow Diagram (CFD) is one of the best ways to visualize the flow of work through a process. The CFD achieves this by showing the count of items at each step in the process over time. Figure 6 provides an example of a CFD.

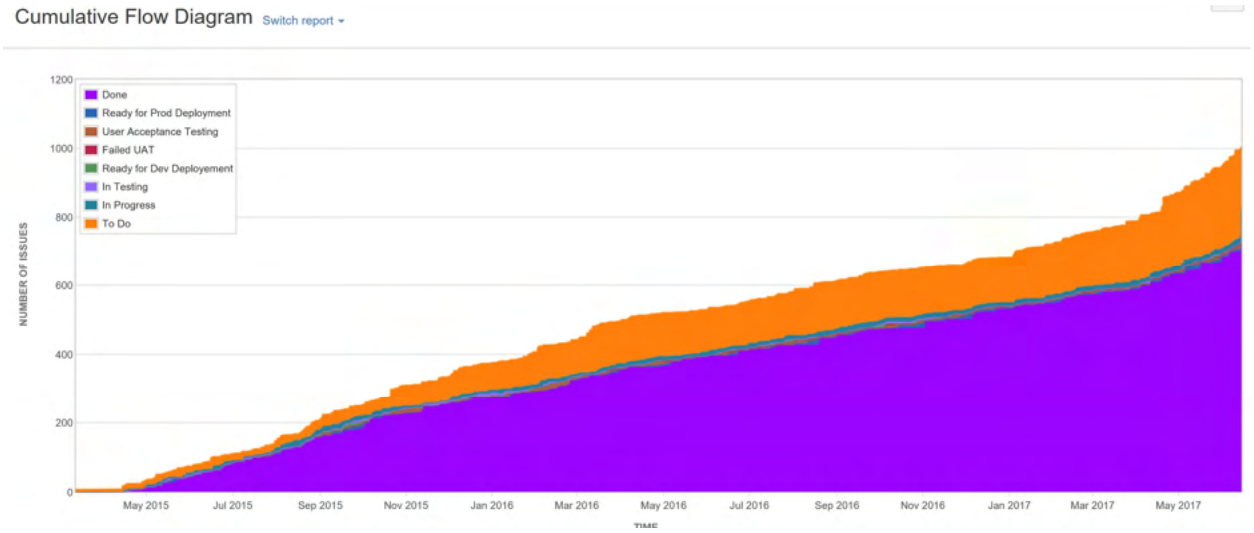

Figure 6 - Cumulative Flow Diagram Example Showing New Work Outpacing Work Completion

The CFD example in Figure 6 contains the count of work by process step. The process steps for this specific example include "To Do," "In Progress," "In Testing," "Ready for Dev Deployment," "Failed UAT," "User Acceptance Testing," "Ready for Production Deployment," and "Done."

The CFD depicts the flow of work, including WIP, by showing arrivals into the process ("To Do"), exits from the process ("Done"), and WIP at every state between "To Do" and "Done." Figure 7 depicts a manually created CFD.

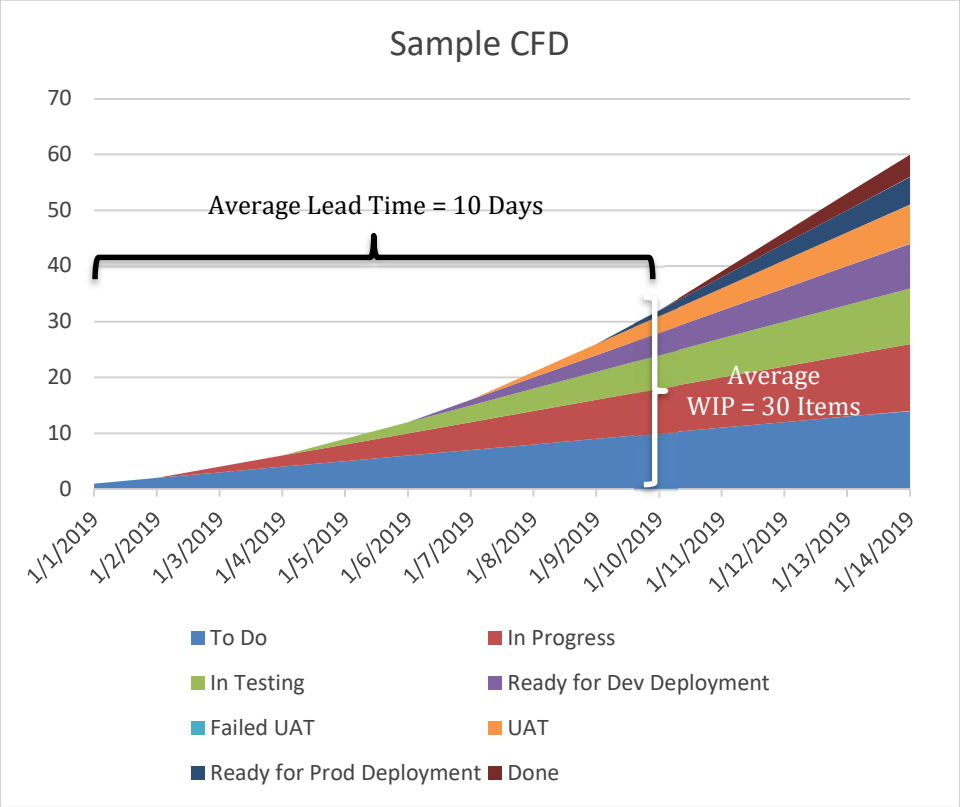

Figure 7 - A Manually Created CFD

To calculate throughput for a given period of time, the divide the number of items by the number of days (30/10 = 3 days).

Benefits

CFDs depict the cumulative arrivals in and exits from a process over time, making the chart an excellent way to visualize a significant amount of information related to the flow of work in a single graphic.[4]

Challenges

If teams are not using a tool to generate the CFD, generating the CFDs will require some effort.

Additional Context

Ideally a CFD will show parallel rising lines to indicate that the rate of work entering the process is about equal to the rate of work exiting the process. In other words, teams want to ensure that they have a throughput that supports the work demand. Predictability requires the stable process depicted in Figure 8.

[4] Source: Actionable Agile Metrics for Predictability by Daniel Vacanti.

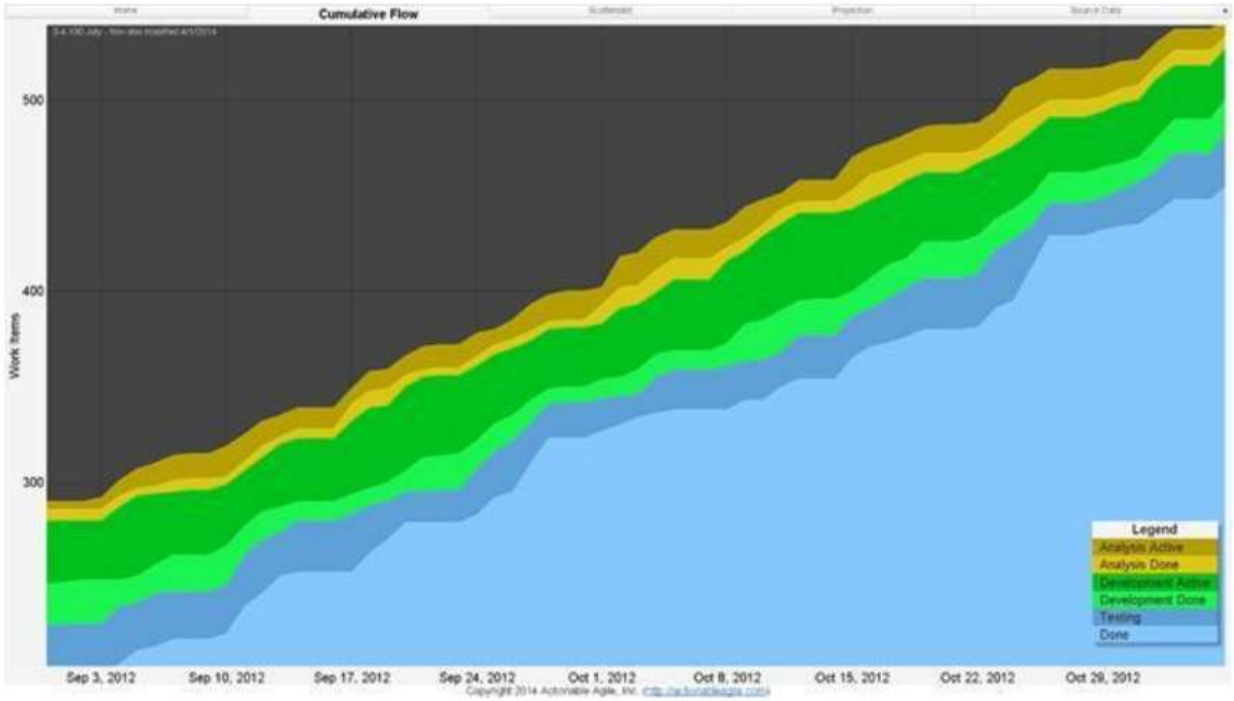

Figure 8 - Stable and Predictable CFD[5]

5.2 Agile Quality Metrics

Agile quality metrics measure the quality of work being delivered. One of the benefits of Agile, if executed correctly, is that the quality of the product being delivered improves over time due to adherence to Lean-Agile and DevSecOps practices such as ensuring that development, testing, security and operations activities are integrated as much as possible into one delivery value stream. The increased collaboration and "shifting-left" of key quality-related activities such as testing contribute to improved product quality over time.

5.2.1 *Recidivism*

Recidivism describes stories that are returned to the team for various reasons.

Benefits
Understanding why stories are returned to the team is important because the team commits to the scope of work within the sprint and all stories should be completed and closed at the end of the sprint. Therefore, if stories are not completed as the users expect, the team will need to understand the reasons, especially if recidivism rates are relatively high or increasing. If stories are not being accepted as complete, then a root cause analysis exercise can help identify possible reasons.

Additional Context
Common reasons for returning stories to the team can include incomplete or incorrect definitions of done or acceptance criteria. Work with the team to determine if such gaps in effectively communicating the work and expectations exist and ensure that the Product

[5] Source: Actionable Agile Metrics for Predictability by Daniel Vacanti

Owner is involved. When performing root cause analyses or participating in retrospectives, the team should avoid placing blame. Executing blameless retrospectives encourages teams to participate actively in root cause exercises. Teams should remember that the fix should focus on avoiding repetition of the same issue if a new person were to join the team.

Variations

- Metrics related to the count of stories that do not pass testing or user acceptance are similar to recidivism metrics.

5.2.2 *First-Time Pass Rate*

First-time pass rate describes the number of stories, features, or capabilities that pass the first time.

Benefits

Tracking first-time pass rates provides insight into how well user requirements are captured, understood, developed, integrated, and delivered. It is a good measure of the effectiveness of the end-to-end value delivery process. Programs and organizations with a mature end-to-end value delivery process have higher pass rates.

5.2.3 *Defect Count*

Defect count measures the number of defects per sprint or release.

Benefits

Defect count can be an indicator of quality control issues impacting delivery. It can also be an indicator of incomplete DevSecOps practices around configuration control, integration, and automation.

Additional Context

A large number of defects slows team velocity. Additionally, tracking defect counts over time can provide the context necessary to understand whether defect rates are trending upward or downward over time. Discovering the root cause of defects will help determine the issues causing defects. Addressing the cause of defects will improve quality.

Variations

- Change fail rate is similar to defect count, but applies to defect rates of changes released to production (See Section 5.4.4)
- The percentage of defects caught by automated testing provides insight into the level of testing automation.
- The number of defects caught in testing versus in field use can provide insight into the overall level of quality in the delivery value stream.

5.2.4 *Test Coverage*

Test coverage metrics provide insight into the level of testing that is integrated within the end-to-end development value stream process.

Benefits
Test coverage is an indicator of the level of rigor applied toward ensuring quality. In a traditional waterfall-based process, most testing usually takes place after development is complete. Lean-Agile practices approach testing as a baked-in effort with multiple touchpoints throughout the development value stream. Having insight into the amount of testing taking place and how it applies to the value being delivered is key to improving overall quality.

Challenges
Improving quality requires a shift in the way testing has been traditionally performed – usually as a handoff among the development, testing, security, and operations teams. To continually improve quality, testing must become more embedded into the planning and development efforts. Test automation is a significant contributor to improving quality, but an initial cost and level of effort are required to gain the core competencies necessary to change the way testing is performed.

Additional Context
Development teams greater responsibility for quality if they adopt practices such as test-driven development (TDD), automated unit tests, and automated tests on code check-ins. Approaches such as pairing in teams of two during development and testing activities (e.g., pair programming) can improve overall quality as well. Many of the same principles that apply to testing also apply to security testing.

Variations
- Percentage of tests that are automated
- Number of automated tests per capability, feature, epic, and/or story
- Percentage test coverage of capabilities, features, epics, and/or stories
- Percentage test coverage of code (by module or line)
- Number of cybersecurity audit/penetration tests
- Percentage of cybersecurity audit/penetration tests that are automated,

5.2.5 *Number of Blockers*

Number of blockers describes the number of events that prohibit the completion of an activity or work item. These blockers cannot be resolved by the individual assigned to complete the activity and the team needs assistance to remove the blocker.

Benefits
Understanding the number of blockers in a given sprint or release can inform the program management team and the organization of potentially larger issues such as challenges related to governance or organizational structure. For example, decision-making authority might not be pushed down to the appropriate level (where the best information rests). As a result, the individual assigned the task might be waiting for a decision. From an organizational structure perspective, if the team is matrixed across functional areas (e.g., requirements, testing, security, operations, etc.) and is still operating in a siloed manner (by performing handoffs within the team, or not communicating or collaborating sufficiently), then investigation might discover the source of the blockers.

Challenges
It is important that reviewing the number of blockers remain a blameless exercise to maintain team cohesiveness and ensure that the team focuses on removing and preventing blockers and making certain that decision-making takes place where the best information resides.

Additional Context
Blockers might be difficult to measure if not tracked daily. Not all tools track or report on blockers. As a result, the team might need to track blockers manually.

Sometimes a story might be blocked that are not identified as such. This can lead to longer lead times for the specific activity. As a result, it is good practice to ask about blockers on outlier stories with outlier lead times to ensure that the root cause is addressed.

Variations
- Number of blockers per sprint or release
- Number of blockers currently open
- Number of blockers closed
- Trends in average number of blockers per week or month.

5.3 Agile Capability Delivery Metrics

Agile capability delivery metrics measure delivery progress over time in alignment to desired outcomes (measured by value). Some of these metrics, related to assessment of delivered value based on user feedback, might not be generated from a tool. Rather, the metrics might reflect a combination of tool-supported data, survey responses, and communication with the users to determine whether the product is delivering the desired value.

5.3.1 *Delivered Features (or Delivered Capabilities)*

The count of delivered features measures the business-defined features accepted and delivered. Some programs may prefer to measure delivered capabilities over features. Both measures can work.

Benefits
A key desired outcome of Lean-Agile practice is to deliver value faster and more often (related to flow) so that the team can receive feedback faster. Given this, measuring delivered features and the pace of feature delivery is key to determining whether users are receiving what they need faster, more frequently, and in a predictable manner.

Challenges
Not all features are equal. Some features may be larger or more complex than others. For this reason, it is important to provide such context as needed.

Additional Context

Some Agile teams might not include features as part of their Agile requirements model, while others might prefer to define business needs in terms of capabilities. What is important is to find a functionally equivalent measure, such as capabilities, features, epics, user stories, or requirements, that provides insight into the pace of value delivery.

Variations

- List of features delivered over time (e.g., list of features implemented per a roadmap)
- List of capabilities delivered over time
- List of epics delivered over time,
- List of user stories delivered over time
- List of requirements delivered over time
- Achievement date of MVP (and/or MVCR for the SW Pathway)

5.3.2 *Delivered Value Points*

This metric represents the count of value points delivered to users for a given release. Value points are usually defined by the users (or user representatives) to indicate a business value assigned to a given feature, capability, epic or story.

Benefits

Value points provide direct insight into how the users perceive the value they are receiving.

Challenges

Value points can be somewhat subjective, as they are based on how the user scores a given value. For this reason, taking the time to establish the definition of done and acceptance criteria helps the team understand how the users will perceive value. If the user community has experienced issues with the perceived value of work provided, then value points might be a good measure to consider. However, capturing value points on an ongoing basis requires ongoing, active involvement by the users to assign value points prior to development activities and reassess the value once a product is delivered.

Additional Context

Some Agile teams work with users to assign values during the planning of the release and give the users another opportunity to adjust the *received* value after seeing a demonstration of the completed work. The difference between the initial value and received value can be another metric that could be used to measure how well the team is interpreting and delivering the desired business value.

5.3.3 *Level of User Satisfaction*

This metric represents the measure of user satisfaction based on the value delivered by the product or solution.

Benefits

User satisfaction is a good indicator of value delivery, since user satisfaction will most likely increase as value is delivered consistently and quality is improved.

Challenges
User satisfaction is not a readily available measure. Most likely, it requires a baseline measurement followed by continual and ongoing incremental measurements to capture user satisfaction trends. This can be done through a survey or face-to-face discussions to collect the data points necessary to gauge user satisfaction.

Additional Context
A survey may require the assistance of a survey specialist to ensure that the survey questions and approach do not lead users to specific answers. The survey should be used for establishing the initial baseline as well as for ongoing incremental checks with users.

Variations
- Usage trends (number of solution users; traffic/visits) – Alternate measures such as usage trends might be an easier way to gauge user satisfaction and would not require conducting a formal survey. For systems with a large number of users who are not required to use the product/solution, if usage increases the team can assume that users are perceiving greater value than previously, and as a result, are accessing the product/solution more often.
- User engagement – Participation in demonstrations or overviews of the product/solution can be another indicator of product value and quality.
- User registrations – Increases in account registrations can be a sign of greater interest by users due to a product's perceived value.

5.4 DevSecOps Metrics

DevSecOps metrics relate to measurements that provide insight into the organization's delivery pipeline. They also include metrics that provide a high-level assessment of the organization's ability to integrate, deliver, monitor, and restore products. DevSecOps focuses on integration of all activities related to planning, development, testing, security, integration, deployment, and operations. In order to deliver on a regular and continuous basis, the organization must adhere to a set of principles that align with continuous planning, continuous integration, continuous delivery, and continuous monitoring. Organizations that actively measure the efficiencies of their delivery pipeline have greater insight into inefficiencies, and therefore will be in a better position to improve the pipeline for consistent and fast delivery.

5.4.1 *Mean Time to Restore (MTTR)*

MTTR measures how quickly a system or solution can be restored to functional use after a critical failure.

Benefits
This measure assesses the organization's ability to maintain configuration control across all levels of the delivery pipeline. Organizations with fast MTTR generally have a relatively mature delivery pipeline, including a continuous integration and delivery approach. They usually can perform code-based deployment and configuration so that deployments across environments are consistent (using tools and technologies to support orchestration and containerization).

Challenges
Although this is an important measure, there is no single method for improving MTTR. Improving the overall DevSecOps maturity of the organization takes time.

Additional Context
MTTR is a valuable metric when assessing a team's ability to move code across environments and deploy it quickly. It does not measure the team's ability to develop code.

5.4.2 *Deployment Frequency*

Deployment frequency provides information on the cadence of deployments in terms of time elapsed between deployments.

Benefits
Lean-Agile practices promote fast feedback and delivery of value to the customer. To accomplish this, the team must develop a regular delivery cadence (e.g., agreed-upon incremental and iterative sprint and release cycles). Some organizations deploy solutions or upgrades daily, or multiple times per day. This does not mean that every Government program should strive for the same deployment cadence, but trying to continually reduce the period between deployments helps improve the feedback cycle and pace of value delivery to the customer.

Additional Context
When considering delivery frequency, it is also important to consider who will receive the deliverable. Some programs may have both internal and external users/stakeholders. These programs might decide that the first delivery will be a limited delivery to the internal users/stakeholders to verify quality and value prior to deploying the solution to a broader audience.

Variations
- Release cadence (time between software releases into operations).

5.4.3 *Lead Time*

This flow metric represents how long it takes to deliver a required solution.

Benefits
The time it takes to deliver a solution to the user is often a source of contention. The user wants a predictable time estimate for completion of work, but the development team cannot accurately and consistently predict the time to completion if there are flow issues in the development value stream, as noted earlier. For this reason, achieving a measurable and predictable lead time for new work would represent a significant advance in improving accuracy and predictability of estimates.

Challenges
Measuring lead time is simple in concept, especially if using a tool that allows the flow of work to be visible, such as a Kanban board. However, improving lead time requires

addressing the flow-related issues that lead to unpredictability, which in turn requires a commitment from everyone involved, including the user community and product stakeholders.

Additional Context
As noted in Section 5.1.4, one of the most significant factors leading to unpredictable flow is the introduction or insertion of new tasks as a top priority. This causes a chain reaction where the flow of work is interrupted to address the new high-priority activity.

From a DevSecOps perspective, understanding lead time and potential bottlenecks that might be causing delays in delivery of requirements is important to ensure a stable and predictable continuous integration and continuous delivery process. From a technology and process standpoint, the Agile team should consider the end-to-end integration and automation of DevSecOps activities and technologies where possible.

Variations
- Lead time is also known as cycle time.
- Segments of the development value stream can also be measured for lead time, such as the time required for full regression test and cybersecurity audit/penetration testing. Examining segments of the development value stream such as the ones listed below will help the team identify bottlenecks and prioritize areas that require efficiency improvements:
    - Lead time from new requirement identification to sprint assignment
    - Lead time from new requirement to completion of development (or start of testing)
    - Lead time from new requirement identification to completion of full regression testing and cybersecurity audit/penetration testing
    - Lead time from new requirement identification to deployment
    - Lead-time required to re-host software after major hardware refresh
    - Lead time to complete testing
    - Lead time to complete security and Authority to Operate (ATO) activities
    - Lead time to field or release
    - Time from code committed to code in use
    - Time required to field high-priority functions or fix newly found security holes. (Inserting new work into the planned flow of work interrupts the flow and predictability of work completion estimates, as discussed in Section 0 and Section 5.1.4. Insight into the number of interruptions will in turn provide insight into the quality of upfront planning. If upfront planning is improved, high-priority insertions into planned work will decrease.)

5.4.4 *Change Fail Rate*

The percentage of changes to production that fail.

Additional Context

This metric applies to changes in software, as well as configuration changes that are applied to production, and measuring the success rate of those changes in production as a percentage of total changes applied to production during a given deployment.

Review fail rates and perform root cause analyses to identify quality issues that could be addressed with improved testing, integration and deployment. For example, integration of automated tests could reduce testing lead times and improve the overall consistency and quality of testing; containerization of environments could improve consistency and reliability of environments. Additionally, involving testing, security, and operations teams early in the planning activities ("shifting-left") could ensure better collaboration and reduce handoffs, which could result in improved quality and reliability of delivered changes.

5.5 Cost Metrics

Cost metrics provide insight into the organization's planned and actual costs. Planning and measuring cost are just as critical for Agile projects as for traditional projects. An advantage of Agile development is that the frequent deliveries of software provide actual data soon after delivery of capability. This information can and should be used to estimate the costs of future delivery efforts. Near-term software delivery efforts can be estimated by leveraging process, quality, and product metrics as well as product backlog information. Estimates for near-term deliveries will have a greater level of fidelity than estimates for later deliveries. Longer term delivery efforts should rely on higher level approaches and establish a connection between near-term work and longer-term work, which is usually planned in a roadmap of capabilities or features to be delivered over time. Some connection between work scope and effort will be needed across the detailed metrics and overall capability plan.

This section describes metrics related to estimating the costs of software acquisition and development as well as any enterprise services and hardware needed. There may be additional program costs outside of these areas that should also be estimated and included in an overall program cost. Additionally, metrics must include costs for both Government and contractor efforts.

5.5.1 *Total Cost Estimate*

This estimate provides the total estimated cost for the product being developed and/or the service being acquired. Total cost must include all relevant cost elements. This includes software development costs, software license costs, hosting and infrastructure costs, and costs for other activities such as program management, system engineering, system testing, training, etc., that may be applicable. The cost estimating approach will vary based on the type and nature of effort to be undertaken, type of products to be procured, and level and type of data available. The approach may also vary by element and by time. For example, estimation approaches for near-term deliveries may use engineering build-up whereas approaches for longer term deliveries may use extrapolation, analogy, or parametric techniques. Elements that could be covered in total cost calculations include:

- Agile team cost – cost of the Agile team as a unit over time (Agile teams should remain a constant size. Therefore, this cost element could be used to extrapolate cost of the

work over time by the development team if a duration to complete the work can be estimated.)
- Requirements costs – usually based on the collection of capabilities, features, epics or user stories
- Software costs – as applicable
- Hardware costs – as applicable
- Cloud services and computing costs (including bandwidth and storage costs as appropriate)
- Licensing costs – as applicable
- Program management cost – cost of the program management activities that usually fall outside of the Agile team.

The cost estimation approach can also depend on whether the program seeks to obtain services over a period of time (e.g., DevSecOps expert; Full Stack Developer; Tester) or delivery of a product. The approach can be based on a set of specific Agile user requirements (user stories) contained in a product backlog that incorporate enough detail to provide sufficient confidence in estimates based on the program's risk tolerance.

Variations
- Total Product Cost Estimate
- Total MVP Cost Estimate.

5.5.2 *Agile Team Cost*

Measures the size and annual cost of the development teams, to include the average size and makeup of each Agile team and the number of Agile teams.

5.5.3 *Total Hardware, Software, Cloud, and Licensing Costs*

Measures the total estimated cost of hardware, software, and licensing fees (including cloud services).

5.5.4 *Total Program Management Costs*

Captures the size and annual cost of the program management team, including both Government and contractor program management (if applicable).

5.5.5 *Allocation of Development Costs*

Pertains to allocation of development costs related to defect resolution, new feature or capability implementation, and code refactoring (paying off technical debt).

5.5.6 *Percentage of Resources by Function*

Captures the percentage of programmers, designers, user interface engineers, system architects, and other key development categories within Agile teams.

5.5.7 *Software Licensing Fees*

Captures license costs related to software and cloud-based application services.

5.5.8 *Computing Costs (including cloud services)*

Captures license costs related to cloud-based application and computing services.

5.5.9 *Bandwidth Costs*

Captures bandwidth costs.

5.5.10 *Storage Costs*

Captures fees for storage costs.

5.5.11 *Other Costs Associated with the Program*

Captures other costs not associated with one of the above categories (e.g., user travel).

5.5.12 *Burn Rate*

Captures incurred cost over a period of time (e.g., monthly burn rate, sprint burn rate, release burn rate).

6 Value Assessment Metrics

Value can be measured throughout the product development and operation life cycle. The types of value measured at the team level, when working on iterative development and release cycles, are focused on measuring value received by the user. Annual value assessments at the Enterprise level are focused more on whether the operational capabilities meet mission outcomes and objectives. Value assessments typically involve a combination of subjective and objective measures.

6.1 Guiding Principles of a Value Assessment

During the planning phase, the program should work with the user community to collaboratively define a plan to measure value (including factors to consider, metrics to collect, periodicity of the assessments, etc.). Value assessments should consider the following factors:

- **Increase in mission effectiveness:** Mission effectiveness measures the extent to which the system contributes to mission accomplishment. This may be a qualitative user assessment or include quantitative or statistical measures. Examples of mission-based metrics include lethality, survivability, accuracy, etc.
- **Cost savings**: Cost savings span the system, related systems, and mission operations. Programs should use quantitative data if possible.
- **User workload reduction:** Workload reduction represents a reduction in the amount of effort an operator requires to accomplish a mission task. Effort in turn

relates to the number of operations required to perform the task or to the cognitive ability required to perform the task. This measure can be qualitative or quantitative.

- **User manpower reduction:** Manpower reduction is a reduction in the number of operators required to accomplish a mission task. This measure can be qualitative or quantitative.
- **Equipment footprint reduction:** Footprint refers to the amount of computer boxes, generators, or other equipment necessary to accomplish the mission. A reduction in footprint represents a significant reduction in the amount of space required to accomplish the mission. Programs should use quantitative data if possible.
- **Number and significance of field reports:** Field reports (or software trouble reports, software problem reports, etc.) capture problems that users report from the field. Software problems can affect reliability and availability. Programs should use quantitative data.
- **User satisfaction:** Satisfaction is a subjective assessment by the end user.

The user community, including end users, should evaluate factors such as those listed above to determine the value of the delivered software. Value assessments should be conducted at least annually or at more frequent intervals as determined by the user community and the Program Manager.

6.2 Comparing Value Delivery against Cost

Value assessment metrics comprise a collection of the already noted metrics related to delivery of product value (e.g., Delivered Features, Delivered Value Points, Level of User Satisfaction), and work completion rates (e.g., Release Burnup). Note that these measures are based on story points, requirements (e.g., capabilities; features; user stories), and value assessments that are subjectively sized/assessed and specific to the project team and the users they support. Given this, the value assessment metrics should be used for internal project performance assessments and projects should very carefully consider whether the metric is transferrable across projects or programs. For example, the information could potentially be used for comparison in cases where the team(s) and user bases do not change.

Projects should develop a set of internal performance metrics, based on the above-noted metrics, and factor in cost information to gauge value delivered and efficiencies gained per dollar spent. Sample metrics supporting this intent include:

Cost-Focused Measures

Value delivered based on the user's assessment of value:

- Cost per Delivered Value Point = Cost to Date / Total Delivered Value Points.

Derivative measures that also provide insight into value delivered from a productivity perspective include:

- Cost per Delivered Feature or Capability = Total Cost to Date / Total Delivered Features or Capabilities

- Cost per Delivered Story Point = Cost to Date / Total Delivered Story Points.

Delivery-Focused Measures

Alternatively, delivered value per unit of cost could be another useful way of looking at value delivered per cost unit (e.g., $/ $K/ $M) spent.

- Delivered Value Points per Unit of Cost = Total Delivered Value Points / Cost to Date *where cost can be measured in $, $K or $M.*

Derivative measures that also provide insight into value delivered from a productivity perspective include:

- Delivered Features or Capabilities per Unit of Cost = Total Delivered Features or Capabilities / Cost to Date [where *cost can be measured in $, $K or $M*
- Delivered Story Points per Unit of Cost = Total Delivered Story Points / Cost to Date *where cost can be measured in $, $K or $M.*

These metrics can be applied to a specific time period (e.g., month, year, or project duration) or by specific project cadence (e.g., sprint, release). The project Metrics Plan should specify the details of the value metrics that will be applied.

7 Metrics Implementation Guidance and Considerations

Assessing which metrics to implement first, and in what order can be a challenge. There is no single, correct order. Rather, the determination depends on the program maturity in terms of its ability to continuously deliver the most important capabilities in a responsive manner to achieve mission outcomes. Understanding the organizational maturity pyramid provides the background and context.

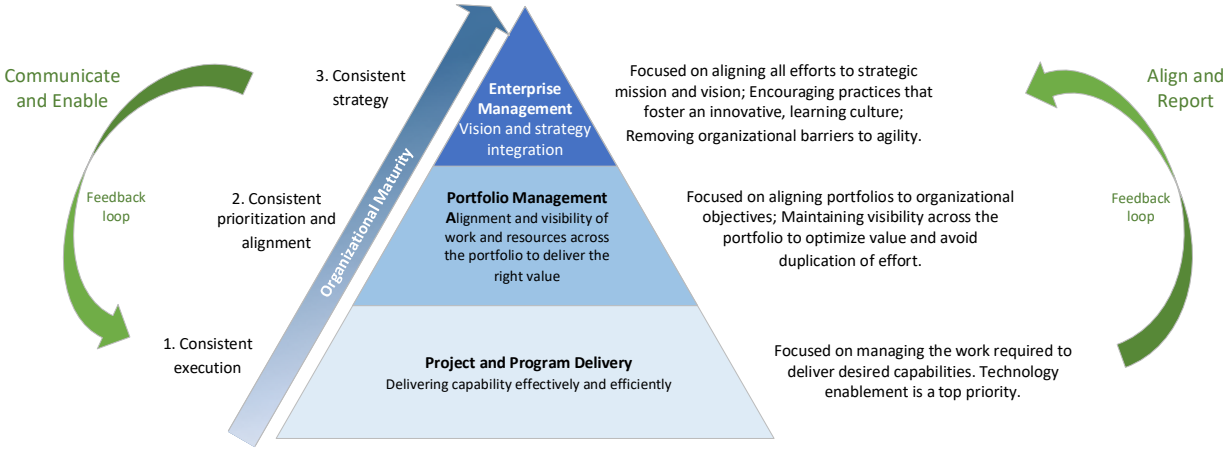

Figure 9 -- Aligning capability delivery with organizational strategy

Organizations stive to align delivery of capabilities to organizational outcomes. At the strategic and highest levels of the organization, the need for visibility of progress toward outcomes and overall value received is of primary importance. Effectively measuring capability delivery to organizational outcomes requires line-of-site visibility at the product, program, portfolio, and enterprise levels. At the project and program level, metrics are also

captured to provide insight into effective and efficient capability delivery. A metrics strategy and implementation approach should address these needs.

As a starting point, a default implementation roadmap is offered in this section that aligns the metrics categories in this document to the organizational pyramid and recommends a "should start by" timeframe relative to the Planning and Execution periods of a typical Agile execution effort.

7.1 Implementation Roadmap

Figure 10 provides a high-level implementation roadmap for implementation of metrics based on the metrics categories in this guide.

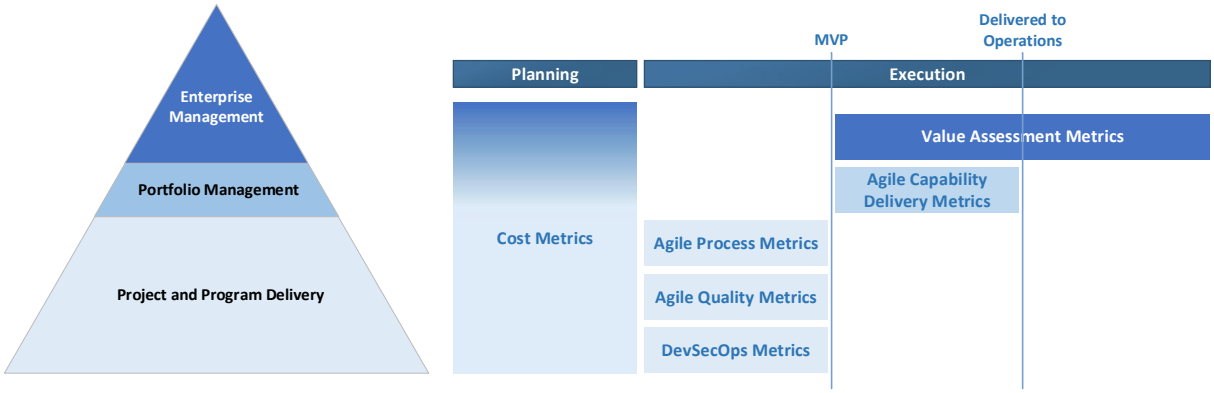

Figure 10 -- Metrics implementation roadmap

The roadmap assumes an initial planning period whereby costs are aligned across all units of the organization. The roadmap indicates when each group of metrics should start to be implemented for collection and reporting. Once implementation begins, the metrics should be collected, reported, and refined throughout the life of the product. For example, the cost metrics, although initially defined prior to execution, should continue to be refined as more information and data becomes available throughout product delivery and operations. Similarly, Agile process, quality and DevSecOps metrics should be collected throughout once execution begins. Agile capability delivery metrics should be collected with the delivery of the first MVP if possible. Upon operational capability delivery, value assessment metrics should be collected to provide insight into mission effectiveness and gained efficiencies. Although the implementation roadmap is defined as a starting point, there are other key factors to consider when developing an overall metrics implementation strategy. A few of the key implementation factors to consider are noted in the next section.

7.2 Implementation Factors

Consider the following factors, as part of a broader self-assessment, to determine organizational readiness and to tailor the metrics implementation strategy.

Agile experience

Organizations just beginning the Agile journey should focus on building the foundational components of agility at the team level, executed in small pilots to build foundational experience. Upon core experience in agility at the team level, organizations can experiment with agility at scale by implementing frameworks, processes, practices, and tools that enable coordination, visibility and reporting among multiple teams, or a team of teams. Programs new to Agile processes should prioritize defining the core set of Agile Process Metrics, and consider impacts to existing costing approaches.

DevSecOps experience

DevSecOps technical practices enable agility and improve quality and predictability of capability delivery by automating as many repeatable activities as possible and providing visibility throughout the delivery and operation of the capability. DevSecOps achieves this through implementation of a delivery pipeline consisting of technology, tools and practices to achieve continuous integration, testing, delivery, deployment and monitoring of software. Successful application of DevSecOps technical practices requires early and continual collaboration among development, testing, security and operations team members. Automation of metrics and reporting at all stages of delivery and operation provides the visibility necessary to identify bottlenecks, address quality issues, and improve delivery predictability. Improved quality and predictability lead to long-term cost savings and improvements in cost estimates. For these reasons, DevSecOps is increasingly considered a required technology enablement need and a foundational building block for sustained agility. However, programs that are new to agile project management practices and DevSecOps technology practices should carefully consider the level of effort and learning curve required to tackle both simultaneously.

Programs with a modern software delivery pipeline will be more prepared to produce DevSecOps metrics that offer information on the efficiency and effectiveness of software development, quality, delivery and operations processes. Conversely, programs that are new to DevSecOps practices will require more time to produce such metrics. Instantiating a mature software delivery pipeline that supports continuous integration, delivery and monitoring requires commitment, planning, resources, and time. Programs will not achieve DevSecOps-in-a-day, just as programs will not achieve agility-in-a-day. For these reasons, each program should consider the cost and level of effort required to collect and report each DevSecOps metric. Programs should consider standing up a software delivery pipeline before project execution, as the level of effort to stand up a pipeline while implementing capability can be a large undertaking, especially for programs new to Agile.

Cross-program integration for large system-of-systems solutions

Large solutions often comprise system-of-systems architectures spanning across multiple programs within a portfolio. The ability to continually integrate at this scale requires agility and responsiveness at the portfolio level. Portfolios govern the integration and delivery of capabilities across programs and in alignment to enterprise objectives. Portfolios should look for strategies to enhance responsiveness and delivery cadence by streamlined processes related to prioritization, funding, oversight, and decision-making. Portfolios consisting of multiple programs should consider alignment of metrics across programs to facilitate aggregation of metrics across programs. The Portfolio roadmap of capabilities

should also align to program and product roadmaps in order to clearly and effectively provide line-of-site visibility into delivery progress toward achieving enterprise outcomes.

Level of engagement with leadership on outcomes and value
Organization leaders require feedback on whether the work performed has had the desired impact on strategic, mission outcomes and objectives. The Value Assessment should be tailored to capture progress toward mission outcomes, and should inform the metrics strategy at the program level as well. Achieving the alignment necessary to deliver the value assessment requires engagement at all levels.

Appendix A Abbreviations and Acronyms

ALM	Application Lifecycle Management
ART	Agile Release Train
ATO	Authority to Operate
CD	Continuous Delivery
CFD	Cumulative Flow Diagram
CI	Continuous Integration
CIO	Chief Information Officer
DaD	Disciplined Agile Delivery
DoD	Department of Defense
DSDM	Dynamic Systems Development Method
FIFO	First-In-First-Out
FITARA	Federal IT Acquisition Reform Act
GAO	Government Accountability Office
IT	Information Technology
LeSS	Large-Scale Scrum
MTTR	Mean Time to Restore
MVP	Minimum Viable Product
OUSD(A&S)	Office of the Under Secretary of Defense for Acquisition and Sustainment
PI	Program Increment
PM	Program/Project Manager
PO	Product Owner
RTE	Release Train Engineer
SAFe	Scaled Agile Framework
TDD	Test-Driven Development
UAT	User Acceptance Testing
WIP	Work in Progress
XP	Extreme Programming