Carnegie Mellon University
Software Engineering Institute

Automated Data for DevSecOps

William Richard Nichols wrn@sei.cmu.edu412-268-1727
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

Table of Contents

1 Abstract

Automation in DevSecOps transforms the practice of building, deploying, and managing software intensive programs. Although automation supports continuous deployment and rapid builds, manual collection of information delays program status metrics and the decision they are intended to inform by weeks. The emerging DevSecOps metrics such as deployment rates and lead times provide insight to how the software development is progressing but fall short to in terms 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 DevSecOps Pipeline and the pipeline's supporting environment continuous measurement of status, identification of emerging risks, and probabilistic projections is possible and practical. This paper discusses research on the information modeling, measurement, metrics, and indicators necessary to establish a continuous Program control capability which can keep pace with DevSecOps management needs. The importance of interactive visualization dashboards targeted to addressing program information needs is discussed. We will also address gaps in the current state of the practice and barriers we have identified. Finally, we present examples we recommend needed future research based on our initial findings.

2 Introduction

DoD software development is now outpacing the ability of program management to exercise program oversight. Software acquisition increasingly involves software development using Continuous Integration/Continuous Delivery (CI/CD) as described in the Software Acquisition Pathway (SAP). The SAP is available to "facilitate rapid and iterative delivery of software capability to the user" (Defense", 2020) and to empower program managers (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 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, relevant data reports can take weeks or months. The actual progress is constantly ahead of program control.

The challenges are numerous and include increasing complexity of software enabled systems, hardware in the loop, and the presence of COTS and/or open-source components. Within this context, acquisition is adapting with the Software Acquisition Pathway. The DoD policy is clear. IT expects program managers to use metrics for planning, control and oversight: "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**."{em 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). The inability of program management to apply lifecycle data analytics to measure program performance, and conduct effective oversight of CI/CD practices, thus prevents the DoD from consistently and effectively adopting modern practices. Realizing the desired benefits of CI/CD requires metrics collected from the modern development pipeline and prediction models derived from that data. This research project examines how we can take advantage of the automation within the modern DevSecOps environment for the benefit of program management.

3 Background

Extensive literature review found that peer reviewed literature is devoid of studies of automated data collection for CI/CD (Prates, Faustino, Silva, & Pereira, 2019). *Although n*on-peer reviewed literature exists, *they either* address operational issues rather than PM issues, *or* limit to a narrow research topic (Vassallo, Proksch, Gall, & Di Penta, 2019) rather than DoD *programmatic* needs. Several sources, PSM, NDIA and INCOSE (Jones, Draper, Golaz, Martin, & Janusz, 2020b), and the DoD (Defense', 2019) recommend metrics for Agile and CI but none have connected to automated collection nor have the metrics been rigorously validated. The situation is similar for DevSecOps with regard to the DORA metrics (Forsgren & Humble, 2015)(Forsgren, Humble, & Kim, 2018). The Defense Innovation Board (DIB) explicitly noted this gap: "In the beginning stages of DoD's transformation to DevSecOps 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, DevSecOps enables Continuous Integration and Continuous Deployment. CI is the automated process by which developers integrate code then build, test, and validate, and deploy new applications. The automation that makes these DevSecOps practices possible in turn spawns a large amount of data as a by-product. This data can be made available to enable 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.

3.1 DSO, Pipelines and Automation

In order to implement automated continuous estimation of software-intensive systems, it is necessary to define what is being measured. The specific measurements depend upon the decisions to be made. For the purposes of this document, we focus on the point of view of Program Managers and the development pipeline as the object of measure.

DevSecOps (DSO) is a software engineering culture and practice that aims at unifying software development (Dev), security (Sec) and operations (Ops) personnel and their practices. The essential concepts of DevSecOps comprise automating, monitoring, and applying security to all activities of software development including , feature planning, bug fixing, feature development, application and support infrastructure builds ant testing, releasing new software, whether that involves: maintaining operational software that supports a user-base, monitoring operational systems for performance and security-related events (CIO (DoD), 2019)

DevSecOps consists of a set of principles and practices which enable better communication and collaboration between relevant stakeholders for the purpose of specifying, developing, and operating software and systems products and services, and continuous improvements in all aspects of the life cycle. (IEEE, 2021)

A pipeline consists of a chain of processing elements arranged so that the output of each element is the input of the next; the name is by analogy to a physical pipeline. The analogy is a weak connection. I.e., there is no requirement for ordered processing or tight coupling. In fact, many pipeline elements use asynchronous messaging and de-coupled processes, such as GitOps. Often the term pipeline is used to specifically describe the 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 to provide performance metrics in a manner that does not distract effort from development work. By eliminating manual data collection activities, it not only reduces the effort associated with performing these tasks it also reduces the opportunity to inject bias into the data. It also provides a continuous data collection and storage capability that can revolutionize the frequency and fidelity of software estimation.

3.2 Programmatic Needs

Program management is usually defined as managing a group of related projects, using specific management techniques, knowledge, and skills. The program manager must work with senior leaders and stakeholders across multiple departments and teams. Decisions are likely to be strategic and connected to the financial calendar. Responsibilities include coordinating resources and outputs across teams rather than with teams in isolation.

Program management responsibilities include strategy, finance, and communication. The overarching purpose is to guide the program to successful outcomes. The program manager responsibilities include (Zein, 2010)

- Manage the program's budget
- Establish High-Level Performance Objectives
- Manage strategy and guide investment decisions
- Define the program governance (controls)
- Plan, monitor, and control the overall program
- Manage risks and issues and taking corrective measurements
- Coordinate the projects and their interdependencies
- Manage and utilize resources across projects
- Manage stakeholders' communication
- Align the deliverables (outputs) to the program's "outcome" with the aid of the business change manager; and
- Daily program management throughout the program life cycle

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

- Baseline and benchmark performance
- Product completion and cost rates with probabilistic cost/schedule projections
- Master plan, master schedule and lead times,
- When work has begun and completed
- Which queues can be bypassed?
- Resource needs and utilization under nominal and "what-if" scenarios

3.3 Prior Work/State of the Practice

3.3.1 Academic Research

Prior work has identified numerous candidate measures and opportunities in the DevSecOps 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. Creating new features, in the machine learning sense, from the raw data to deduce status and improve predictive power is a key challenge.

To demonstrate the feasibility of automated continuous estimation we will simulate software projects (Raffo, 2004)(Abdel-Hamid, Tarek and Madnick, 1991) using synthetic data and an instrumented DevSecOps pipeline. This demonstration will focus on a subset of the DoD PM information needs, leaving a more comprehensive effort for future work. Our planned focus will be on projections for satisfying the requirements for coordination dates such as the Minimum Viable Product (MVP) or Minimum Viable Capability Release (MVCR). We will review and validate work with quarterly advisory review panel sessions (QuARPs) involving DoD program managers and other SMEs.

3.3.2 Government and DevSecOps

One defining characteristic in the DoD is the fact that the environments in which systems operate are highly regulated. Because of this, agencies are not free to simply adopt strategies and frameworks that are found in industry environments. The SEI has written guidance describing special conditions found in these environments, difficulties generated by them and possible solutions to make DSO practices work.

One of the biggest pushes to Agile, DevOps and DevSecOps (DSO) has started since the incorporation of the Chief Software Officer at the USAF. The DoD has since then seen a huge effort towards the internal standardization of a platform with artifacts and processes that may be used across the department agencies. While not being a one-size-fits-all solution, this initiative has promoted the DSO mindset across multiple programs that found now is the right time to implement the DSO practices in those programs. To support these initiatives, the DoD prepared guidance on how to adopt DSO practices and issued different documents to provide ideas of teams and personnel organization, cost and level of effort. Most of the organizations and agencies following guidance from the DoD are currently using these documents, that are listed and described below:

- **DoD Enterprise DevSecOps Playbook** (CSO DoD, 2021) : provides detailed coverage of all the aspects related to the design, development and operation of systems under the DSO lens. Among the topics covered, one will find guidance about shifting a program culture towards DSO, assembling a software factor (SWF), implementing DSO pipelines in a SWF, basic metrics to capture and monitor, orchestration frameworks, and finally suggestions on how to secure the system and its infrastructure.
- **DoD Enterprise DevSecOps Reference Desig**n (CIO (DoD), 2019): preceded the two others mentioned above. This document provides more technical implementation details such as containers vs. virtual machines, DoD centralized artifact repository, and how to organize a DSO pipeline and its environment.

One defining characteristic in the DoD is the fact that the environments in which systems operate are highly regulated. Because of this, agencies are not free to simply adopt strategies and frameworks that are found in industry environments. The SEI has written guidance describing special conditions found in these environments, difficulties generated by them and possible solutions to make DSO practices work (Morales et al., 2020).

3.3.3 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 taken by the industry comes from documents published by consortiums of organizations that have a solid track record in implementing DSO and implementing software factories that apply very advanced concepts to be more competitive and secure (e.g., Netflix Chaos Monkey and Amazon fast-turnaround live-release deployments).

3.3.4 Metrics

Measurement of DevSecOps draws from both traditional Agile (Kupiainen, Mäntylä, & Itkonen, 2015) and Lean (Staron, Meding, & Palm, 2012) (Poppendieck & Poppendieck, 2013) and flow(Vacanti, 2015) metrics. Measurement objectives include tracking of 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 some adaptations of metrics common in DevSecOps.

Using analysis surveys completed by DevOps subject matter experts (SMEs) the DevOps Research Association (DORA)(Forsgren et al., 2018) identified four key metrics associated to software development and delivery performance. Two of them relate to tempo, two to stability, while another was added to measure reliability.

Deployment frequency - The frequency of an organization's successful releases is referred to as deployment frequency. Because different organizations define release differently, deployment frequency may measure how frequently code is deployed preproduction staging, to production or to end customers. Higher frequency is considered better.

- **Mean lead time from commit to deploy** - The Mean lead time for change is the average time required for a commit to reach production. Short mean lead times enable engineering and management to determine that post code production process is healthy and could likely support sudden increase of requests. This metric, like deployment frequency, is a measure of software delivery speed.
- **Mean time to recover** – Also called mean time to restore [**MTTR**)], is the average duration in time required to restore service after an unanticipated issue or outage. Shorter outage durations are better. 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 times are better.
- **Change failure rate** - A change failure rate or percentage measures the frequency that changes to the production system result in a problem, including rollbacks, patches, and failed deployments. Lower change failure rate is better and indicates the production process is effective. Higher rates indicate that developer time spent on rework rather than new value.

The General Services Administration (GSA) , provides a larger set of metrics to measure success at implementing DevSecOps https://tech.gsa.gov/guides/dev_sec_ops_guide/ . The GSA's set of high-value DevSecOps metrics include Deployment Frequency, Change Lead Time, Change Volume, Change Failure Rate, Mean Time to Restore, Availability, Customer issue volume, Customer issue resolution time, Time to value, Time to ATO, Time to Patch vulnerabilities.

The Practical Software Measurement Group (PSM) issued three framework documents for measurement of continuous iterative development (CID). Part 1 of the PSM CID Framework describes the concepts and definitions(Jones et al., 2020b) . Part 2 addresses Measurement Specifications and Enterprise Measurement (Jones, Draper, Golaz, Martin, & Janusz, 2020a) , Part 3 addresses Technical Debt [Jones 2021c](Jones, Golaz, Draper, & Janusz, 2021)

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

4 Our Research

4.1 Objectives

This research was constrained in budget and limited to a one-year execution. Its long-term goal is to improve support to program management decision-making. The short-term objective is to explore the subject for gaps, needs, and research opportunities. A successful project would thus lead to more focused follow-on work. With all these objectives in mind, the research team posed a number of related questions to explore the, including:

- What are Information gaps DoD Program Managers have with DevOps related projects?
- What program information is needed for prediction and actionable decisions in this environment?
- What data supports to 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 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 in three categories:

- What one needs to know
- How progress against the goals can be measured

- How the information should be presented

These three questions guide the output of this research.

4.2 Approach

4.2.1.1 Identify SME for Consultation and Review

Our subject matter experts were not randomly selected. Instead, we identified individuals with significant responsibilities in DoD defense industrial base Program Management, DevSecOps consulting, and government policy. Although selection risks introducing bias, the benefit was a small group with whom we could engage in deeper discussions.

4.2.1.2 Select Key Program Management Scenarios

Although we entered this research with prior beliefs, our SME group was invaluable for elaborating on their use cases and work arounds. Our SME group guided us to focus on percent complete, predictions of capability delivery dates with status quo, and predication of capability delivery dates with program interventions.

4.2.1.3 Construct Prototype Pipeline

We next constructed a demonstration DevSecOps pipeline with instrumentation points to prototype data collection and storage. This pipeline was reviewed with our SME to verify that it was sufficiently representative to address their concerns.

4.2.1.4 Hypothesize Indicators

To make decisions requires that the decision maker have information on 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 with the SME. These indicators then helped analyze the information needed to create them.

4.2.1.5 Prototype Pipeline and Data Collection

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

4.2.1.6 Produce Indicators with Synthetic Data

Actual data was impractical because of the limited time in which to complete 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 indicate. We began with an exemplar project one of us had worked on previously. We separated the work items into capabilities, features, and stories to develop a reference roadmap and work breakdown structure. 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 of and end of development for each component to an initial estimated plan schedule and an actual (simulated) schedule. We then simulated the flow of stories through our pipeline to model data collection and migrate the data into a data base. We then extracted data from the database to build the

indicators. We computed the percent complete based on estimated costs and estimated costs of work complete. 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.

4.2.1.7 Validate with SME

To conclude, we demonstrated the simulations and resulting indicators to our SME for their review.

4.3 Workflow

The collection of data throughout the design, development and operation of a system provides 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.

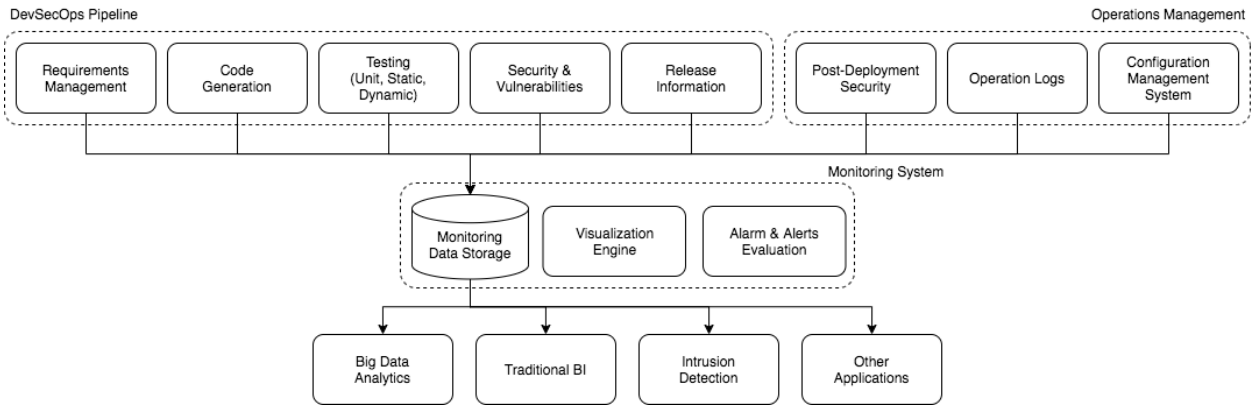


Figure 1: DevSecOps Pipeline and Data Storage

From the planning stages of the system, recorded requirements can be monitored 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 we head into the design and architecture phases, these requirements take shape and—based on architectural principles—turn into a skeleton that will guide the feature generation at a higher-level. Code is then generated to implement the features proposed by the architecture and refined by epics and stories. At this moment, artifacts enter a version control system and start flowing through a continuous integration (CI) framework that allows us to capture data such as code style, quality and security, which can be inspected and discovered by linting and a static analysis process. Still in the CI framework, the system will be built and tested by a dynamic analysis process that will even evaluate the quality and security of both the built system and its dependencies, as detected in the build process. The data coming from this phase is extremely valuable to the teams involved in the implementation, as

it will guide them through a path to fix issues and minimize risk. It is also useful for teams managing resources and following the cost and schedule, as this will inform them about the efficacy of the development team during implementation and help forecast estimated dates and overall cost for completion.

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

With all this data available, it is easy for anyone who monitors it to feel overwhelmed by the amount and coverage of the information. That is the reason why introducing mechanisms that reduce the load of mental analysis, make good use of human cognitive capabilities and allow people to have faster insights is adequate here. By understanding the needs of stakeholders to receive information that allows them to get answers to their questions about system planning, development and operation, we are creating 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. A team of analysts attempted to answer questions that may have a large impact in the development and operation of the system, and chose metrics based on their analysis. Once these metrics were defined, developers introduced means to capture and store the data related to them and then proceeded to generate visualizations that could make the data easier to understand. Based on the stakeholders needs these visualizations have been aggregated into dashboards that now provide full transparency into the development and operation of the system.

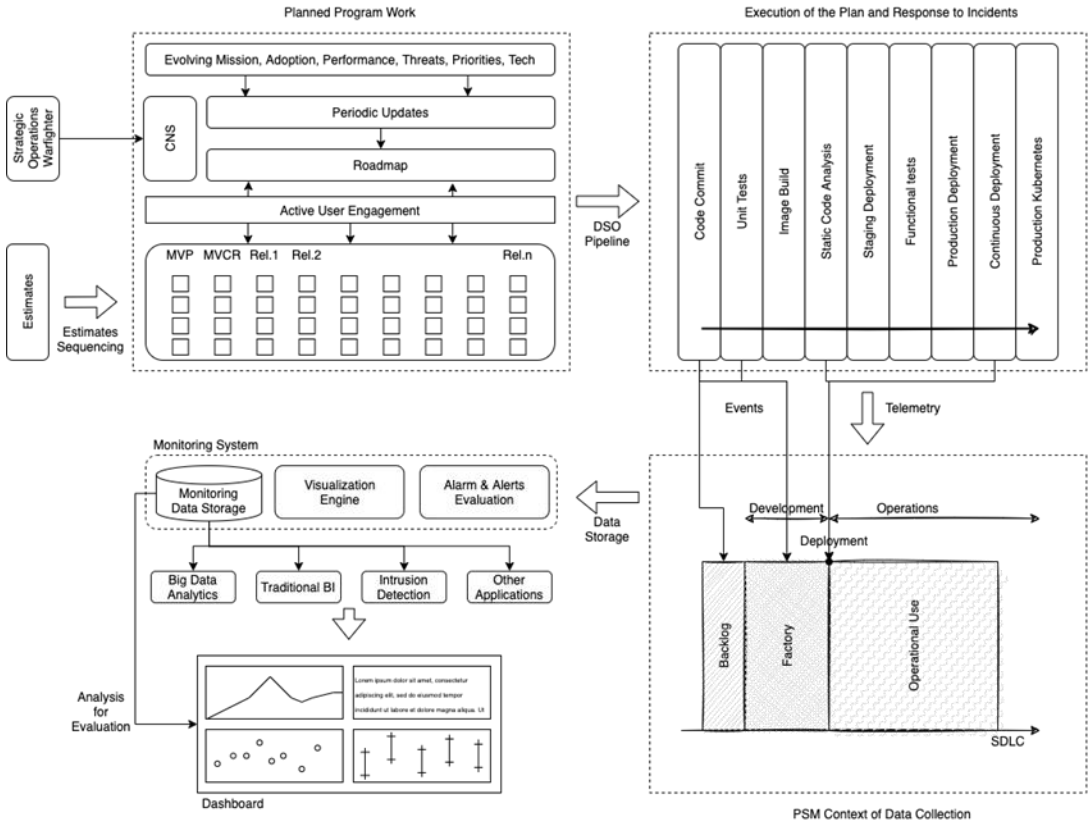
The fact that we are paying attention not to one pipeline, but to an association of pipelines, introduces more complexity for both capturing and organizing data while understanding their origin and process sequencing, and also classifying and separating information at presentation time. Such amount of information can be overwhelming and become extremely misleading if the design of the dashboards does not provide enough situational awareness to the information consumers.

4.3.1 CONOPS

The idea for the Automated Continuous Estimation in a pipeline-of-pipelines (PoPs) context is roughly an amplification of the same process implemented for a single pipeline. The overall goal is to be able to understand the behavior of the processes in the SDLC by capturing measurements that will 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 a substantial load to how the different parts of the environment are monitored. Teams must be careful when introducing instrumentation, so an accurate view of different paths is provided and considers the right timing for measuring signals.

Figure 2 illustrates the different phases of the whole process, including planning, implementation and operation. All of them provide valuable information for monitoring and creating an accurate situational awareness model for stakeholders.

Figure 2: Concept of Operations



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

As we head into the production of the system, a DSO pipeline—or group of pipelines—will provide access to a large amount of information that can be captured directly from the tooling used in the pipelines. At every interaction with the framework, information for monitoring becomes 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, Staging / Production environment analysis result.

All 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. By doing this, it will be possible to introduce adjustments to the initial system construction and operation estimates and also corrective actions that will generate a positive impact in the time for developing or correcting issues in the system.

4.4 Early Results

We presented scenarios and questions for which program managers might need measurement to support evaluations or decisions. The intent was to prioritize programmatic needs for immediate focus rather than to identify all programmatic needs. We recorded the results from these discussions, categorize the questions as "Status and Projections", and "What if" and summarize the results in Table 1: Program Scenarios.

Table 1: Program Scenarios

Scenario 1: Status and Projections	Scenario 2: "What if?"
Will we make the schedule commitment?	**Can we accept a change?**
Where are we now?	**What if we reduce 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 a capability?	• How are capability commitments affected?
• % Complete overall/capability?	
When will we finish current work?	**If we add effort, how long will it take?**
• Projection to complete (schedule/cost)	• New projection to complete
• Projection to complete capability (schedule/cost)	• New projections for capability complete
• Confidence range of estimates	• Confidence range
• Completion rates and estimation bias	
• Rework rates	

The primary scenario considered current status. Status requires understanding the overall body of work, the specific work complete, the planned and actual cost and schedule for that work. Of specific interest in a CI/CD development was a percentage complete overall and for specific capabilities. In addition to status, the panel also wanted projections for schedule and cost at complete for capability and sets of capability. In addition, credible ranges of cost and schedule were requested. These were considered important for making commitments and planning resources.

The second scenario involved decisions for program interventions. Typical interventions include changing priorities, adding or removing 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 concerns of program management, having timely information had been problematic because

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 an atomic level to answer the necessary questions.

For example, if stories recorded during a sprint represented traced to different capabilities,

- 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 the story level.
- projections require detailed knowledge of planned work order
- capability work could be spread across different teams.

Continuous measurement of start and completion times for each story help to resolve some of these problems, but still rely upon joining information from the work breakdown structure, the master plan, and master schedule. Successful program managers described resolving some of these issues using pivot tables. This is provided a manual solution to the data join problem but did not fully address the unit of measure or analysis problems.

4.4.1 Indicator Displays

Prototype indicators are provided for status in Figure 3 and what if planning in Figure 4.

These indicators use data from a simulated project. A representative project was structured into capabilities, features, and stories. Work was estimated and sequenced for execution. Work package duration was parameterized with log normal distrubtion for actual duration uncertainty and a small underestimation bias was introduced. The planned line represents the rate of progress sequential execution of the work packages assuming the nominal effort was required. The Actual represents a Monte Carlo simulation for 10% of the estimated effort. The Projection measures the estimation bias and variation, then 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, from which a 90% likelihood can be estimated.

The percentage complete serves effectively as an earned value. A horizontal line from the work complete to the plan provides a visual representation of schedule days ahead or schedule slip. This serves a similar purpose to earned schedule [Lipke 2003]

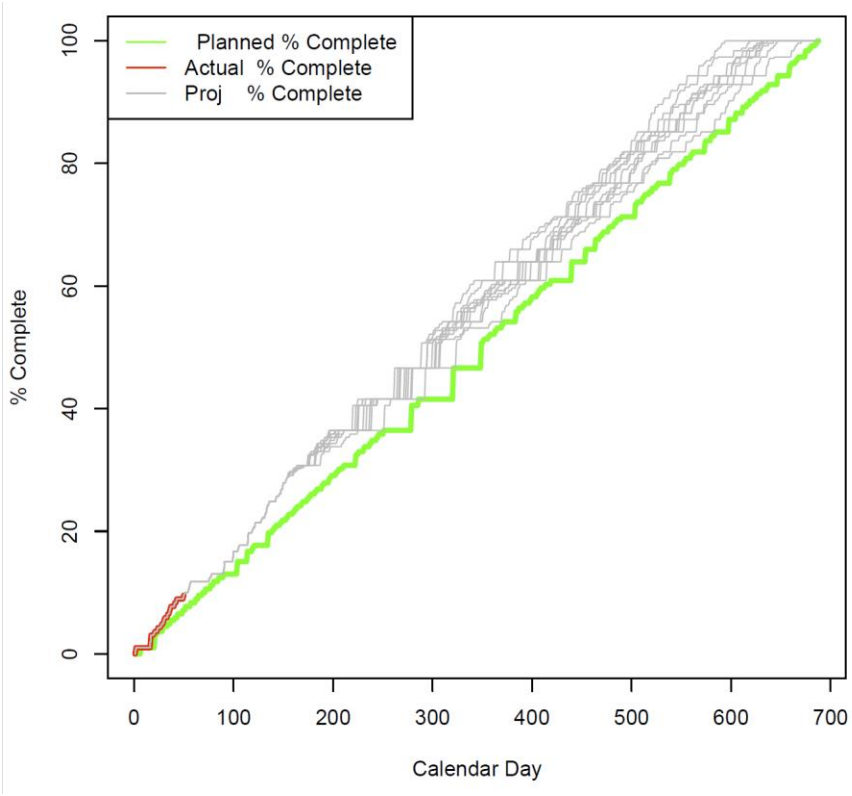


Figure 3: Planned, Actual, and Projected Completion

We next presented the SME with Figure 4, showing the effects moving half the work to a second team and rebalancing work as needed. This represents one of many possible Program Interventions. Although we recognize that this is an oversimplification, the presentation was adequate for the purpose of obtaining SME validation for the requirement. The SME agreed that a similar graphic to compare the current likely outcomes with a probabilistic range of completion dates after an intervention is the need.

Other interventions not included in this paper were to add or remove capabilities, shift commitment dates. It is a straight forward matter to indicate the completion of specific capabilities along the time line.

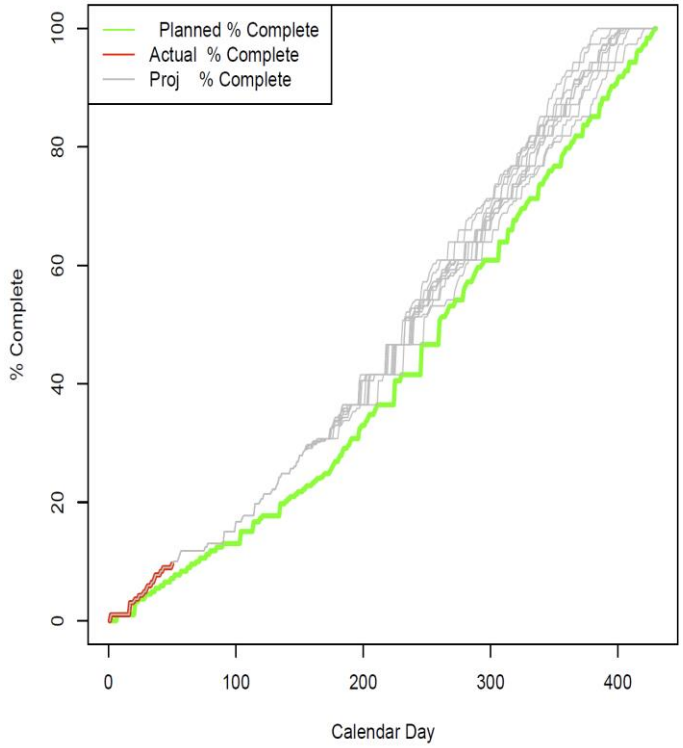


Figure 4: "What if" scenario for completion

4.4.2 Supporting Metrics

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

- Estimation bias from completed items continues, that is 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
- 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 number of developers

metrics supporting these indicators include

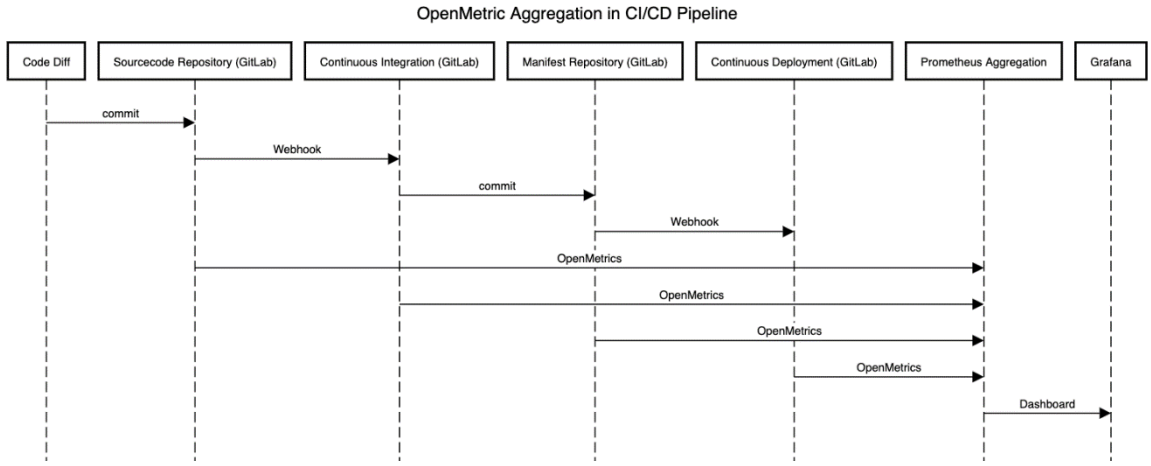
- Percent Complete – (the estimated cost of all capabilities)/(estimated cost of capabilities complete)
- Completion Rates
- Schedule Projections – Monte Carlo projected completion date for each sequenced story

The measures include

- capability, feature, and story estimates in labor days
- story start date
- story completion date from deployment
- story effort equals the development duration in labor days

4.4.2.1 Tool Sequence and Data Collection

Figure 5: Example sequence diagram between commit and deploy



Collection endpoints are offered by almost all significant CI/CD tools. These endpoints offer structured data in pre-defined formats which allow for the collection of metrics regarding builds, health, load and frequency of use among others.

While clients are responsible for generating their own metric endpoints for an aggregator to consume. However, the format in which the tools output their data must be standardized. OpenMetrics offers a standard format to display 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 provides for tagging metrics with any number of labels, adding context to each metric where appropriate.

One such data aggregation engine like Prometheus can be configured to point to these endpoints for data collection, and even 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 require aggregated statistics as described in 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 in order to retrieve and manage aggregate statistics about various vendor pipelines as development efforts take place. Federation | Prometheus

Once data is flowing into a metric aggregation engine such as Prometheus, further visualization of that data can be done using tools like Kibana or Grafana. These visualization tools allow for the creation of custom dashboards which can be used to keep operators informed in real-time about changes in pipelines across a number of DoD projects.

5 Discussion, Open Issues, Next Steps

5.1 Lessons Learned

5.1.1 Lessons for Practice

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

5.1.1.1 Capability Based Work Breakdown Structure

The first issue is to state the obvious, that the product roadmap needs to be sufficiently developed to estimate the entire scope of work contained in the capability. We are aware that scope will often change, but a nominal scoping and initial estimate are a minimum requirement. We further found that successful practitioners tracked work using a work breakdown structure that included capabilities further subdivided into features (or epics) and stories. It is critical that traceability of the work package (story or feature) to the capability be maintained throughout. It is not 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 defines the up-to-date master plan which determines the master schedule. Capability complete occurs when the last story associated with a capability has been released. Although this seems straight forward, rework complicates it's use in practice.

Reworking stories or adding defect fix stories confounds this approach. We recommend that stories not be counted as complete until they have been 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 has the effect of adding cost and schedule, but not adding to percent complete.

5.1.1.2 Connecting the Stories to the WBS

The traceability of stories to the work breakdown structure 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 to not link directly to the roadmap, or work breakdown structure. The mapping can be overcome with careful use of labels. However, labeling requires consistency and is error prone. An alternative is to maintain a separate mapping between the work breakdown elements and their representation in the workflow tool. As long as the mapping is maintained, the story flow can be traced through the DevSecOps tool chain.

5.1.1.3 The instrumentation of a pipeline vs a pipeline instance poses another problem. There exist several arbitrary ways to organize similar DevSecOps tool chains. Different tools may provide similar functionality but have different interfaces. Some tools may have different orders of execution. It is necessary that the instance be described sufficiently so that the actual progress of

a story is known, and that the data can later be used with context. In principle, there should be ways for a toolchain to self-describe. Nonetheless, an automated tool chain should be repeatable and stable. For this reason, we characterized a pipeline instance by activity and by which tool performed or was used in the performance of that activity. To effectively use automated data collection events from the example sequence diagram in Tool Sequence and Data Collection

Figure 5 for capability or analysis requires tracing the work package to the speciric capability and feature.

The biggest gap in data collection is the start of work. Once the story achieves code complete, 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 for start, and the DSO deployment tool for completion.

5.1.1.4 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 Warehouse follows an Extract, Transform, Load one. This means that both begin by extracting data from the system. While the Data Lake loads the data into storage, the Warehouse transforms the data by performing logical joins and adding related contextual information prior loading into the storage. Data can then be retrieved and instantly used to build indicators. This is efficient because the transformation is applied only once, and structure can be tuned to support the desired indicators. The drawback is that support of other indicators or uses can be inefficient and cumbersome.

The Data Lake, on the other hand, delays transformation until the data is used. This is inefficient because the transformation must be applied every time the data is used but leaves the flexibility to use data for other purposes. 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. In practice, a workable approach is to stage the data in a data lake and immediately transform into a separate warehouse. This is inefficient for storage, but supports both the needs for repeated use with

5.2 Opportunities for further research

In this research we have identified a number of gaps in which the state of practice does not fully support the needs of defense acquisition. Although some of the gaps are predominantly DoD needs, their solution has more general application.

5.3 Software Factories and Multiple Pipelines

A software factory contains multiple continuous integration / continuous delivery (CI/CD) pipelines, each producing unique artifacts. It is designed for multi-tenancy and automate software production for multiple products.

As illustrated in Figure 6, this software factory contains multiple pipelines, which are equipped with a set of tools, process workflows, scripts, and environments, to produce a set of software deployable artifacts with minimal human intervention. It automates the activities in the develop, build, test, release, and deliver phases. A DoD organization may need multiple pipelines for different types of software systems, such as web applications or embedded systems. (CIO (DoD), 2019)

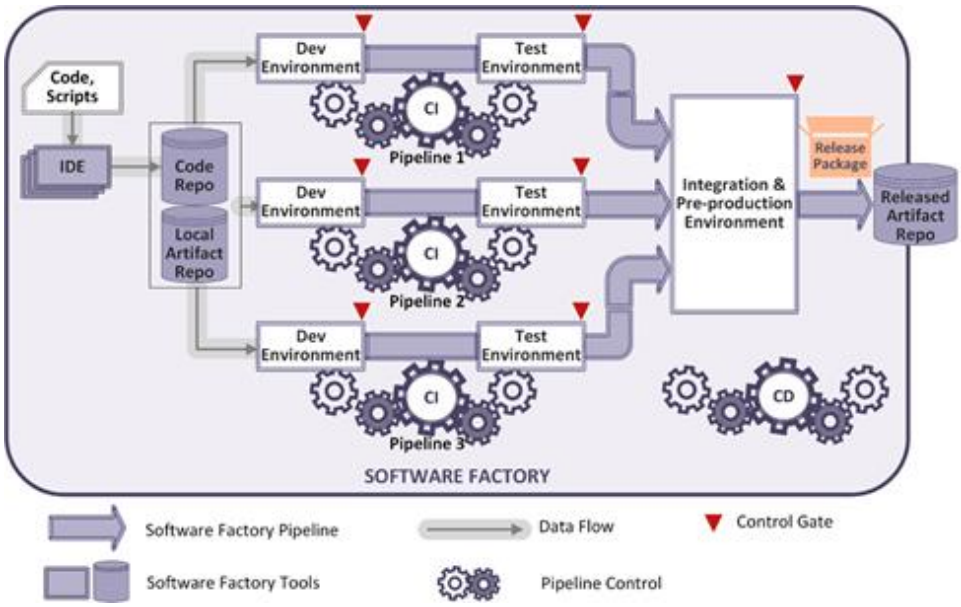


Figure 6:DevSecOps Software Factory (CIO (DoD), 2019)

5.4 Parametric Cost Modeling Evolution

In our ACE research we focused on automating the collection of size data given it is essential to estimating effort and schedule. By tracking size growth and changes in size at greater frequency than ever before, we were to demonstrate the value of continuous estimation. Our research did not venture into estimating the cost drivers and effort multipliers. These parameters often involve subjective assessments of people (e.g., team cohesion), process (e.g., process maturity), project (e.g., development flexibility) and product characteristics (e.g., platform volatility). These parameters pose a future challenge as research efforts turn their focus to fine tuning an automated software estimation capability. The historically labor-intensive Likert scale subjective assessments will need to be replaced by identifying and quantifying objective evidence that can be collected automatically. For example, the 'team cohesion' may be able to be replaced by tracking the number and frequency of developer communications. And process maturity may be updated continuously based in a combination of measures associated with process monitoring and defect containment. As digital engineering continues the drive more and more automation, the historically subjective parameters will be able to be replaced with objective data collected more frequently which will increase the value and fidelity of continuous estimation.

As engineering data becomes more transparent and available as software development projects rely more and more on comprehensive tools suites, there effort tracking remains a hurdle for organizations seeking to improve their cost estimate. At its most basic core, a cost model correlated the estimated software size of the final product to estimate effort. Since effort to create software is the largest cost element associated with producing software, effort is used to estimate cost. As we showed in ACE, we are able to increase the precision and frequency of collecting size data directly from the software development tools suite. Unfortunately, the technology advances enabling the greater precision on size data collection haven not transpired in the collection effort. Across defense industry, people are responsible for tracking their own effort. They submit their effort daily, weekly, or even monthly based on their organizations policies and customer contractual requirements. The effort is generally allocated to defined labor categories associated with the project and type of work performed. The effort is used

by the organization's finance department to invoice the DoD customer per contractual roles and regulations. This paradigm creates a limitation to improving cost estimates. As long as cost accounts and their associated cost strings are used to collect effort data there will be a lower limit on the granularity of effort data.

Additionally, program managers in the pursuit of managing their project move money from one cost string to another to cover unforeseen cost overruns or underruns. This happens most often when cost accounts are very granular. The manager needs to freedom the reallocate resources to be successful. There a several ways a manager or employee can do this but if the reallocation of effort isn't reflected in the employee's effort data submission, improving the fidelity of the effort and therefore the cost of specific engineering activities will not be possible.

5.5 Quality, Rework and Technical Debt

Members of our review panel noted that they currently have limited insight into issues of quality or rework. Practices carried over from traditional agile development were not satisfying the enterprise or program needs for several reasons. First, they usually failed to isolate rework, conflating rework with new work and confounding measures of completion. Second, interpreting defect counts requires consistency in when an issue is counted. This may vary between teams, even with teams. Third, contextual data such as in which component an issue was found, which activity injected the defect, even which iteration injected the defect, the size of the component or of the overall code may not be recorded.

5.6 Cybersecurity

A recent DoD Memorandum (DoD 2021a)_ that addressed Continuous Authority to Operate (cATO)(DoD CIO, 2021) states:

- "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

This memo assumes a heterogeneous environment, expects automation where practicable, and expects dashboards to be available in real time for the decision authority. Because cybersecurity is a risk management activity, the tradeoffs between risk a known weakness may be traded off against the risk of a capability becoming unavailable. We suggest that the cybersecurity risk assessment include an analysis of the capability delivered.

6 Summary

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

This research is focused 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

Subject matter experts provided the research team with key program management scenarios to focus the research. A prototype pipeline was created to provide a frame of reference for generating candidate indicators of program performance. Using synthetic data, we simulated a software development activity. We used the data to build indicators to validate with the subject matter experts. The overall workflow that was 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 there are several lessons worth sharing. One lesson is to adopt a capability based work breakdown structure. The most fundamental information an organization's leadership wants to know is, 'when will it be done'. In a DevSecOps environment, done is measured by delivered capabilities. Therefore, aligning a WBS to capabilities is an essential first step. A second lesson learned is the need to connect engineering artifacts (e.g., stories) to the work breakdown structure and associated work packages. When performance indicators reveal failure to meet plan, their next question will be 'why not?'. In order to drill down into the data and identify the source of the discrepancy, the cost and schedule targets will need to align to engineering activities, subsystems, or even individual components. A third lesson is that in conjunction with establishing a robust analysis capability is the need to create and maintain a data storage system sufficient for the needs. The types of analyses and robustness of reporting will create data storage requirements. Planning what data will be collected and stored will drive data infrastructure design considerations. The information needs of the organization will drive data warehousing and data lake designs options.

As this research continues, it will focus on refining and improving the data collection, storage and reporting of project performance data that is front and foremost needed by DoD Leadership. There some very important areas that our research has deemed identified but not included in the scope of this research but pose great challenges. These are: parametric cost estimation modeling, collection tooling and APIs, quality, rework, and technical debt, and cybersecurity. While each of these are significant in their own way, this research is going to tackle the challenges associated with integrating software factories and multiple pipelines in the upcoming year.

7 Acknowledgements

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM22-0184

8 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

CIO (DoD). (2019). *DoD Enterprise DevSecOps Reference Design*.

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

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

Defense", "Office of the Undersecretary of. (2020). *DoD Instruction 5000.02 Operation of the Adaptive Acquisition Framework*.

DoD, Office of the Secretary of Defense "Memorandum for Senior Pentagon Leadership: Continuous Authorization to Operate (cATO)", 2021, url=https://dodcio.defense.gov/Portals/0/Documents/Library/20220204-cATO-memo.PDF

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

DoD CIO. (2021). DoD Enterprise DevSecOps Strategy Guide. Department of Defense, Office of the CIO. Retrieved from https://software.af.mil/wp-content/uploads/2021/05/DoD-Enterprise-DevSecOps-2.0-Strategy-Guide.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.

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

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

9 Biographies

William Nichols is a senior researcher at SEI

Hasan Yassar is a director at SEI

Chris Miller is a senior engineer at SEI

Luiz Antunes is a senior engineer at SEI

Robert McCarthy is an engineer at SEI