Carnegie Mellon University
Software Engineering Institute

Automated Data for DevSecOps Programs

William Richard Nichols wrn@sei.cmu.edu 412-268-1727
Hasan Yasar hyasar@sei.cmu.edu 412-268-9219
Luiz Antunes, lantunes@sei.cmu.edu 412-268-2395
Christopher L. Miller clmiller@sei.cmu.edu 703-247-1416
Robert McCarthy remccarthy@sei.cmu.edu 412-268-6920

Abstract

Automation in DevSecOps (DSO) transforms the practice of building, deploying, and managing software intensive programs. Although this automation supports continuous delivery and rapid builds, the persistent manual collection of information delays (by weeks) the release of program status metrics and the decisions they are intended to inform. Emerging DSO metrics (e.g., deployment rates, lead times) provide insight into how software development is progressing but fall short of replacing program control metrics for assessing progress (e.g., burn rates against spend targets, integration capability target dates, and schedule for the minimum viable capability release). By instrumenting the (potentially interacting) DSO pipelines and supporting environments, the continuous measurement of status, identification of emerging risks, and probabilistic projections are possible and practical. In this paper, we discuss our research on the information modeling, measurement, metrics, and indicators necessary to establish a continuous program control capability that can keep pace with DSO management needs. We discuss the importance of interactive visualization dashboards for addressing program information needs. We also identify and address the gaps and barriers in the current state of the practice. Finally, we recommend future research needs based on our initial findings.

1 Introduction

We undertook this research because program management in DoD face challenges measuring program performance and conducting effective oversight of continuous integration/continuous delivery (CI/CD). Closing this gap should enable adoption of modern practices. To realize the benefits of CI/CD, we investigate how to collect and use metrics from the modern development pipelines to support cost and schedule prediction models derived from that data. Our research project, therefore, examines how to exploit the automation within the DevSecOps (DSO) environment to benefit program management.

Software acquisition increasingly involves software development using CI/CD, as described in the Defense Acquisition University (DAU) training, *Software Acquisition Pathway (SAP) Interim Policy and Procedures*. This SAP training is available to "facilitate rapid and iterative delivery of software capability to the user" (CSO DoD, 2021b) and empower program managers (PMs) (Brady & Rice, 2020). However, acquisition program management professionals struggle to keep pace with continuous delivery because it does not come with continuous data or continuous estimation models. Continuous delivery can produce working software not only at the end of sprints, but also daily or even multiple times per day. To make commitments, changes, or program interventions, the program office needs up-to-date information on capability readiness, costs, and progress rates. However, the delivery of relevant data reports can take weeks or months. The actual CI/CD progress is thus constantly ahead of effective program control.

There are many challenges to managing CI/CD intensive programs including the increasing complexity of software-enabled systems, hardware-in-the-loop testing and simulation, and inclusion of COTS and/or open-source components. Within this context, acquisition is adapting by using the SAP. DoD policy is clear; IT expects PMs to use metrics for planning, control, and oversight. As stated in SAP policy, "The PM shall identify, collect, and use management, cost, schedule, and performance metrics to enable effective program execution by the PM and other stakeholders. **Metrics collection should leverage automated tools to the maximum extent practicable**"(emphasis added) ("Under Secretary of Defense" DoD, 2020).The specific list of minimum requirements includes process efficiency, software quality, software development progress, cost, and capability delivery (e.g., value delivered).

2 Background

After conducting an extensive literature review, we found that the peer reviewed literature is devoid of studies about automated data collection for CI/CD (Prates et al., 2019). Although non-peer reviewed literature exists, *it either* addresses operational issues rather than PM issues, *or it* is limited to a narrow research topic rather than DoD *programmatic* needs (Vassallo et al., 2019). Moreover, little attention has been given to managing multiple interacting pipelines, each with a distinct technical stack, personnel, and rates.

Several sources—Practical Software Measurement Group (PSM), National Defense Industrial Association (NDIA), and International Councils on Systems Engineering (INCOSE) (Jones et al., 2020b), and the DoD (Defense, 2019)—recommend metrics for Agile and CI, but none connect to automated collection or have metrics that have been rigorously validated. The situation is similar for DSO with regard to the DevOps Research and Assessment (DORA) metrics (Forsgren & Humble, 2015; Forsgren et al., 2018). The Defense Innovation Board (DIB) explicitly identified this gap, "In the beginning stages of DoD's transformation to DSO methods, the development and operations community will need to work closely with the cost community to derive new ways of predicting how fast capability can be achieved. For example, estimating how many teams' worth of effort will be needed to invest in a given period of time to get the functionality needed. [...] New parameters are needed, and more will be discovered and evolve over time" (McQuade et al., 2019).

By replacing practices that, in the past, have been labor intensive and prone to error, DSO enables CI/CD. CI is the automated process that developers use to integrate code and then build, test, validate, and deploy new applications. The automation that makes these DSO practices possible, in turn, spawns a large amount of data as a byproduct. Making this data available enables stakeholders to assess the health of a project, including its development performance, operational performance, whether it is sufficiently secure, and how frequently upgrades are being delivered.

2.1 DSO, Pipelines, and Automation

To implement the automated continuous estimation of software-intensive systems, you must first define what is being measured. The specific measurements depend on the decisions to be made. In this paper, we focus on the decisions made by program managers and the development pipelines as the object of measure.

DSO is a software engineering culture and practice that unifies software development (Dev), security (Sec) and operations (Ops) personnel and their practices. The essential concepts of DSO comprise automating, monitoring, and applying security to all activities of software development. These activities include feature planning, bug fixing, feature development, application and support infrastructure builds ant testing, and releasing new software—whether that involves maintaining operational software that supports a user base, or monitoring operational systems for performance and security-related events (CSO DoD, 2021b).

DSO consists of a set of principles and practices that enable better communication and collaboration among relevant stakeholders for specifying, developing, and operating software and systems products and services and making continuous improvements to all aspects of the lifecycle (IEEE, 2021).

A DSO pipeline consists of a chain of processing elements arranged so that the output of each element is the input of the next, much like a physical pipeline. The analogy to a physical pipeline is a weak connection (i.e., there is no requirement for ordered processing or tight coupling). In fact, many DSO pipeline elements use asynchronous messaging and decoupled processes (e.g., GitOps). Often the term *pipeline* is used to describe a set of processes that tie together and eventually produce a software artifact. Sometimes this output is then used as an input into a different, possibly distinct, pipeline or pipeline instance (CIO DoD, 2019; CSO DoD, 2021).

Automation of the DSO pipeline provides an unprecedented opportunity to collect software development data from the engineering tool suite without burdening the software development staff with providing performance metrics, thereby distracting effort from their development work. Eliminating manual data-collection activities not only reduces the effort associated with performing these tasks, it also reduces the opportunity to inject bias into the data. Automated data collection also provides a continuous data collection and storage capability that can revolutionize the frequency and fidelity of software estimation.

2.2 Programmatic Needs

Program management is usually defined as managing a group of related projects using specific management techniques, knowledge, and skills. PMs must work with senior leaders and stakeholders across multiple departments and teams. Their decisions are likely to be strategic and connected to the financial calendar. Their responsibilities include coordinating resources and outputs across teams rather than within teams.

PM responsibilities include strategy, finance, and communication. Their overarching purpose is to guide their program to successful outcomes. Specific responsibilities include the following (Zein, 2010):

- Manage the program's budget.
- Establish high-level performance objectives.
- Manage a strategy, and guide investment decisions.
- Define the program governance (i.e., controls).
- Plan, monitor, and control the overall program.
- Manage risks and issues and implement corrective measures.
- Coordinate the projects and their interdependencies.
- Manage and use resources across projects.
- Manage communication with stakeholders.
- Align the deliverables (i.e., outputs) to the program's outcome with the aid of the business change manager.
- Manage daily program operations throughout the program lifecycle.

The PM needs information to provide adequate resources, negotiate commitments, and otherwise satisfy stakeholder needs. The status of any project reflects not only the status of its own code, but also how its dependencies affect it. These needs include, but are not limited to the following:

- baseline and benchmark performance
- product completion and cost rates with probabilistic cost/schedule projections
- a master plan, master schedule, and lead times
- when work begins and is completed
- which queues can be bypassed
- resource needs and resource utilization assuming nominal conditions and "what-if" scenarios

2.3 Prior Work/State of the Practice

Prior work has identified numerous candidate measures and opportunities in the DSO pipeline (McQuade et al., 2019; Jones et al., 2020b; Defense', 2019) at all stages of development, including feature request, requirements development, architecture, design, development, test, delivery, and operations. Automatically collecting data generated by the tools used during these stages can provide information on product size, effort, defects, rework, and durations—often on a feature, story, and/or component level of granularity. A key challenge is creating new features, in the machine-learning sense, from the raw data to deduce status and improve predictive power. Nonetheless, peer-reviewed research is severely limited for programmatic metrics. In this section, we summarize the current literature on DSO metrics research and practice.

2.3.1 Academic Research

The research community has only recently begun to study measurement in DSO. A multivocal literature review by Prates found limited prior academic research about DSO metrics (Prates et al., 2019). Moreover, the metrics that Prates identified focused on security and quality (e.g., defect burn rate, critical risk profiling, defect density, top vulnerability types, number of adversaries per application, adversary return rate, point of risk per device). Prates' summary noted "It was very hard to find information regarding metrics associated with DSO in academic literature." Primarily, the metrics identified were security related rather than programmatic. In a 2020 paper, Mallouli focused on cybersecurity rather than programmatic issues (Mallouli et al., 2020). A more general contribution from Mallouli included a metrics-driven DSO architecture that includes measuring tools, a core platform, a database, and analysis tools. Mallouli's architecture diagram aligns with our vision of general-purpose needs for DSO measurement.

2.3.2 The Government and DSO

One defining characteristic in the DoD is that the environments in which systems operate are highly regulated. Because of this, agencies are not free to simply adopt strategies and frameworks from industry environments. The Software Engineering Institute (SEI) has written guidance that describes special conditions found in these environments, difficulties generated by them, and possible solutions to make DSO practices work (Morales, Turner, Miller, Place, & Shepard, 2020).

One of the biggest pushes to Agile, DevOps, and DSO started after appointing a Chief Software Officer at the USAF. The DoD has since undertaken an effort towards the internal standardization of a platform with artifacts and processes that may be used across departments and agencies. While not a one-size-fits-all solution, this initiative has promoted the DSO mindset across multiple programs that now is the right time to implement DSO practices. To support these initiatives, the DoD prepared guidance on how to adopt DSO practices and provided ideas about teams and personnel organization, cost, and levels of effort. Most of these departments and agencies follow the guidance in these DoD documents:

- The **DoD Enterprise DevSecOps Playbook** provides detailed coverage of all aspects of the design, development, and operation of systems under the DSO lens. The topics covered include shifting a program culture towards DSO, assembling a software factory (SWF), implementing DSO pipelines in an SWF, capturing basic metrics monitor progress, orchestrating frameworks, and securing a system and its infrastructure (CSO DoD, 2021a).
- The **DoD Enterprise DevSecOps Reference Design** preceded the playbook mentioned above. This document provides technical implementation details such as selecting containers versus virtual machines, using a DoD centralized-artifact repository, and organizing a DSO pipeline and its environment (CSO DoD, 2021b).

2.3.3 Industry

Industry tends to be on the "bleeding edge" of technology and is always adopting practices that can provide a competitive advantage to its organizations. Much of the guidance initially adopted by industry comes from documents published by consortiums of organizations that have a solid track record in implementing DSO and software factories that apply advanced concepts to be more competitive and secure (e.g., Netflix Chaos Monkey and Amazon fast-turnaround live-release deployments).

2.3.4 DSO Measurement and Metrics

Measurement of DSO draws from both traditional Agile (Kupiainen et al., 2015), Lean (Staron et al., 2012) (Poppendieck & Poppendieck, 2013), and flow (Vacanti, 2015) metrics. Measurement objectives include tracking project progress, increasing visibility into complex aspects of development, providing adequate resources, balancing workloads, understanding and improving quality, ensuring adequate testing, and verifying readiness for release (Kupiainen et al., 2015). This section includes descriptions of adaptations of metrics common in DSO.

Using analysis surveys completed by DevOps subject matter experts (SMEs) the DORA (Forsgren et al., 2018) identified four key metrics associated with software development and delivery performance. Two metrics relate to tempo, two to stability, and one to reliability.

- **Deployment frequency** is the frequency of an organization's successful releases. Because different organizations define *release* differently, deployment frequency might measure how frequently code is deployed to preproduction staging, to production, or to end consumers. Higher frequency is considered better.
- **Mean lead time from commit to deploy** is the mean lead time for change or the average time required for a commit to reach production. Short mean lead times enable engineering and management to determine that the post-code production process is healthy and likely could support a sudden increase of requests. This metric, like deployment frequency, is a measure of software delivery speed.
- **Mean time to recover (MTTR)**[1], aka mean time to restore, is the average duration in time required to restore service after an unanticipated issue or outage. Short recovery times are enabled by rigorous monitoring, full configuration control, infrastructure as code, and automation that enables a prompt roll back to a stable system. Shorter outage durations and recovery times are better.
- **Change failure rate** is a percentage that measures the frequency at which changes to the production system result in a problem including rollbacks, patches, and failed deployments. A lower change failure rate is better and indicates the production process is effective. Higher rates indicate that developer time is spent on rework rather than new value.

The General Services Administration (GSA) provides a larger set of metrics for measuring the success of implementing DSO (GSA, 2021)These high-value metrics include deployment frequency, change in lead time, change in volume, change in failure rate, mean time to restore, availability, customer issue volume, customer issue resolution time, time to value, time to authorization to operate (ATO), and time to patch vulnerabilities.

The PSM issued three framework documents for measuring continuous iterative development (CID). *PSM CID Measurement Framework Part 1* describes the concepts and definitions (Jones et al., 2020b), *PSM CID Measurement Framework Part 2* addresses measurement specifications and enterprise measurement (Jones et al., 2020a), and *PSM CID Measurement Framework Part 3* addresses technical debt (Jones et al., 2021).

[1] https://en.wikipedia.org/wiki/Mean_time_to_recovery

3 Our Research

In this section, we describe our research objectives, approach, workflow, and early results.

3.1 Objectives

To demonstrate the feasibility of automated continuous measurement and estimation we simulated a software project using synthetic data and a prototype instrumented DSO pipeline (Raffo, 2004; Abdel-Hamid et al., 1991). In the demonstration, we focused on a subset of DoD PM information needs, leaving a more comprehensive effort for future work. We also focused on projections for satisfying the requirements for coordination dates such as the minimum viable product (MVP) or minimum viable capability release (MVCR). To validate our work, we used quarterly advisory review panel sessions (QuARPs) involving DoD PMs and other SMEs.

Our long-term research goal is to improve the support of PM decision making. The short-term objective was to explore the subject for gaps, needs, and research opportunities. Thus, a successful project would lead to more focused follow-on work. With all these objectives in mind, the research team posed a number of related questions to explore, including the following:

- What information gaps do DoD PMs have with DevOps-related projects?
- What program information is needed for prediction and actionable decisions in this environment?
- What data supports answering those questions?
- What data can we gather to support real-time reports and analysis?
- How should the data be joined, transformed, and labeled to retain the context?
- What algorithms should we use to develop models and indicators?
- How should we present indicators to decision makers?

The above questions can be binned into three ideas that guide our research:

- what a PM needs to know from a software CI/CD pipeline
- how progress against goals can be measured using this information
- how the information should be presented

3.2 Approach

Our approach to this research included the following steps:

- Select key program management scenarios.
- Construct a prototype pipelines
- Hypothesize performance indicators.
- Prototype pipelines of and collection of data
- Predict performance using synthetic data.
- Create prototype dashboards from scenarios
- Validate dashboards with Subject Matter Experts

Because entered this research with assumptions, our SME proved invaluable challenging, validating, and elaborating on use cases and workarounds. They guided us to focus on percent complete, predictions of capability delivery dates with the status quo, and predictions of capability delivery dates with program interventions.

We selected SMEs who had significant responsibilities in the DoD and defense industrial base in areas such as program management, DSO consulting, and government policy. Although selecting SMEs risked introducing bias, the benefit was a small group with whom we could engage in deeper discussions. We constructed a demonstration DSO pipeline with instrumentation points for prototyping data collection and storage. We reviewed this pipeline with our SMEs to verify that it addressed their concerns.

To make decisions, a decision maker must have information about the scenarios. We borrowed indicators typical of earned value (*Department of Defense Earned Value Management Interpretation Guide*, 2018) and earned schedule (Lipke, 2003) management and validated these indicators with the SMEs.

Based on information needs, we explored the prototype pipelines and other data sources. This helped us identify data sources and reason how to collect the data with the sufficient context to construct the indicators.

Using actual data was impractical because of the limited time available for completing our work. Instead, we generated synthetic data, which was suitable for our purposes and provided additional benefits. The purpose of the simulation was to demonstrate that our data could be stored and that our data storage models would be suitable for producing the desired indicator.

We began with a hypothetical project. We separated the work items into capabilities, features, and stories to develop a reference roadmap and work breakdown structure (WBS) and distributed work among two hypothetical teams Next, we added an artificial estimate for direct effort to each story. We approximated duration as proportional to effort and parameterized the variation in the actual effort required. We used a nominal team load and effort calendar to map the beginning and end of development for each component to an initial estimated plan schedule and an actual (simulated) schedule.

We simulated the flow of stories through our pipelines to model data collection and migrated the data into a database. We then extracted data from the database to build the indicators. We computed the percent complete based on estimated costs and estimated costs of complete work. The results were displayed as an earned schedule. We computed projected scenarios using the *a priori* effort variation and Monte Carlo to estimate a range of completion dates. We then demonstrated the simulations and resulting indicators to our SMEs for their review.

3.3 Workflow

The collection of data throughout the design, development, and operation a system provides the people involved in these processes with situational awareness and actionable information.

Information from processes and tooling can be captured from the early stages of planning, throughout the execution of the system, and finally from the environment in which the system operates (in a post-deployment scenario). Figure 1 displays the different phases of the system lifecycle and suggests types of data that could be collected along the way.

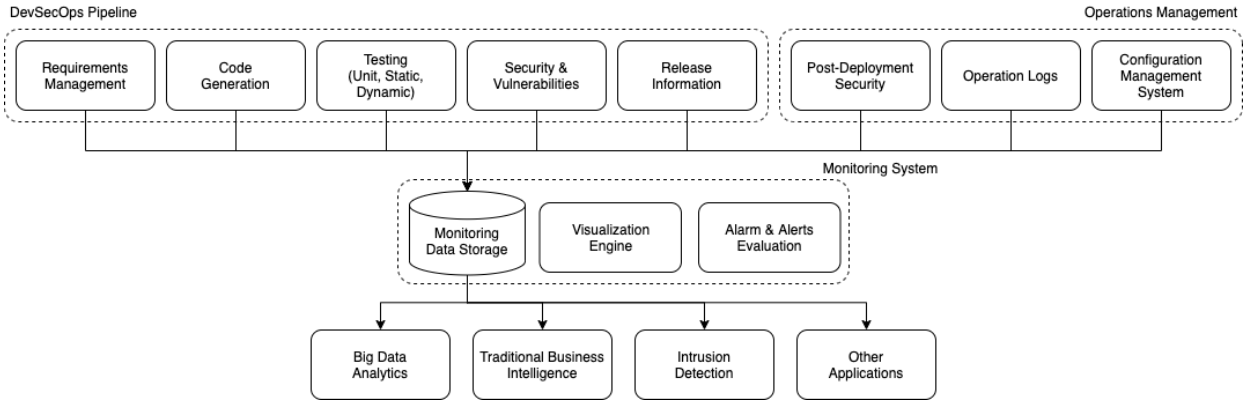


Figure 1: DSO Pipeline and Data Storage

Recorded requirements can be monitored from the planning stages of the system through the development phases to ensure that features are implemented according to the original plan. Because specifications may change along the way, changes to these requirements can and should be incorporated to tell a complete story and indicate the reasons why modifications in the implementation are necessary.

As the design and architecture phases begin, these requirements take shape and—based on architectural principles—are turned into a "skeleton" that will guide high-level feature generation. Code is then generated to implement the features proposed by the architecture and is then refined by epics and stories.

At the same time, artifacts enter a version control system and start flowing through a CI framework that allows data to be captured, such as code style, quality, and security. This data can be inspected and discovered by a linting and a static analysis process.

In the CI framework, the system is built and tested by a dynamic analysis process that evaluates the quality and security of the built system and its dependencies, which were detected in the build process. The data collected from this phase is extremely valuable to the teams involved in feature implementation because it guides them through fixing issues and minimizing risk. This data is also useful for teams managing resources and following the cost and schedule because it (1) informs them about the efficacy of the development team during implementation and (2) helps forecast estimated dates and the overall cost of completion.

As development teams release system versions and move them to staging environments—and finally production—all the data about post-deployment issues and system utilization can also be captured to inform operation teams about resource utilization and system growth.

There is so much data that anyone who monitors it can feel overwhelmed by its volume and what it covers. That is why it is important to introduce mechanisms that reduce mental analysis, make good use of human cognitive capabilities, and allow people to form faster insights. Stakeholders need information that helps them answer questions about system planning, development, and operation. Understanding their needs enables us to create conditions for better sustainment, faster problem solving, and increased security.

In this study, our teams analyzed data captured in many of the different phases described above, for both real and simulated projects. Our research team wanted to answer questions that might have a significant impact in the development and operation of the system, and they chose metrics based on their analysis. Once we defined metrics, our developers introduced means to capture and store the data supporting to those metrics and generated visualizations that could make the data easier to understand. Based on stakeholders needs, these visualizations we aggregated the metrics into dashboards that now provide full transparency into the development and operation of the system.

The fact that we are paying attention to not one pipeline, but an association of pipelines introduces complexity to capturing and organizing data while understanding (1) the origin of the data and process sequencing and (2) classifying and separating information at presentation time. Such a significant amount of information can be overwhelming and become extremely misleading if the design of the dashboards does not provide enough situational awareness to those consuming the information.

3.3.1 CONOPS

The overall goal of this research is to understand the behavior of the processes in the system development lifecycle (SDLC) by capturing measurements that provide situational awareness of the efficiency of the different parts of the framework and system. However, the complexity introduced by the interactions across multiple frameworks and pipelines can add substantially to how much data is monitored in the different parts of the environment. Teams must be careful when introducing instrumentation to ensure that it provides an accurate view of different paths and considers the right timing for measuring signals.

Error! Reference source not found. is a concept of operations (CONOPS) that illustrates the different phases of the overall process, including planning, implementation, and operation. All phases provide valuable information for monitoring and thus creating an accurate situational awareness model for stakeholders.

Before teams implement the mechanisms described in Section 3.3, they must understand what the organization sponsoring the development of the system expects and what plan has been generated for this process. The initial plan contains estimates of complexity for different system modules as well as forecasts of the schedule and costs involved in each phase. Part of the reporting is generated by comparing this plan with its execution. Because the SDLC is using Agile methodologies, changes to requirements are always welcome and must feed back into estimates at the end of each iteration, making this whole process more dynamic.

Approaching the production of the system, a DSO pipeline—or group of pipelines—provides access to a large amount of information that can be captured directly from the tooling used in the pipelines. Every interaction with the framework allows information for monitoring to become available, such as code style and quality, secure coding practices, results of unit tests, static code analysis, dynamic application analysis, functional testing results, container security testing results, and staging/production environment analysis results.

All of this information should be properly captured and made available to stakeholders through visualizations and dashboards—or alerts and alarms for critical and more urgent events. This approach makes it possible to introduce adjustments to system construction and operation estimates and initiate corrective actions that will generate a positive impact in time to develop or correct system issues.

3.4 Early Results

We presented common PM scenarios and questions that might require measurement to support PM evaluations or decisions. Our intent was to prioritize programmatic needs for immediate focus rather than identify all programmatic needs. We recorded the results of these discussions, categorized the questions as "Status and Projections" and "What if," and summarized the results in Table 1.

Table 1: Program Scenarios

Scenario 1: Status and Projections	Scenario 2: What if?
Will we meet the schedule commitment?	**Can we accept a change?**
Where are we now?	**What if we reduce the scope?**
What is our completion rate?	**What if we add resources?**
• How much actual effort was applied?	• What is the required effort?
• Which items are complete?	• How will our completion rate change?
• Which items remain for each capability?	• How are capability commitments affected?
• What is the percentage complete overall and per capability?	
When will we finish the current work?	**If we add effort, how long will it take?**
• What is the projection for completing the project, including schedule and cost estimates?	• What is the new projection for completing the project?
• What is the projection for when each capability will be fully realized, including schedule and cost estimates?	• What is the new projection for when each capability will be fully realize?

• What is the confidence range of current estimates?	• What is the confidence range?
• What are the completion rates and the amount of estimation bias?	
• What are the rework rates?	

Scenario 1 (Status and Projections) focused on the project's status. Status requires understanding the overall body of work, the specific work complete, and the planned and actual cost and schedule for that work. Of specific interest in CI/CD development is the percentage complete overall and for specific capabilities. After the advisory review panel reviewed the primary scenario, it also wanted projections for schedule and cost at completion for each capability and sets of capabilities. In addition, it also requested credible ranges of cost and schedule. These additions were considered important for making commitments and planning resources.

Scenario 2 (What If?) focused on making decisions about program interventions. Typical interventions include changing priorities, increasing, or decreasing scope, and shifting resources. For each of these interventions, the panel wanted a credible range of estimates before and after the intervention.

Although these are typical PM concerns, having timely information has been problematic because of the following:

1. Information was scattered across different systems.
2. Information across the systems, even if available, was not easily joined.
3. The measurements were seldom at the level needed to answer the necessary questions.

For example, if stories recorded during a sprint could be traced to different capabilities, then the following problems could occur:

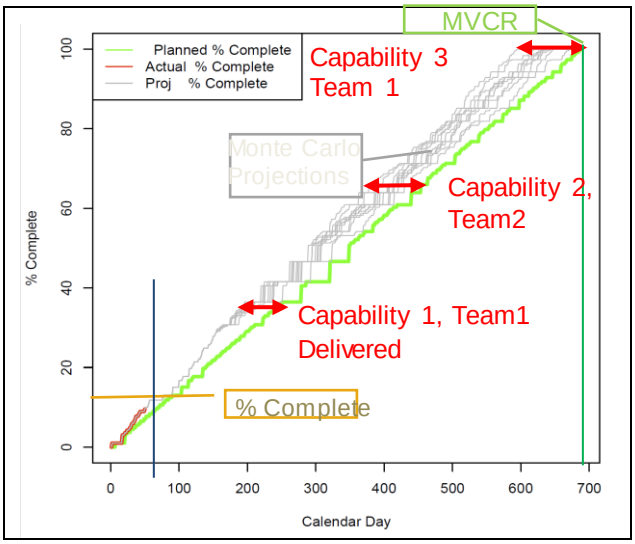
- External mappings would be needed to determine capability completion.
- Effort variances could not be distinguished among capabilities or types of work.
- Variation information would be limited to the sprint rather than to the story level.
- Projections would require detailed knowledge of the planned work order.
- Capability work could be spread across different teams.

The continuous measurement of start and completion times for each story helps resolve some of these problems, but that measurement still relies on fitting information together from the WBS, the master plan, and master schedule. Successful PMs described resolving some of these issues using pivot tables. This is a manual solution to the data join problem, but it does not fully address the unit of measure or analysis problems.

3.4.1 Indicator Displays

We provide prototype indicator for status in Figure 2. These indicator uses data from a simulated project. This indicator shows the plan and projected delivery for each three capabilities developed by two different teams. We structured a representative project into capabilities, features, and stories. We estimated work and sequenced it for execution. We parameterized work package duration with lognormal distribution for actual duration uncertainty and a small underestimation bias was introduced. We separately measured rates and variances for each of the teams. The Planned line represents the rate of progress of the sequential execution of the work packages assuming estimated effort was both available required. The Actual line represents a Monte Carlo simulation though 10% of the estimated duration. The Projection line measures the estimation bias and variation, then it applies the empirical bias and variation to the remainder of the work packages. A number of Monte Carlo simulations then show a range of probable dates, enabling a 90% likelihood estimate. The significance of this simulation is that data is collected automatically from tools using the events defined using Figure 3: Example Sequence Diagram Between Commit and Deploy.

Percentage complete against scheduled cost indicates an earned value. A horizontal line from the work complete to the plan provides a visual representation of days ahead or behind schedule. This serves a similar purpose for earned schedule (Lipke, 2003).



Alternate "what-if" scenarios are addressed in a separate report.

3.4.2 Supporting Metrics

For the purposes of these simulations, we made simplifying assumptions. At this stage, our objectives were to validate the displays with the SMEs and verify the data-collection approach. The following are the simplifying assumptions we made:

- Estimation bias from completed items continues (i.e., the average completion rates will continue to follow the historic trend).
- The estimation error will distribute lognormally.
- Applied effort (cost) is accurately recorded and projected.
- Effort in labor days has been entered for each capability and feature.
- The relative size of stories has been converted into effort days.
- Story effort equals the development duration in labor days.
- A story is worked by only a single developer.
- The stories are worked sequentially in a batch size that does not exceed the number of developers.

Metrics supporting these indicators include the following:

- percent complete (i.e., the estimated cost of all capabilities/estimated cost of capabilities complete)
- completion rates
- schedule projections (i.e., Monte Carlo projected completion date for each sequenced story)

The measures include the following:

- capability, feature, and story estimates in labor days
- story start date

- story completion date from deployment
- story effort (i.e., the development duration in labor days)

3.4.2.1 Tool Sequence and Data Collection

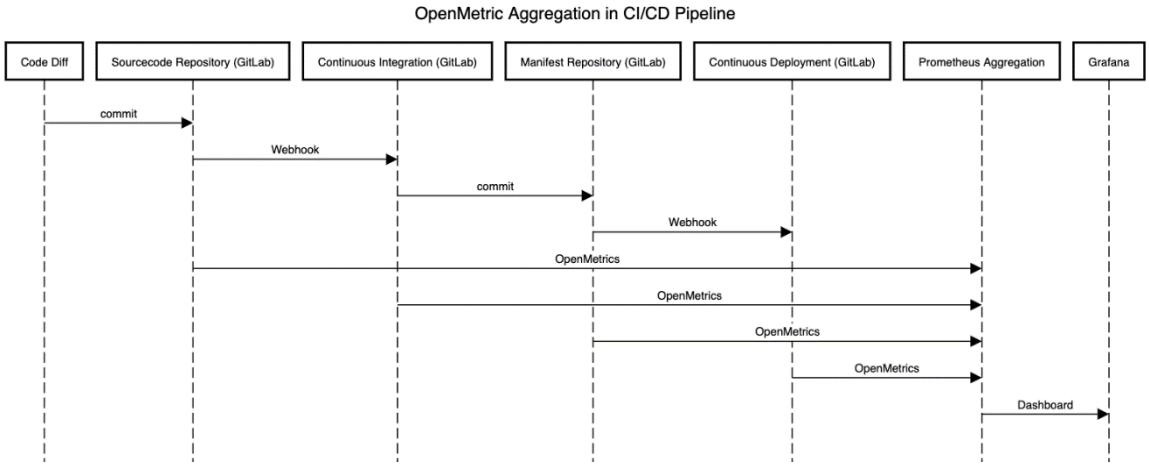


Figure 3: Example Sequence Diagram Between Commit and Deploy

Almost all CI/CD tools offer some sort of collection endpoint, such as an API. These endpoints offer structured data in predefined formats that allow for the collection of metrics regarding builds, health, load, and frequency of use (among others).

Clients are responsible for generating their own metric endpoints for an aggregator to consume. However, the format the tools use to output their data must be standardized. OpenMetrics offers a standard format that displays this data for aggregation engines to consume. This format ensures that metrics are newline separated, with their key:value space separated. This simple format also allows tagging metrics with any number of labels and adding context to each metric where appropriate.

A data aggregation engine like Prometheus can be configured to point to these endpoints for data collection and point to itself to collect data about its metric outputs. Prometheus servers can also be distributed to have a central collection point in the context of several pipelines, which requires aggregated statistics as described in Pipelines of Pipelines (PoPs). (Prometheus can be installed either as a standalone server or within a Kubernetes cluster via Helm or as an Operator.)

DoD customers and the government may leverage techniques, such as Federation, to retrieve and manage aggregate statistics about various vendor pipelines as development takes place.[2]

Once data is flowing into a metric aggregation engine such as Prometheus, tools like Kibana or Grafana can be used to further visualize that data. These visualization tools can be used to create custom dashboards for keeping operators informed in real time about changes in pipelines across a number of DoD projects.

[2] Learn more about Federation on the Prometheus website (https://prometheus.io/docs/prometheus/latest/federation/).

4 Discussion, Open Issues, Next Steps

4.1 Lessons Learned

During this prototyping, we identified several issues that must be overcome to achieve the desired ability to measure schedule and cost progress.

4.1.1 Capability-Based Work Breakdown Structure

The first issue is obvious: The product roadmap needs to be sufficiently developed to estimate the entire scope of work contained in the capability. We are aware that a project's scope will often change, but a nominal scoping and initial estimate are a minimum requirement.. It is critical that traceability of the work package (feature, story, or task) to the capability be maintained throughout the project. It is not, however, required that all stories related to a capability be done to the exclusion of other work or that they be done in a specific order. Nonetheless, the sequencing of features and stories define the up-to-date master plan, which determines the master schedule. Variances from that order must be recognized, as do changes to the work scope. A capability is complete when the last task associated with that capability is released. Although this seems straightforward, rework complicates its use in practice.

Reworking stories or adding defect fix stories confounds this approach. We recommend not counting stories as complete until they are thoroughly tested and released. Defect fixes should be included as separate stories that do not count toward the earned schedule, but that do consume resources. This can be accomplished by adding defect fixes to WBS elements that do not contribute to the earned schedule but that do require flow through the system. This also has the effect of adding cost and schedule, but not adding progress to percent complete.

4.1.2 Connecting the Stories to the WBS

The traceability of stories to the WBS is not directly supported by existing tools. Although workflow is often managed by Jira, some instances use GitLab or other tools. Typically, these workflow management tools do not link directly to the roadmap or WBS. The mapping can be overcome with the careful use of labels. However, labeling requires consistency and is error prone. An alternative is to maintain a separate mapping between WBS elements and their representation in the workflow tool. As long as the mapping is maintained, the story flow can be traced through the DSO tool chain.

The instrumentation of a pipeline versus a pipeline instance poses another problem. Several arbitrary ways exist to organize similar DSO tool chains. Different tools can provide similar functionality but have different interfaces. Some tools might have different orders of execution. The instance must be described sufficiently so that the actual progress of a story is known, and that the data can later be used with its context. In principle, there should be ways for a toolchain to describe itself. Nonetheless, an automated tool chain should be repeatable and stable. For this reason, we characterized a pipeline instance by its activity and by which tool performed or was used in performing that activity. To effectively use automated data collection, events from the example sequence diagram in Figure 3 for capability or analysis must trace the work package to the specific capability and feature.

The biggest gap in data collection is the start of work. Once the story achieves code completion, the automation accurately tracks progress, including rework. However, designating the start of work can be problematic. Currently, we rely on an entry to the workflow management tool as the start and entry to the DSO deployment tool for completion.

4.1.3 Data Warehouse vs. Data Lake

We considered using both a data lake and a data warehouse in our design. The primary difference is that the data lake follows an extract, load, transform model, while the data warehouse follows an extract, transform, load model. Both begin by extracting data from the system. However, while the data lake loads the data into storage, the data warehouse transforms the data by performing logical joins and adding related contextual information prior to loading it into storage.

Using the data warehouse, data can be retrieved after it is loaded and instantly be used to build predefined indicators. The warehouse is efficient because the transformation is applied only once, and the structure can be tuned to support the desired indicators. The drawback is that support of other indicators or uses can be inefficient and cumbersome. Nonetheless, designing the warehouse requires forethought into the required context that will be needed. If this context was not stored or is not accessible, the indicator may not be possible to build.

The data lake, on the other hand, delays transformation until the data is used. This is inefficient for repeated operations because the transformation must be applied every time the data is used. However, late transformation provides more flexibility to use data for other purposes and new indicators. In practice, a workable approach is to stage the data in a data lake and immediately extract and transform into a warehouse. Although this, is inefficient for storage, it supports both needs for repeated use and research.

4.2 Opportunities for Further Research

In this research, we identified gaps where the state of the practice does not fully support the needs of defense acquisition. Although some of the gaps apply predominantly to DoD needs, their solution has more general application for all organizations. Additional gaps remain; for example, a recent DoD memorandum that addressed continuous authority to operate (cATO) (McKeown, 2021) states the following:

- "Service providers will continuously monitor and assess all of the security controls within the information system's security baseline, including common controls.
- Automated monitoring should be as near real time as feasible.
- For cATO, all security controls will need to be fed into a system-level dashboard view, providing a real time and robust mechanism for AOs to view the environment.

Automation of data collection from DSO pipelines promises to address this and other information needs. We foresee future research that includes the following:

- modeling parametric cost estimation as the program evolves
- extension to software factories and multiple interacting pipelines
- inclusion of quality, rework, and technical debt in management goals
- modeling cybersecurity authorization and risk

5 Summary

In our review of DSO metrics practice, we found limited integration of DSO measurement into program management decisions. Identifying measures, validating measures, and providing a supporting infrastructure remain largely unexplored.

This research focuses on improving program management decision making by improving the fidelity and frequency of program performance metrics and indicators, including information needs, what to measure, and how to display the information.

SMEs provided the research team with key program management scenarios to focus our research. We created prototype pipelines to provide a frame of reference for generating candidate indicators of program performance. Using synthetic data, we simulated software development activity. We used the data to build indicators that we validated with SMEs. The overall workflow that we created and captured provides a unique conceptual view of how data can be extracted, stored, and reported from an Agile and DSO pipeline.

A year into this research project, we have several lessons that are worth sharing:

- **Adopt a capability based WBS**. The most fundamental information an organization's leadership wants to know is, "When will it be done?" In a DSO environment, *done* is measured by delivered capabilities; therefore, aligning a WBS to capabilities is an essential first step.
- **Connect engineering artifacts (e.g., stories) to the WBS and associated work packages**. When performance indicators reveal failure to meet the plan, PMs then ask the question "Why not?" To drill down into the data and identify the source of the discrepancy, the cost and schedule targets must align to engineering activities, subsystems, or even individual components.
- **Establish a robust analysis capability in conjunction with creating and maintaining a sufficient data storage system**. The types of analyses and robustness of reporting drive the data storage requirements. The data to be collected and stored drives data infrastructure design considerations. The information needs of the organization drive data warehousing and data lake design options.

As this research continues, it will focus on refining and improving the collection, storage, and reporting of project performance data that is most needed program management. We identified important areas, but we did not include them in the scope of this research. These areas pose great challenges and include parametric cost estimation modeling; collection tooling and application programming interface (APIs); quality, rework, and technical debt; and cybersecurity. While each of these is significant in its own way, our research is tackling the challenges associated with integrating software factories and multiple pipelines in the upcoming year.

6 References

Abdel-Hamid, Tarek and Madnick, S. E. (1991). *Software Project Dynamics: An Integrated Approach*. River, NJ United States: Prentice-Hall, Inc.

Brady, S., & Rice, C. (2020). *Software Acquisition Pathway Interim Policy and Procedures Training*. Retrieved from https://www.dau.edu/Lists/Events/Attachments/193/SW Acq Policy Training_3.17.20.pdf

CSO DoD. (2021a). *DoD Enterprise DevSecOps Fundamentals Playbook*. Retrieved from https://software.af.mil/wp-content/uploads/2021/05/DoD-Enterprise-DevSecOps-2.0-Playbook.pdf

CSO DoD. (2021b). DoD Enterprise DevSecOps Reference Design. Retrieved from https://dodcio.defense.gov/Portals/0/Documents/Library/DevSecOpsReferenceDesign.pdf

Defense', 'Department of. (2019). *DEPARTMENT OF DEFENSE Agile Metrics Guide*.

Department of Defense Earned Value Management Interpretation Guide. (2018). Retrieved from https://acqnotes.com/wp-content/uploads/2014/09/DoD-Earned-Value-Management-Interpretation-Guide-Jan-2018.pdf

Forsgren, N., & Humble, J. (2015). DevOps: Profiles in ITSM Performance and Contributing Factors. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.2681906

Forsgren, N., Humble, J., & Kim, G. (2018). *Accelerate*. IT Revolution.

GSA, C. (2021). DevSecOps Guide. Retrieved September 15, 2021, from https://tech.gsa.gov/guides/dev_sec_ops_guide/

IEEE. (2021). IEEE Standard for DevOps: Building Reliable and Secure Systems Including Application Build, Package, and Deployment. *IEEE Std 2675-2021*. https://doi.org/doi: 10.1109/IEEESTD.2021.9415476

Jones, C. L., Draper, G., Golaz, B., Martin, L., & Janusz, P. (2020a). *Practical Software and Systems Measurement Continuous Iterative Development Measurement Framework Part 3: Software Assurance and Technical Debt*.

Jones, C. L., Draper, G., Golaz, B., Martin, L., & Janusz, P. (2020b). *Practical Software and Systems Measurement Continuous Iterative Development Measurement Framework PSM Continuous Iterative Development Measurement Framework*. Washington, D.C.

Jones, C. L., Golaz, B., Draper, G., & Janusz, P. (2021). *Practical Software and Systems Measurement Continuous Iterative Development Measurement Framework Part 2 : Measurement Specifications and Enterprise Measures*.

Kupiainen, E., Mäntylä, M. V., & Itkonen, J. (2015). Using metrics in Agile and Lean software development - A systematic literature review of industrial studies. *Information and Software Technology*, *62*(1), 143–163. https://doi.org/10.1016/j.infsof.2015.02.005

Lipke, W. H. (2003). Schedule is Different. *The Measurable News*, *2*, 31–34. Retrieved from http://www.pmi-cpm.org/members/library/Schedule Is Different.lipke.pdf

Mallouli, W., Cavalli, A. R., & Bagnato, A. (2020). Metrics-driven DevSecOps, (Icsoft), 228–233. https://doi.org/10.5220/0009889602280233

McKeown, D. (DoD S. I. S. O. (2021). *Memorandum for Senior Pentagon Leadership: Continuous Authorization to Operate (cATO)*. Office of the Secretary of Defense. Retrieved from https://dodcio.defense.gov/Portals/0/Documents/Library/20220204-cATO-memo.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 (supporting information).

Morales, J., Turner, R., Miller, S., Place, P., & Shepard, D. J. (2020). *Guide to Implementing DevSecOps for a System of Systems in Highly Regulated Environments* (No. CMU/SEI-2020-TR-002).

Poppendieck, M., & Poppendieck, T. (2013). *Lean Software Development: an Agile Toolkit*. Boston: Addison-Wesley.

Prates, L., Faustino, J., Silva, M., & Pereira, R. (2019). DevSecOps metrics. *Lecture Notes in Business Information Processing*, *359*, 77–90. https://doi.org/10.1007/978-3-030-29608-7_7

Raffo, D. M. (2004). Using software process simulation to assess the impact of IV&V activities. In *ICSE 2004* (pp. 197–205). https://doi.org/10.1049/ic:20040459

Staron, M., Meding, W., & Palm, K. (2012). Release Readiness Indicator for Mature Agile and Lean Software Development Projects. *Lecture Notes in Business Information Processing*, *111 LNBIP*, 93–107. https://doi.org/10.1007/978-3-642-30350-0_7

"Under Secretary of Defense" DoD. (2020). Software Acquisition Pathway Interim Policy and Procedures. US DoD USA002825-19.

Vacanti, D. S. (2015). *Actionable Agile Metrics for Predictability: An Introduction*. ActionableAgile Press. Retrieved from https://leanpub.com/actionableagilemetrics

Vassallo, C., Proksch, S., Gall, H. C., & Di Penta, M. (2019). Automated Reporting of Anti-Patterns and Decay in Continuous Integration. *Proceedings - International Conference on Software Engineering*, *2019-May*, 105–115. https://doi.org/10.1109/ICSE.2019.00028

Zein, O. (2010). Roles, responsibilities, and skills in program management. In *2010 PMI Global Congress Proceedings – EMEA*. Retrieved from http://www.pmi.org/learning/library/roles-responsibilities-skills-program-management-6799

7 Biographies

William Nichols is a senior researcher at the SEI with a PhD in Physics. His experience includes data acquisition, scientific software development, nuclear engineering, and software project management. While at the SEI he has coached software project management and published research on using software metrics for project and program management.

Hasan Yasar is the Technical Director of the Continuous Deployment of Capability group in the SSD Division of the Software Engineering Institute. Hasan leads an engineering group to enable, accelerate, and assure transformation at the speed of relevance by leveraging DevSecOps, Agile, Lean AI/ML, and other emerging technologies to create a Smart Software Platform/Pipeline. Hasan has more than 25 years' experience as a senior security engineer, software engineer, software architect, and manager in all phases of secure software development and information modeling processes.

Chris Miller is a senior engineer at SEI

Luiz Antunes is a member of the technical staff at the SEI. With a background in Distributed Computing, Information Architecture & Visualization and HCI, he is responsible for the creation of artifacts that generate visual analytics from large data sets. Prior to joining the SEI, he developed software for

the utilities sector in South America and later acted as one of the architects of the CoMotion data visualization and collaboration suite at General Dynamics. He holds a B.S. in Civil Engineering and a M.Sc. in Hydraulic Engineering from the State University of Campinas, Brazil."

Robert McCarthy is at Member of the Technical Staff at SEI supporting DevSecOps engineering.