Setting the Stage

Consider the following scenario:

You are a Special Operations Command (SOCOM) commander and are told that intelligence has discovered that an adversary has unexpected capabilities. As a result, you must reprioritize your capabilities. You inform the program manager (PM) for your advanced aircraft platform that a lower priority capability of the Whiz-Bang Software for Sensor Fusion, which was added to the roadmap 18 months ago, must now become the top priority and be delivered within the next 6 months. However, the two capabilities that were planned to be the next priority and that are due in 3 and 9 months are critical and must be completed as close to the original due dates as possible.

To address this scenario, you need to know the following:

1. What options are available for delivering the new, highest priority capability and the two next-priority capabilities without changing the staffing?
2. Will additional teams be needed to deliver the Whiz-Bang Software for Sensor Fusion capability within the next six months while staying on schedule for the other two capabilities? If so, how many? What is the cost of doing so?

Introduction

A PM is responsible for the overall cost, schedule, and performance of a program. They provide leadership, decision making, and oversight throughout the program’s and system’s lifecycle. They lead the program, understand requirements, balance constraints, manage contractors, build support, and use basic management skills.

The PM’s job becomes even more complex in large programs with multiple software development pipelines, where they must consider the following when making decisions:

- the cost, schedule, performance, and risk for every product in every pipeline
- the interdependencies among products developed in different pipelines
At the Software Engineering Institute (SEI), our work on Automated Cost Estimation in a Pipeline of Pipelines (ACE/PoPs) teaches PMs how to transform raw DevSecOps development data into actionable information that helps leadership make decisions during program execution. Using the actionable information from ACE/PoPs, the PM can keep their program on track by continuously monitoring and analyzing raw data from tools in multiple interacting DevSecOps pipelines of pipelines.

In this paper, we explore the decisions that PMs make and the information they need from DevSecOps pipelines to decide confidently. We will describe technical details of implementing ACE/PoPs in future papers.

Data that Program Managers Need

PMs make decisions almost constantly throughout a program’s execution, and there are many areas where the PM requires objective data to make the best decision possible at the time. This data can be divided into five main categories: cost, schedule, performance, quality, and risk. In this section, we discuss each of these categories. However, these categories—and many PM decisions—are also influenced by other factors. These factors include staffing, program stability, process effectiveness, the quality of program documentation, and the data that documentation provides. It is important to recognize how these five factors and other data are interrelated, as illustrated in Figure 1.

![Figure 1: Notional Program Performance Model](image)

All PMs track cost and schedule, but changes in staffing, program stability, and process effectiveness can affect both. If cost and schedule remain constant, these other changes are reflected in the performance or quality of the final product. Risks exist in every category, and managing risks requires collecting data to determine their probability of occurrence and impact if they manifest into issues.
Cost

Cost is typically one of the biggest drivers of a PM’s decisions. Cost overruns can be common in Acquisition Category I (ACAT I) programs and can trigger a Nunn-McCurdy breach.¹ The cost a contractor charges a program can have many different components, such as management, engineering, production, testing, and documentation. In this paper, we focus on the metrics for one aspect of cost: software development, where labor is typically the most significant contributor to costs. These costs cover all aspects of software development, including software architecture, modeling, design, development, security, integration, testing, documentation, and release.

Cost is, by far, the most understood metric. When comparing planned and actual software development costs, the units (e.g., dollars) are used consistently and require little to no interpretation or guidance. Although some accounting factors (e.g., time value of money, base year dollars) may need to be considered in practice. In this paper, we assume that a dollar is a dollar—planned, actual, or otherwise.

Schedule

Schedule is typically another major concern for PMs, who rely on accurate information to make decisions based on delivery timelines. Schedule changes can impact the delivery of capability to the warfighter. Schedule is also important when considering the availability of funding, the need for test assets, commitments to interfacing programs, and many other aspects of the program. In programs with multiple software pipelines, PMs must understand not only the technical dependencies but also the lead and lag times between inter-pipeline capabilities and rework. Schedule metrics available from a DevSecOps pipeline can help the PM make decisions based on how software development and testing activities are progressing across multiple pipelines.

Performance

Information about functional performance is critical when making decisions about the priority of capabilities and features in an Agile environment. When developing software, understanding its required level of performance contributes to informed decisions about which capabilities to develop further and which to reassess. The concept of “fail fast” cannot be successful without performance metrics that inform the PM (and the team) when an idea leads to a technical dead end.

¹ A Nunn-McCurdy breach “refers to Title 10, U.S.C. § 2433, Unit Cost Reports (UCRs). This amendment to Title 10 was introduced by Senator Sam Nunn and Congressman Dave McCurdy in the National Defense Authorization Act (NDAA) for Fiscal Year (FY) 1982. The amendment requires that ACAT I PMs maintain current estimates of Program Acquisition Unit Cost (PAUC) and Average Procurement Unit Cost (APUC). If the PAUC or APUC increases by 25 percent or more over the current Acquisition Program Baseline (APB) objective, or 50 percent or more over the original APB objective, the program must be terminated unless the Secretary of Defense (SECDEF) certifies to Congress that the program is essential to national security” [DAU 2023a].
Quality

Working software may not necessarily mean high-quality software. A PM must understand the overall quality of the software being developed. Software with a low level of quality can affect its performance and its long-term maintenance. In addition to functionality, there are many engineering performance factors (i.e., ‘ilities’) to consider based on the domain and software requirements. Additional performance factors become more prominent in a PoPs environment, including interoperability, agility, modularity, and compliance with interface specifications.

Risk

While the PM is making a decision, risk should always be a factor. Risks generally threaten cost, schedule, performance, and quality. The PM requires information to assess (1) the probability and impact of risks if they are not managed and (2) possible mitigations—including the cost and reduction of risk consequence for each mitigation. The risks affecting software development can result from factors such as an inadequate technical solution, supply chain issues, obsolescence, software vulnerabilities, issues with the DevSecOps environment, and overall staffing.

Other Considerations for Program Managers

Besides cost, schedule, performance, quality, and risk, the PM must also consider other contributing factors when making program decisions, especially when managing software development. In this section, we describe some common considerations: organization and staffing, processes, stability, and documentation. Each of these factors can affect cost, schedule, and performance.

Organization and Staffing

A PM must understand the structure and staffing of the program management office (PMO) team and the contractor team, including any subcontractors or government personnel on those teams. This understanding is especially important in an Agile/Lean development environment. To ensure that the development project\(^2\) meets the users’ needs and expectations, the PMO and users must provide subject matter experts (SMEs) to the developing organization. Users can include operators, maintainers, and trainers. The PMO must have appropriate staff with the specific skills needed to participate in Agile events and review the artifacts developed. The PM must also monitor the staffing of development and pipeline maintenance teams in programs being developed by government personnel.

In PMOs, an emerging trend is for all program partners (i.e., subcontractors, prime contractor, and government technical personnel) to identify staff members who, regardless of their tasking, have

---

\(^2\) Projects and programs differ in scope and focus. A project is typically a temporary endeavor, bounded in time and budget, that delivers a specific product or service. A Department of Defense (DoD) program provides a new, improved, or continuing materiel, weapon, or information system or service capability in response to an approved need. The program typically coordinates acquisition, research, development, logistics, and sustainment.
critical skills, capabilities, and experience. This information helps the PM manage staff members and conduct recruiting activities for the entire program.

**Processes**

Process issues can affect costs and schedules. For multi-pipeline programs, process inconsistencies (e.g., definition of done) and differences in software deliverables can cause massive integration issues. For discrete pipelines in an organization, process standardization (e.g., adopting common business rules for Jira) can improve overall process efficiency and productivity.

The PM must ensure that all PMO, contractor, and supplier processes are defined and executed consistently. All program partners in the same pipeline must understand the processes and practices used by all upstream and downstream DevSecOps activities. These processes and practices include understanding the project’s coding practices, coding standards, and pipeline tooling environments.

**Stability**

A PM must track metrics for factors such as staffing, cost, schedule, and quality; however, they must also know whether these areas are stable. Even if some metrics are positive (e.g., the program is below cost), trends or volatility can indicate future issues if there are wide swings in the data that are not explained by program circumstances. It can also be important to monitor stability in requirements and long-term feature prioritization. An Agile environment encourages changes in priorities; however, the PM must understand the costs and risks incurred. Furthermore, an Agile principle is to “fail fast,” which can accelerate learning about the software’s strengths and weaknesses. These are normal parts of Agile development, but the PM must understand the overall stability of the Agile process to be able to make informed decisions.

**Documentation**

One of the founding Agile principles encourages prioritizing code over documentation. However, it is important for a PM to receive all required documentation for the programs they manage. Further, the Department of Defense (DoD) requires documentation for all acquisition programs. The PM can find it challenging to strike a balance between the Agile philosophy of avoiding non-value-added documentation and the need to record essential design, architecture, coding, integration, and testing knowledge. This documentation must be accessible and understandable to be useful to the engineering team responsible for maintaining the software (while also meeting DoD documentation requirements). Automation can help balance these competing needs.

In a DevSecOps environment, documentation should be developed and reviewed incrementally. The PM must ensure that documentation that is part of a government baseline is completed as planned. This documentation can include models, software architecture, development environment information, Risk Management Framework (RMF) documentation, designs, and interface specifications.
Metrics that Your DevSecOps Pipeline Can Provide

Tools that automate the DevSecOps pipeline or manage the work surrounding the pipeline can provide useful metrics. DevSecOps pipeline tools that perform software development functions (e.g., build, test, deploy) provide direct access to data related to interim work products and overall code stability. Source control, static analysis, requirements management, issue tracking, and financial systems are examples of tools in the pipeline that can provide additional metrics to help PMs track and monitor growth and quality. Figure 2 depicts a simplified single pipeline. However, programs typically include multiple pipelines with integration points.

Figure 2: Simplified Software Factory Pipeline

Figure 3 illustrates how the main components of a pipeline-based measurement system could work. Failure data and other measurements (e.g., timestamps, duration, product size) are available at many points in the pipeline. Data is collected from several sources, including the work ticketing systems and the continuous integration/continuous deployment (CI/CD) pipeline.
The requirements, financial system, and roadmap (i.e., work breakdown structure [WBS]) provide the context for tracking the flow of software work items through the pipeline. This context enables work to be aggregated into meaningful functional capabilities and physical performance characteristics (i.e., performance requirements). With sufficient context, software apps can automatically collect and display much of the information needed to manage a program at many points in the DevSecOps pipeline. Some metrics are collected automatically, while others must be derived through analysis, which can also often be automated.

Agile development prioritizes speed of delivery and continuous adaptation; therefore, PMs must often interpret traditional measures of cost and schedule differently when assessing progress in Agile software projects. Programs require additional attention to ensure that data is properly aggregated from various projects and/or pipelines.

---

Figure 3 refers to DevSecOps (DSO), the industry group Practical Software Metrics (PSM), and software development lifecycle (SDLC).
**Capability Delivery**

In a DoD environment, the delivery of a capability should follow the schedule outlined in the project’s strategic roadmap. This roadmap should include a minimum viable product (MVP), a minimum viable capability release (MVCR), and subsequent releases. In other words, the capability is defined and the definition of *done* is established.

The incremental delivery of each work item should measurably advance the objective of fielding the capability. This information from the strategic roadmap leads to metrics for percent complete and completion dates, but only if the following are true:

1. The capability maps to the work items (e.g., stories, features, epics).
2. The work items are sequenced to support a delivery roadmap.
3. The work is staffed.
4. The overall capability and work items have reasonable estimates.

An additional challenge in the PoPs environment is aggregating deliveries from multiple pipelines, often with cross-pipeline dependencies. For example, Figure 4 depicts a fictitious program, Satellite Communication Device (SDC), with three teams: Main, Antenna Group, and Encryption HardWare (HW). Each node represents a task; nodes 5, 6, 7, and 8 represent integration points between Main and either Antenna or Encryption. The green lines represent the lead times \(t\) between two nodes. (To avoid clutter, we include only the necessary lead times in Figure 4.) The special case of \(t_{8-1}\) represents a lead time for rework discovered in integration that must reenter the development pipeline at node 1.

![Figure 4: Multiple Interacting Pipelines for a Fictitious Cyber Physical Device](image)

Nonetheless, the principles of tracking the percent complete and completion rates remain. The DevSecOps pipeline can provide data that PMs can use to track progress toward capability delivery. To deliver a capability, work items that implement the capability (i.e., meet product requirements) must be identified and organized, and they must be estimated and timestamped as they progress through the system. The most reliable way to measure completion is through delivery or deployment.
In the DoD, work items must be organized in a product-based WBS, which must be estimated and tracked as work items are completed and functionality is added to a delivery. A WBS is organized around features and capabilities that are often distributed to teams as stories (or epics that the teams will decompose into stories). A good indicator of progress in relation to the roadmap can be the completion of features and/or capabilities at the end of an increment.

High-level estimates must be provided to the PM in meaningful units (e.g., staff days, dollars). If development teams re-estimate the work, the new estimates must be converted into units that are meaningful to the PM. A percent complete can then be computed for the entire program or along any sub-branch of the WBS, provided that the completed work items are identified as product features linked to the WBS. Additional measures, such as build success rates and test success or failure rates, can be used to ensure that the product has been built well.

Cost

The DevSecOps pipeline provides data that can help PMs make decisions about costs. While the pipeline typically does not directly provide information about the dollars spent, it can feed earned value management (EVM) systems, and it can provide EVM-like data even absent of a requirement for the use of EVM. Cost information is most evident from the work applied to specific work items, which, in turn, requires information about staff and their activities.

For software developed using Agile processes in a DevSecOps environment, metrics available through the pipeline can provide data about team size, actual labor hours, and work completed versus planned. Tracking labor charges (i.e., hours worked) and full-time equivalents (FTEs), while clearly not the same as cost, can also provide some insight into cost performance. ACAT 1 program data shows that FTEs correlate strongly with cost at the pipeline or factory level. At the team level, the DevSecOps cadence of planning increments and sprints (i.e., the schedule) is a source of labor hour data, which scales linearly with cost. Some adjustments must be made to calibrate cost and hours for each pipeline since labor rates vary.

To identify potential cost overruns for a capability or feature, a PM can use metrics on the work completed versus planned. These metrics can also help a PM prioritize work and decide whether to continue working in specific areas or redirect funding to other capabilities. The work can be measured by comparing estimated to actual cost, and optionally comparing estimated to actual size. Predictability is measured by comparing the estimated cost of work planned to the actual cost of work delivered. The DevSecOps pipeline provides several direct measurements, including each work item that moves through development and production, the time each enters the DevSecOps pipeline, the time each is built, and the time each is deployed. This information must be joined with the work ticketing system to identify specific work elements and team-level cost estimates. To be most useful to the PM, the information must be traced to customer needs and/or requirements tracking and roadmap estimates.
Schedule

The DevSecOps pipeline can provide information about progress against the plan at several different levels. The most important level for the PM is the schedule related to delivering the capability to the intended users. The pipeline typically tracks stories and features; however, with links to a WBS, features can also be aggregated to show progress versus the plan for capability delivery. This traceability does not occur naturally, nor do the metrics if they are not adequately planned and instantiated.

The work must be prioritized, the effort estimated, and a nominal schedule derived from the available staff and teams. For programmatic purposes, the granularity of tracking should be detailed enough to detect schedule slips but broad enough to avoid excessive plan churn as work is reprioritized. Summary reports should provide assurance to the PM that the work has been planned and tracked at the expected level of detail.

Of course, there can be interactions between schedule and quality. Often, if there is an incentive to deliver quickly, quality is sacrificed. The PM must balance the need for faster delivery against the increased cost of fixing problems later in the program.

The schedule will be more accurate in the short term, and the plans must be updated whenever priorities change. In terms of schedule, predictability is one of the main metrics to look for in Agile development. Consider whether development is working to a repeatable cadence and delivering what was promised when expected. The PM requires credible ranges for program schedule, cost, and performance. Measures that inform predictability can be obtained from pipeline production rates, estimation bias, estimation variability, throughput, and lead times.

When assessing schedule and progress for large, multi-pipeline software development programs, it is important to remember that there is a clear distinction between indicators of progress (i.e., interim deliverables) and actual progress. The seventh principle of the Agile manifesto states, “Working software is the primary measure of progress.” With that in mind, a PM must differentiate between leading indicators of progress and actual measures of progress; story points are a leading indicator of progress. For example, a burn-up or burn-down chart showing completed story points is an indicator that work is being performed and software is being produced. However, work performed to complete individual stories (and/or sprints) is not guaranteed to generate working software. From the PM’s perspective, only completed software products that satisfy all conditions of done are true measures of progress (i.e., working software).

A problem in the multi-pipeline scenario, especially across organizational boundaries, is achieving co-
nor events (i.e., milestones). Because products usually require the integration of outputs from multiple pipelines, programs should monitor schedule performance reports from each pipeline to assure that the work is progressing toward key milestones. Issues that escape a pipeline are particularly problematic to isolate. After integration of products from multiple pipelines, it is often difficult to trace an issue to its origin because a fault propagated throughout the system before detection. The
potential schedule consequences of these delays include impact to the lead times within each activity in the pipeline (e.g., design, develop, test) that allows the following to occur:

1. Identify the source of the problem.
2. Fix the problem.
3. Propagate the fix throughout the system.

A practical approach to addressing different pipeline performance parameters is to report individual pipeline performance in units that are independent of the pipeline. Alleman and Henderson describe a process for converting velocity into EVM-type data [Alleman 2003]. The specific pipeline throughput and lead time distributions must be viewed within the context of that pipeline’s inventory of work and reported in units that are meaningful to PM oversight of a multi-team program. In other words, work items (e.g., stories) should be converted into cost. For each pipeline, the total body of work (e.g., in the WBS) must be costed and scheduled. Throughput and lead times use estimated costs and days (or similar schedule units). These rates are specific to the pipeline and inform its ability to meet integration commitments (e.g., milestones). The multi-pipeline problem then becomes recognizable as a network that can be analyzed using the Program Evaluation Review Technique (PERT).

Unless quality is extremely high, any rework discovered should be probabilistically mapped to the origin so that work can re-enter the system. Many of the measures can be accomplished with pipeline timestamps; however, the information must be joined to requirements and task management systems. These considerations imply measurement requirements that may not be explicitly addressed in the DevSecOps literature. Consider the merge points in the example of a multi-pipeline system shown in Figure 4.

For a typical DoD program, PMs can manually schedule estimates for multiple pipelines; for example, they can use an earned schedule [Lipke 2003]. As part of our work, we have interviewed PMs who successfully managed large programs using EVM data and pivot tables. However, manual calculations are time consuming and can use old data. While simple calculations can reveal whether a program has unrealistic goals, the risks posed by the variations in merge dependencies and rework are more difficult to assess. Fortunately, stochastic approaches (e.g., Monte Carlo Network [MCN]) are well established. What is needed are credible metrics to populate the simulations. Pipeline instrumentation can provide data for real-time analysis and projections.

When dealing with multiple pipelines, the PM must aggregate the data carefully. Since each team will have its own method of weighting story points, story point totals cannot simply be added up to track progress. Instead, the PM should assess the progress of capabilities and features that span one or more pipelines against the roadmap. The PM must also consider how to deal with blockers (i.e., tasks that must be completed before other tasks can start or be completed).

Results from integrated testing are another potential measure of progress. If the code from all pipelines is regularly integrated and tested, comparing the completed features that have passed integration testing to the roadmap may be another way to assess overall progress. This approach could enable the PM to make decisions that require an understanding of where software development progress stands in relation to the plan.
For most programs, there are many other tasks (e.g., certification, authority to operate, training) that can impact the schedule and, ultimately, the delivery of capability to users. While some of these tasks are associated with stories that will be tracked in the pipeline, many happen either entirely or partially outside of software CI/CD pipelines.

**Technical Performance of the Capability Delivery**

All work items required for a capability must move through the pipeline before that capability can be delivered. However, delivery is insufficient for a capability to be considered complete; a complete capability must also satisfy the specified requirements and the needs of the intended environment. To provide early indicators of the capability’s technical performance, the development pipeline can define *done* to include both the delivered capability and the schedule. In other words, the schedule performance cannot be determined until the requirements for the capability are satisfied. The customer normally validates technical performance, for example during end-of-sprint demonstrations. This feedback can provide valuable information about the system’s ability to meet users’ needs and expectations. Technical performance also includes indicators that can be measured using metrics available in the DevSecOps pipeline.

Test results collected using modeling and simulation runs or various levels of testing within the pipeline can be used to measure technical performance. If automated testing is implemented, tests can be run with every build. With multiple pipelines, these results can be aggregated to provide PMs with insight into test passage rates at various levels of testing. PMO staff must be assured that these tests actually measure the required technical performance.

Another way of measuring technical performance is to conduct specialized testing in the pipeline. Stress testing that, for example, includes requirements for key performance parameters (e.g., total number of users, response time with maximum users) can help determine how the system will perform when deployed. Reports to the PM can summarize the test success or failure against specific requirements.

**Risks**

Uncertainty creates risk, which includes potential threats to product capability, operational issues (e.g., cyber attack), the schedule, and costs. The PM must ensure that the program’s risks are identified, quantified, and, as appropriate, tracked until mitigated. Risk exposures and mitigations should be monetized (i.e., expressed in the form of currency) to meet the PM’s needs. Risk mitigations should be prioritized, added to the work items, and scheduled.

Because the effort to burndown risk is not available for development, risk burndown must be explicitly planned and tracked. The PM should monitor risk burndown and the cost ratios of risk to overall costs. Two separate burndowns that should be monitored are cost and value (exposure):

- Low variance from the expected risk cost burndown assures that risk mitigations have been adequately funded and executed.
- The value burndown indicates whether risk consequences have been successfully avoided.
Development teams may assign specific risks to capabilities or features. Risks affecting the development team are typically discussed during increment planning. Risk mitigation tasks should be identified, categorized, and estimated, and the totals should be included in reports to the PM.

Quality

The PM must be satisfied that the development team uses effective methods, identifies and remediates issues, and ensures that the delivered product is of sufficient quality to be used by the primary delivery pipeline and all upstream pipelines. Each individual story must pass through a DevSecOps toolchain, which includes automated activities, before it can be completed. The overall workflow includes additional tasks, designs, tests, and reviews that can be tracked and measured across the entire PoPs.

Categorizing work items as features, bugs, risk items, technical debt, or adaptations can provide a lagging measure of program health [Kersten 2018]. Each work item is assigned a work type category, an estimated cost, and, when complete, an actual cost. The portion of work in each category for all completed work items can be compared to plans and baselines. A deviation from the plan or unexpected drift in at least one measure can indicate a problem that must be investigated. For example, an increase in work categorized as “bug” suggests quality problems, whereas an increase in work categorized as “technical debt” suggests design or architectural deficiencies.

A DevSecOps environment typically includes one or more code analysis applications that automatically run every day or after each code commit. These analyzers produce lists of discovered weaknesses. Timestamps from analysis execution and code commits can be used to infer the time delay in addressing the issues. Issue density—measured by physical size, functional size, or production effort—can provide a first-level assessment of the code’s overall quality. Significant lead times for this stage indicate a high cost of quality. A static scanner can also identify issues with design changes in cyclomatic or interface complexity and might also predict technical debt. For PoPs, analyzing upstream and downstream results across pipelines can provide insight into the effectiveness of quality programs on the final product.

Automated builds provide another indicator of quality. Build issues typically involve inconsistent interfaces, obsolete libraries, or other global inconsistencies. The build lead time and the number of failed builds indicate quality failures and may predict future quality issues. By using a zero-defect build time as a baseline, the build lead time provides a way of measuring build rework. For PoPs, build time following the integration of upstream content directly measures how well the individual pipelines collaborated.

The test capabilities within the DevSecOps environment also provide insight into overall code quality. Defects found during testing, not after deployment, can help evaluate the overall quality of the code and the development and testing processes.

A common way to measure software quality is by tracking defects, which records the changes required for the code to function properly. Since tests, inspections, and scans are all imperfect, the number of defects found prior to release is a useful predictor of defects that escape into release. Tracking defect discovery and closure provides a PM with rich information about the overall quality
of the software and the production process. Software that meets only minimal requirements for release can contain latent quality issues that accumulate over time, creating a backlog of technical debt that will be costly to rework later. The PM can identify areas for improvement by (1) tracking defect insertion across pipelines and (2) analyzing defects to identify those commonly committed across independent pipelines.

Defect escapes include all post-delivery issues required to correct the product. Relevant measures include the number of issues, the estimated and actual fix costs, and fix lead-time distributions. The PM is most interested in the rate of escapes, the portion of rework, the defect burndown, and lead times. Although the average lead time to fix is informative for planning, risk assessment requires distributions. Expressing the burndown as a flow diagram allows the PM to monitor it to verify that the defect escapes are not exceeding the fixes. For a PoPs, if the PM identifies a pipeline with a high defect-insertion rate, they can focus their efforts on fixing or replacing that pipeline.

Individual fix time distributions include fix lead time in individual stages (e.g., issue recorded to planned, enter development to deploy, code complete to deploy). These values determine the lead time required for high-priority fixes. An alternative is to assign priority levels and divide lead times according to defect priority.

**Cybersecurity**

Another facet of quality involves cybersecurity. The PM must be satisfied that cybersecurity practices are being performed, issues are being found, and discovered issues are being addressed. Cyber issues include library changes, bugs associated with weaknesses, and cybersecurity-related work items. These issues should be counted, and effort and lead times should be measured.

The risk lead times (i.e., age), burndown, and planned and actual mitigation effort should be tracked and reported since escaped cybersecurity weaknesses represent risk. The DevSecOps pipeline can collect metrics related to cybersecurity, such as the following:

- effort applied to cybersecurity activities
- weaknesses found during static or dynamic analysis
- cybersecurity test failures
- cybersecurity-related issues
- cybersecurity weaknesses that escape a pipeline

This data can be used to inform cybersecurity risk, assess whether processes adequately support cybersecurity goals, identify gaps, and identify the effects of cybersecurity on schedule and cost goals. For PoPs, tracking common vulnerabilities across all the pipelines can help identify embedded or redundant weaknesses.

**Organization and Staffing**

The PM is concerned about the breadth of pipelines that comprise the overall pipeline for complex programs and the staffing levels for each organization involved. Often, staffing metrics from other organizations are not provided to the PM. If they are, the data is usually provided as an overall
headcount or an FTE number. Rarely does a PoPs’ PM have quantitative insight into the retention rates for critical skill positions.

Some metrics (e.g., team structure and hours worked) can be gleaned from pipeline data. A PM should also be able to derive metrics about team and general turnover rates. If a PM believes there are staffing issues, pipeline data can provide metrics to confirm it.

As mentioned earlier, in PMOs, an emerging trend is for all program partners (i.e., subcontractors, prime contractor, and government technical staff) to identify staff members who, regardless of their tasking, have critical skills, capabilities, and experience. This information helps the PM manage staff members and conduct recruiting activities for the entire program. Comparing the availability of all staff members and those with specialized skills enables the PM to determine the validity of assumptions that affect schedules and take appropriate action.

**Putting It All Together**

When a PM manages a project with multiple pipelines, the amount of available data can be overwhelming. However, there are tools that can aggregate that data and generate a dashboard of available metrics. Pipelines can generate several different dashboards that developers, testers, and PMs can use. The key to developing a useful dashboard is to select the metrics needed for decision making. Of course, specific metrics will vary by program and during the lifetime of a program.

For example, the PM will want to analyze the risk of staffing early in the program based on the plan so that decisions can be made based on the staff that is available. Later in the program, performance and quality may be higher priority risk items that will drive decisions. The dashboard should be updated to highlight metrics related to the changing priorities of the program’s needs.

It is never easy to determine which metrics are required to inform PM decisions. It takes time and effort to determine which risks drive decisions and which metrics can inform those decisions. With instrumented DevSecOps pipelines, those metrics are more readily available, and many can be provided in real time rather than waiting for future metrics reports. This approach can help the PM make decisions based on current data, especially in large, complex programs with multiple pipelines.
Glossary

**Capability**
“The ability to achieve a desired effect under specified standards and conditions through a combination of ways and means to perform a set of tasks” [DoD 2008]

**DevSecOps**
“DevOps is a modern software development approach that strives to bring development and operations teams together along with other stakeholders to improve efficiency and outcomes by focusing on shared business goals. DevOps follows and expands on key principles of the Agile software development and Lean engineering movements and represents a fundamental shift in how large, distributed enterprise organizations develop and deliver software” [SEI 2018].

**Features**
“A customer-understandable, customer-valued piece of functionality that serves as a building block for prioritizing, planning, estimation, and reporting” [Palmquist 2013]

**Increment**
“Agile software projects deliver the system in increments, which represent the value added to the system such as newly implemented features, removed defects, or an improved user experience” [Palmquist 2013].

**Pipeline**
“Each software factory executes multiple DevSecOps pipelines, where a pipeline is analogous to a manufacturing assembly line. Each pipeline is dedicated to a specific process uniquely tailored for the artifact being produced. There are no one-size-fits-all solutions for cybersecurity testing. Therefore, every DevSecOps pipeline is a collection of process workflows and scripts running on a set of DevSecOps tools operating in unison with their associated software factory” [DoD 2021].

**Program Manager**
“Designated individual with responsibility for and authority to accomplish program objectives for development, production, and sustainment to meet the user’s operational needs. The PM shall be accountable for credible cost, schedule, and performance reporting to the Milestone Decision Authority (MDA)” [DAU 2023b].

**Roadmap**
“The roadmap distills the vision into a high-level plan that outlines work spanning one or more releases; requirements are grouped into prioritized themes, each with an execution estimate” [Palmquist 2013].

**Story**
“A high-level requirement definition written in everyday or business language” [Palmquist 2013]
References

[Alleman 2003]

[DAU 2023a]

[DAU 2023b]
Defense Acquisition University. Program Manager. DAU Glossary. April 2023 [accessed]. https://www.dau.edu/glossary/Pages/Glossary.aspx!both|P|28271

[DoD 2008]

[DoD 2021]

[Kersten 2018]

[Lipke 2003]

[Palmquist 2013]

[SEI 2018]