Carnegie Mellon University
Software Engineering Institute

YOUR DEVSECOPS PIPELINES CAN PROVIDE PROGRAMS MANAGERS WITH ACTIONABLE DATA

Julie Cohen jcohen@sei.cmu.edu
William Nichols wrn@sei.cmu.edu

July 2022

Introduction

The Program Manager (PM) is accountable for the overall cost, schedule, and performance of a program. They exercise leadership, decision-making, and oversight throughout a program and a systems life cycle. They need to be the leader of the program, understand requirements, balance constraints, manage contractors, build support, and put to use the basic skills of management. The PM's job is even more complex in large programs with multiple software development pipelines where cost, schedule, performance, and risk for the products of each pipeline must be considered when making decisions, as well the inter-relationships between products developed on different pipelines.

The ACE/PoPs research work will advance the Program Management (PM state of the art for translating raw development DevSecOps data into actionable program management information to inform the decisions that need to be made in course of program execution. The capability to continuously monitor and analyze raw data from tools in multiple interacting DevSecOps pipelines-of-pipelines (PoPs) and provide actionable data to the PM to help keep the overall program on-track.

This white paper will explore the decisions that PMs have to make and information they need to confidently make those decisions that would be available from DevSecOps pipelines. Technical details of the implementation will be described in other papers.

What Data Do Program Managers Need?

PMs are required to make decisions almost continuously over the course of program execution. There are many different areas where the PM needs objective data in order to make the best decision possible at the time. This data falls into the main categories of cost, schedule, performance, and risk. These will each be discussed below. But these areas, and many PM decisions, are also impacted by other areas of concern to include staffing, processes effectiveness, program stability, and the quality and data being provided by program documentation. Even though we have used these categories, it is important to recognize how these data are related to each other as illustrated in Figure 1:Notional Program Per-

formance Model. All PMs track cost and schedule. But changes in staffing, program stability and process effectives can drive changes to both cost and schedule. And if cost and schedule are held constant, then these changes will manifest as changes in the end product's performance and/or quality.

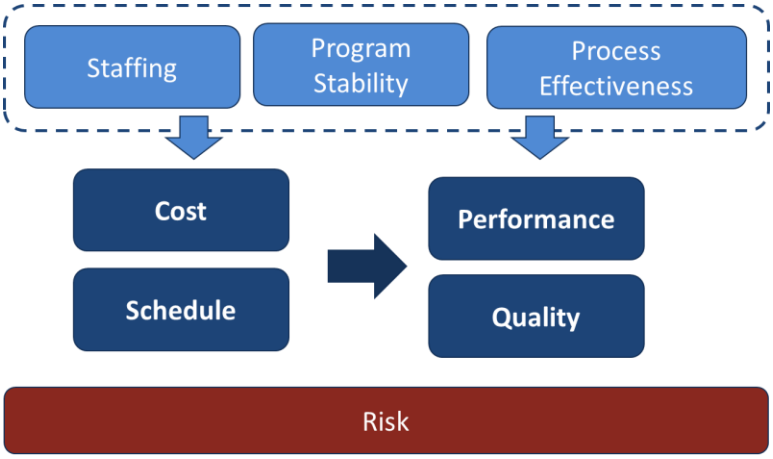

Figure 1:Notional Program Performance Model

Risk is a unique measure that can apply to any aspect of the program that the PM has identified as having a probability of affecting the program outcomes. These will be discussed in the following section.

Cost

Cost is typically one of the largest drivers of decisions for a PM. Cost overruns are often common and on ACAT I programs and can trigger a Nunn-McCurdy[1] breach. Cost charged by the contractor(s) on a program has many facets to include management costs, engineering costs, production costs, testing costs, documentation costs, etc. For this paper we are going to focus on providing metrics for one aspect of costs, software development. For software development projects, labor is usually the single most significant contributor to cost. This includes costs for all software development activities to include software architecture, modeling, design, development, security, integration, testing, documentation, and release.

[1] 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. Requires that Acquisition Category I (ACAT I) program managers (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. (https://www.dau.edu/glossary/Pages/GlossaryContent.aspx?itemid=28040)

Cost is by far the most commonly understood metrics. The units, dollars, are used everyone every day and require little to no interpretation guidance in terms of tracking planned versus actual software development costs. Although there are some accounting factors that might need to be considered (e.g., time value of money, base year dollars, etc.) in practice, this paper will assume a dollar is a dollar (planned, actual, or otherwise).

Schedule

Schedule is typically another major concern for the PM. There are often many dependencies related to the delivery timelines and many decisions need accurate schedule information. Schedule changes can impact the delivery of capability to the warfighter along with the availability of funding, test assets, interfacing programs, and many other aspects of the program. On programs with multiple software pipelines, it is important to understand the dependencies of the work on various pipelines. Schedule metrics available from the DevSecOps pipeline can help make decisions based on the how the software development and testing activities on multiple pipelines are progressing.

Product Performance

Performance is critical in making decision regarding the priority of capabilities and features in an agile environment. Understanding the required level of performance of the software being developed can allow informed decisions on what capabilities to continue developing and which to re-assess. The concept of "fail fast" can't be successful unless you have metrics to quickly inform the PM (and the team) when an idea leads to a technical dead end.

Risk

Risk should always by a factor when the PM is making a decision. Risks generally threaten cost, schedule, or performance. The PM needs information on the risks involved and possible mitigations (to include the cost of the mitigations) for each possible course of action. The risks involved in software development can result from adequacy of the technical solution, supply chain issues, obsolescence, software vulnerabilities, as well as issues with the DevSecOps environment and with overall staffing.

Other Program Manager Concerns

In addition to the traditional PM responsibilities to make decisions related to cost, schedule, performance and risk, there are contributing factors that need to be considered by a PM when making program decisions, especially with respect to software development. This section discusses some of the most common ones. These include software quality, organization and staffing, process effectiveness and stability, and if all the documentation needed is being produced on schedule. Each of these factors can affect cost, schedule, performance. These will be discussed in the following section.

Quality

Working software may not mean high quality software. A PM needs insight as to the overall quality of the software. In addition to functionality, there are many engineering performances factors (i.e., 'ilities') to consider based on the domain and requirements of the software.

Organization / Staffing

A PM needs to understand the organization/staffing for both their own Program Management Office (PMO) team and the contractor's team (to include any subcontractors or government personnel on those teams). This is especially important in an agile/lean development. The PMO and users need to provide Subject Matter Experts (SMEs) to the developing organization to ensure the development is meeting the users' needs and expectations. Users can include operators, maintainers, trainers, and others. The PMO also needs appropriate staff with specific skills to be involved in the agile events and to review the artifacts developed. For programs where the development is being done by government staff, then staffing for the development teams and for pipeline maintenance are also important to track.

Processes

It can also be important for a PM to understand if both PMO and contractor processes are efficient and are being executed as planned. Process issues can have both cost and schedule impacts.

For multi-pipeline programs, process inconsistencies (e.g., definition of 'done') and differences in the contents of software deliverables can create massive integration issues.

Stability

In addition to tracking metrics for items like staffing, cost, schedule and quality, a PM also needs to track if these areas are stable. Even if some metrics are positive (for example the program is below cost), if there are wide swings in the data that are not explained by program circumstances these can point to possible issues in the future. In addition, stability in requirements and long-term feature prioritization could also be important to track. While agile allows changes in priorities, if the priorities are constantly changing then the PM would need to understand the risks involved. Also, one of the agile principles is to "learn fast" which can result in some "failures." These are a normal part of agile development, but the overall stability of the agile process needs to be understood by the PM.

Documentation

One other area that is important to the PM is receiving the required documentation. In a DevSecOps environment that documentation should be developed and reviewed incrementally. For any documen-

tation that will be a part of the government baseline, the PM needs to ensure that documentation is being completed as planned. This could include models, software architecture, Risk Management Framework (RMF) documentation, design documentation, interface specifications, etc.

What metrics can be provided through your DevSecOps pipeline?

Information can be obtained from tools that automate DevSecOps Pipeline (such as build, test, and deployment), tools that the DevSecOps Pipeline uses (e.g. source control, static analysis) and tools surrounding the pipeline that manage the work including requirements management, issue tracking, and financial systems. A simplified single pipeline is shown in Figure 2: Simplified Software Factory Pipeline. . A program typically includes multiple pipelines with integration points. Many measures are available from instrumentation of environment, for example financial, requirements, and ticketing systems.

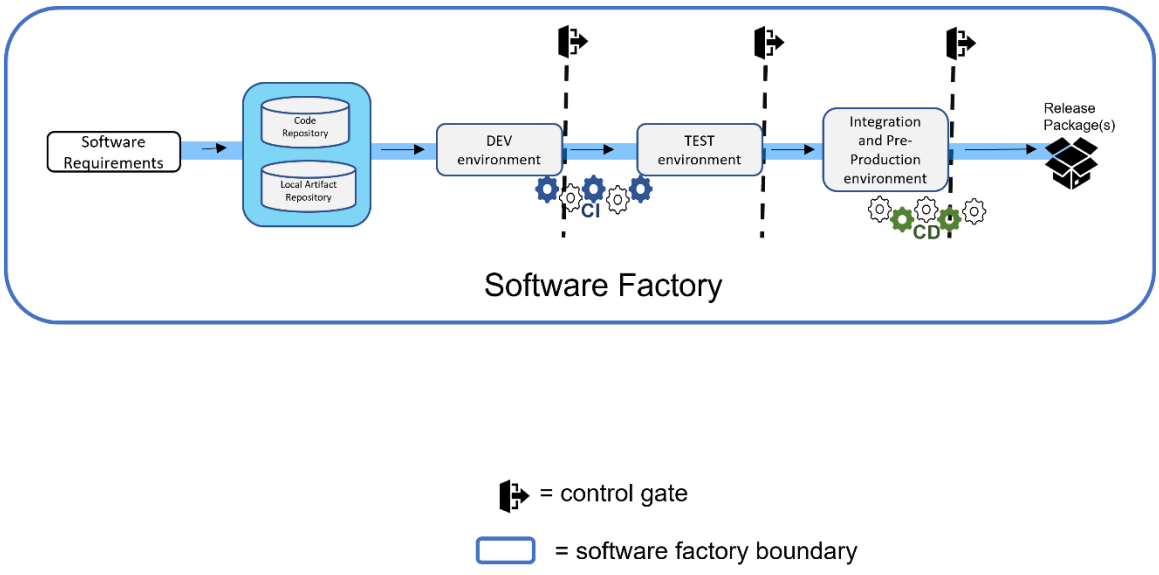

Figure 2: Simplified Software Factory Pipeline

An overview used for a concept of operations in provided in Figure 3: DevSecOps Data Collection Overview. Measurements including timestamps, duration, product size, and failures are available at many points in the pipelines. Data is collected from a number of sources, including the work ticketing systems and the CI/CD pipeline. Contextual information from the requirements, financial system, and roadmap (or work breakdown) provide information for tracking the flow of software work items through the pipeline. Context allows the work to be aggregated into meaningful capabilities or non-

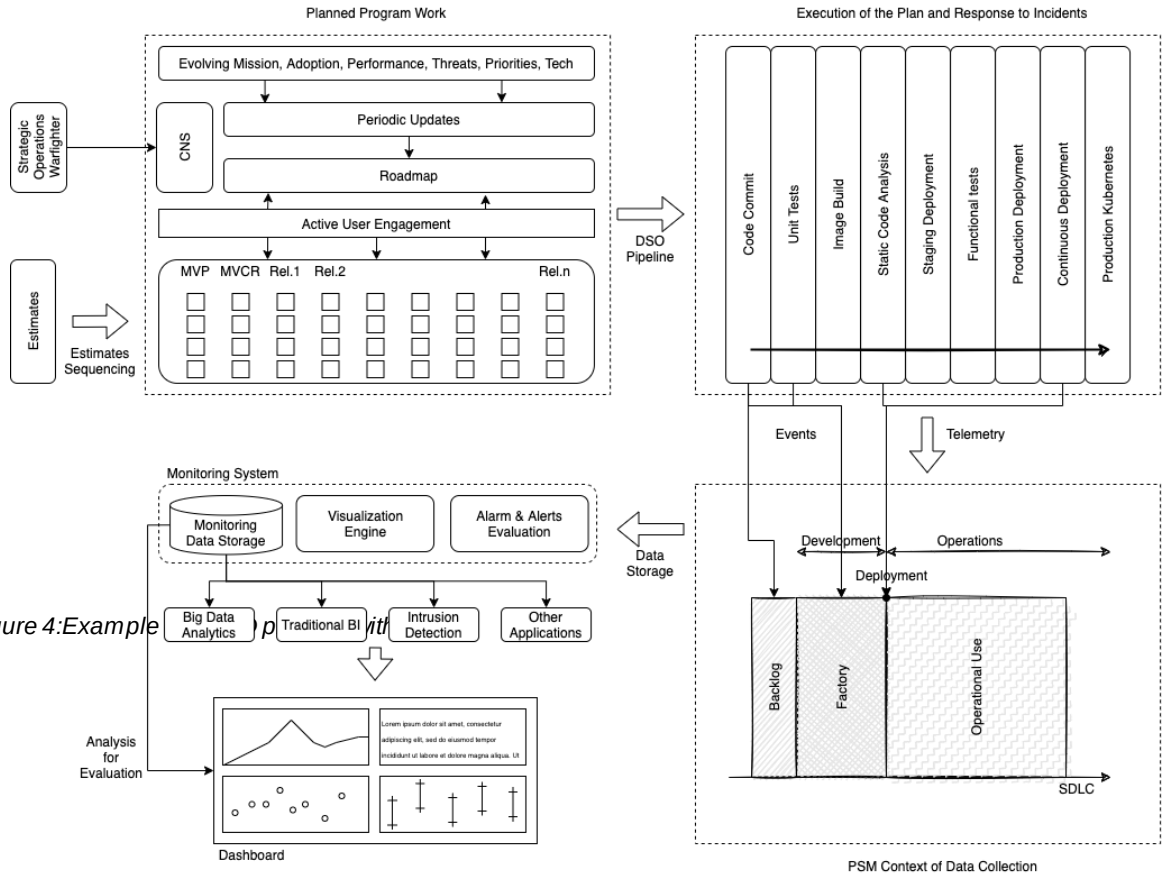

Figure 4:Example p */API*

Figure 3: DevSecOps Data Collection Overview

capability categories. With sufficient context, much of the information needed to manage a program can collected and displayed automatically.

Introduction:

Metrics can be collected at many points in the DevSecOps pipeline. Some of these are collected automatically, others must be derived through analysis, which can often be automated.

One point to note is that agile development prioritizes speed of delivery and continuous adaptation, so traditional measures of cost and schedule often need to be interpreted in different ways when assessing progress in agile software projects. Programs require additional care to assure that data from differing projects or pipelines is properly aggregated.

Product Completion

The DevSecOps pipeline can provide data that can help PMs measure the progress toward product completion. Product completion requires that work items that implement product requirements have been identified, organized in a product break structure (PBS), estimated, and tracked as they are added

to a delivery. Typically, a PBS is organized around features and capabilities that are often distributed to teams as epics or stories. High level estimates must be in units meaningful to the PM, staff days or dollars. If the work is re-estimated by development teams, the new estimates must be converted to the PM units. A percent complete can then be computed for the entire project or along any sub-branch of the PBS provided that the completed work items are identified as product features linked to the PBS. The most reliable way to measure completion is through delivery or deployment. Additional measures can be used to assure that the product has been built well.

Cost:

The DevSecOps pipeline can provide data that can help PMs make decisions regarding cost. While the pipeline typically doesn't directly provide information on dollars spent, it can feed typical Earned Value Management (EVM) systems and can also provide EVM-like data absent of a requirement for EVM. For software developed using agile processes in a DevSecOps environment, there are metrics available through the pipeline that can provide data on team size and, actual labor hours and worked completed vs. planned. Although, clearly not the same as cost, tracking labor charges (hours worked) and Full Time Equivalents (FTEs) can provide some indication of cost performance. Data from ACAT 1 programs shows that Full Time Equivalents (FTEs) correlate very strongly with cost at the pipeline or factory level. Thus, at the team level, schedule provides labor hours and labor hours scale linearly with cost. Some care or adjustment must be made to calibrate cost and hours for each pipeline because cost rates vary.

Metrics on work completed versus planned can provided a PM the data need to make decisions regarding potential cost overruns for a capability or feature and can be used to inform decisions regarding prioritization and whether or not to continue work in specific areas or to move funding to other capabilities. The work can be measured in estimated/actual cost, and optionally an estimated/actual size may be measured. The predicted cost of work planned vs actual cost of work delivered measures predictability. The DevSecOps Pipeline provides several direct measurements, including the actual work items taken through development and production, the time they enter the DevSecOps Pipeline, when they are built, and when they deploy. This information must be joined with the issue tracking system to identify specific work elements and team level cost estimates. To be most useful to the PM, the information must be traced to requirements tracking and the roadmap estimates.

Schedule:

The DevScOps pipeline can provide progress against plan at several different levels. The most important level for the PM is the schedule related to delivering capability to the users. The pipeline typically tracks stories and features, but with links to a PBS, features can be aggregated to show progress versus the plan for capability delivery as well. 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 small enough to detect schedule slips, but large enough to avoid excessive

plan churn as work is reprioritized. One caution, there can be interactions between schedule and quality. Often, if there is an incentive to deliver quickly, quality is often sacrificed. The PM must balance the need for faster delivery against the need to fix problems later in the program, which typically increases cost.

This will be more accurate on a short-term scale and the plans would need to be updated whenever there is a change in priorities. Note that in agile development, one of the main metrics to look for with respect to schedule is predictability. Is the developer working to a repeatable cadence and delivering what was promised when expected? The program manager needs credible ranges for program schedule, cost, and performance. Measures that inform predictability that can be obtained from the pipeline bias and variability of estimates, throughput, and lead times.

It is important to remember that when assessing schedule and progress for large multi-pipeline software development programs, there is a clear distinction between 'indicators' of progress (i.e., interim deliverables) and actual progress. The 8th tenant of the Agile manifesto states "working software" is the primary measure for tracking 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. As a program populates a burn-up or burn-down chart showing completed story points, this indicates that work is being performed. It provides a leading indication of software being produced. It is not a guarantee that the work performed to complete individual stories (and/or sprints) will result in working software. From the PM perspective, focus on completed software products that are true measures of progress (i.e., working software).

A problem in the multi-pipeline scenario, especially across organizational boundaries is the achievement of coordination events (milestones). Programs should independently monitor schedule performance of each pipeline to determine that progress toward key milestones (usually requiring integration of outputs from multiple pipelines) Issues that escape a pipeline are particularly problematic because of delays in identifying the source and propagating changes through the system once they are integrated with the products from other pipelines. The potential schedule consequences include the lead times within each activity in the pipeline (design, develop, test, etc.) to

1) identify the problem source
2) lead time to fix
3) lead time to propagate through the system

When dealing with multiple pipelines, the PM must be careful in how the data is aggregated. Each team will have their own method of weighting story points, so the story point totals can't just be added up to track progress. Instead, the PM should judge progress of capabilities and features which may span one or more pipelines against the roadmap. Another important consideration for the PM is dealing with any blockers. These are tasks that need to be completed before other tasks can start (or be completed).

Another potential progress measure could be results from integrated testing. If the code from all the pipelines is regularly integrated together and tested, then looking at the completed features that have passed integration testing versus the roadmap may be another way to assess overall progress and allow the PM to make decisions that require an understand of where software development progress is against the plan.

In addition, for most programs there are many other facets that can impact schedule progress and ultimately delivery to users. Tasks such certification, Authority to Operate, training, and others can all impact the schedule and while some of these have stories that related to them that will be tracked in the pipeline, many happen either fully or partially outside the software CI/CD pipelines.

Product Performance:

Product performance has several facets that can be measured through metrics available in the DevSecOps pipeline. One measure of product performance can be measured using test results. These results can be collected using modeling and simulation runs or through various levels of testing within the pipeline. If automated testing has been implemented, then tests can be run with every build. With multiple pipelines, these results can be aggregated to give decision makers insight into test passage rates at different levels of testing. One caveat is that the PMO staff must make sure that tests being run are in fact testing for the needed performance.

A second way to measure performance is by asking users for feedback after sprint demos and end of increment demos. Feedback from these demos can provide valuable information on if the system performance is meeting use needs/expectation.

A third way to measure product performance is through specialized testing that can take place in the pipeline. Stress testing that can test requirements such as total number of users, response time with maximum users, etc. can help determine how the system will perform when deployed.

Risk:

Risk includes potential threats to the product capability delivery schedule and operational issues such as cyber-attack. The program must assure that risks have been identified and if necessary addressed. For the purposes of the PM, risk exposures and mitigations should be monetized. The risk mitigations should be prioritized and included among the work items and scheduled. Because effort applied to burning down risk is not available for development, risk burndown must be explicitly planned and tracked. The PM should monitor the risk burndown and cost ratios of risk to the overall period costs. Two separate burndowns should be monitored, the cost, and the value (exposure). The cost assures that risk mitigations have been adequately funded and executed. The value burndown indicates actual reduction in risk level.

Development teams may assign specific risks to capabilities or features. Development team risks are usually discussed during increment planning. Risk mitigations added to the work items should be identified as risk and the totals included in reports to the PM.

Quality:

The program must be satisfied that the development uses effective methods, that issues are identified and remediated, and that the delivered product has sufficient quality. A DevSecOps toolchain includes a number of automated activities through which the individual stories must pass prior to completion. In addition, the overall workflow includes additional tasks, design, and reviews that can be tracked and measured.

Categorizing work items, e.g. as "feature", "bug", "risk item", "technical debt", "adaptation" can provide a lagging measure of program health[2]. Each work item is given a work type category, an estimated cost, and an actual cost. For the completed work items, the portion of work in each category can be compared to plans and baselines. Variance from plan or unexpected drift in one of the measures can indicate a problem that should be investigated. For example, an increase in "bug" work suggests quality problems while an increase in "technical debt" issues can signal design or architectural deficiencies.

Typically, a DevSecOps environment includes one or more code analysis applications that are automatically run daily or with every code commit. The outputs of these analyzers include weaknesses discovered and the time delay introduced to address issues. Issue density can provide a first-level assessment of the overall code quality. Large lead times for this stage indicates a high cost of quality. . Static scanner can also identify issues with design changes in to cyclomatic or interface complexity and may predict future "technical debt.

Automated build provides another indicator of quality. Build issues usually involve inconsistent interfaces, obsolete libraries, or other global inconsistencies. Lead time for build and number of failed builds indicate quality failures and may predict future quality issues. By using the duration of a zero defect build time as a baseline, the build lead time provides a build rework measure.

Test capabilities within the DevSecOps environment also provide insight into overall code quality. Defect found during testing versus after deployment can be used to help evaluate the over quality of the code and the development and testing processes.

The most basic and universal way to measure software quality is by tracking defects. Defects provide a record of failed tests (or informal inspections) which indicate the software doesn't perform in a manner it needs to perform. Tracking defect discovery and closure can provide a PM with a wealth of knowledge as to the overall quality of the software. Software that meets minimum requirements for

[2] Project to Product: How to Survive and Thrive in the Age of Digital Disruption with the Flow Framework Kindle Edition, Mik Kersten, 2018

release may contain lingering quality concerns that can build up over time and create a backlog of technical debt which will be costly to correct later.

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 the 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. The burndown as a flow diagram can be monitored to verify that the defect escapes are not exceeding the fixes.

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

Cybersecurity

Another facet of quality involves cybersecurity. The program manager must be satisfied that cyber security practices are being performed, that issues are found, and that the discovered issues are addressed. The DevSecOps pipeline can collect metrics related to cybersecurity such as effort applied to cyber security activities, weaknesses found during static or dynamic analysis, cybersecurity test failures, and information on cybersecurity related issues. Cyber issues include library changes, bugs associated with weaknesses, and cybersecurity related work items. These should be counted, with effort and lead times measured. Because escaped cybersecurity weaknesses represent a risk, the risk lead times (age), planned and actual mitigation effort, and burn down should be tracked and reported. ….

Organization/Staffing

For complex programs the breadth of pipelines that make-up the overall pipeline and their respective organizations' staffing are a concern for the PM. Often staffing metrics of other organizations are not provided to the PM. And if they are, many times the data is provided as an overall headcount or FTE number. Rarely does the PM of a PoP, have quantitative insight as to critical skills positions and their retention rates.

But some metrics can be gleaned from pipeline data. Team structure and hours worked should be available. A PM should also be able to get metrics on team turnover and overall turnover rates from pipeline data. F a PM believes there are staffing issues, pipeline data could provide metrics to show whether this might be an issue.

Putting it all together

There are many different metrics available from a DevSecOps pipeline and environment. Sources of data inside the pipeline include, but are not limited to, the revision control system, automated test case

management, build and deployment tools, and static analysis. Environment tools include work management or issue ticketing, requirements management, and financial systems. Preparing and joining the data to convert data into information and metrics is beyond the scope of this note but is the subject of related work. We advocate, in this note, for the use of this data for program management decisions. To that end, we will conclude with some additional description of how this can provide usable information, especially in increasingly complex multi-pipeline environments.

When a PM manages a project with multiple pipelines, the amount of data available can get overwhelming. But there are tools available for use within pipelines that will aggregate data and create a dashboard of the available metrics. Pipelines could generate several different dashboards for use by developers, testers and PMs. The key to developing a useful dashboard is to select appropriate metrics to make decisions. Of course specific metrics will vary by not only by program, but also during the lifetime of a program.

For example, early in a program, staffing may be a risk and something the PM wants to make sure is building to the plan so that decisions can be made based on staff available. But later in the program, performance and quality may be higher risk items that will drive decisions. The dashboard should change to highlight metrics related to those changing facets of program needs.

Determining what metrics are needed to inform PM decisions is never an easy task. It takes time and effort to determine what risks will drive decisions and what metrics could inform those decisions. But with instrumented DevSecOps pipelines, those metrics are more readily available, and many can be provide in real time, without the need to wait for a monthly metrics report. Especially in large complex programs with multiple pipelines, this can help the PM to make decisions based on timely data.

Some key points to note arise in a system with more than one pipeline.

- each pipeline will have its own personality, throughput and lead time distributions from one pipeline differ significantly among pipelines.
- late discovered rework, for example discovered in integrations may require sending work back to different pipelines.
- because traditional flow measures violate assumptions of Little's Law, stochastic modeling, as with PERT is needed.

These considerations imply measurement needs that have not been explicitly considered in the DevSecOps literature. For illustration, consider the example of a multipipeline system shown in Figure 5: Multiple interacting pipelines for a fictitious cyberphysical device. The diagram includes three teams, Main, Antenna Group, and Encryption. 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 lead times (t) between two nodes. To avoid clutter, only enough lead times to illustrate have been included in the diagram. The special case of $t_{8\text{-}1}$ represents a lead time for rework discovered in integration to re-enter the development pipeline at node 1.

A practical approach to address different pipeline performance parameters is report individual pipeline performance in units that are independent of the pipeline. Converting a velocity into EVM type data has been described by Alleman and Henderson (Alleman 2003) Thespecific pipeline throughput and lead time distributions must, therefore, be placed into the context of the that pipeline's inventory of work and reported in units meaningful to the program manager overseeing a multi-team program. That is, work items (stories) are converted into cost. The total body of work, for example in the work breakdown structure (WBS) must be costed and scheduled for each pipeline. Throughput and lead times use estimated cost and days (or similar schedule units). These rates apply only to the specific pipeline and inform the ability to meet integration commitments (Milestones), The multi-pipeline problem then becomes recognizable as a network that can be analyzed using PERT techniques.

Unless quality is extremely high, rework discovered at any point should be probabilistically mapped to the origin so that work can re-enter the system. Much of the measures can be accomplished with pipeline time stamps, however, the information must be joined to requirements and task management systems.

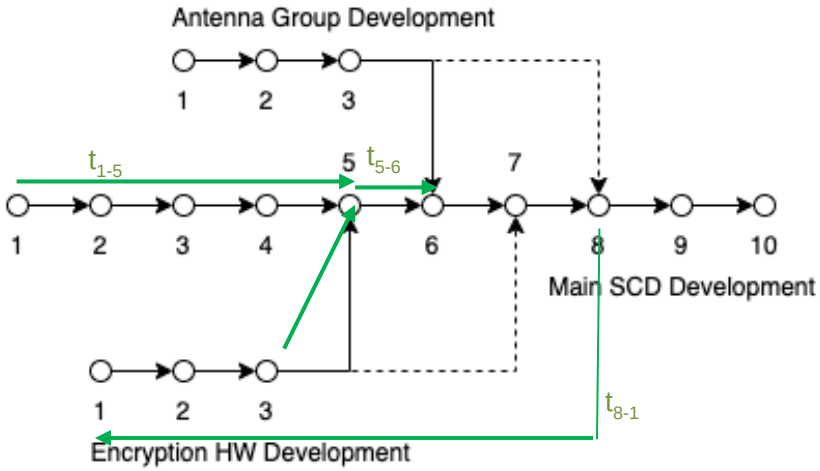

Figure 5: Multiple interacting pipelines for a fictitious cyberphysical device

If you are a PM on typical on a typical DoD program, schedule estimates can be performed manually, for example using Earned Schedule (Lipke 2003). The authors of this paper have spoken to successful program managers who have successfully managed large programs using EVM data and pivot tables. Nonetheless, manual calculations will be time consuming and likely use old data. Moreover, while simple calculations can show a program has unrealistic goals, the risks from variation at merge dependencies and rework cannot assessed so easily. Fortunately, stochastics approaches such as PERT are well established, what is needed are credible metrics to populate the simulations. Pipeline instrumentation offers opportunities not only for real time collection of the relevant metrics, but also for real time analysis and projections.

Definitions

Capability: The ability to achieve a desired effect under specified standards and conditions through combinations of ways and means to perform a set of tasks (U.S. DoD *Systems Engineering Guide for Systems of Systems* [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 (https://resources.sei.cmu.edu/asset_files/Brochure/2018_015_001_521283.pdf)

Features: A customer-understandable, customer valued piece of functionality that serves as a building block for prioritization, planning, estimation, and reporting. (https://resources.sei.cmu.edu/asset_files/TechnicalNote/2013_004_001_62918.pdf)

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. (https://dodcio.defense.gov/Portals/0/Documents/Library/DevSecOps%20Playbook_DoD-CIO_20211019.pdf)

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). (https://www.dau.edu/glossary/Pages/Glossary.aspx#!both|P|28271)

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 (https://resources.sei.cmu.edu/asset_files/TechnicalNote/2013_004_001_62918.pdf)

Story: A high-level requirement definition written in everyday or business language (https://resources.sei.cmu.edu/asset_files/TechnicalNote/2013_004_001_62918.pdf)

References

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

Kersten, Mic, **Project to Product: How to Survive and Thrive in the Age of Digital Disruption with the Flo**w, IT Revolution Press, 2018.

Alleman, G. B., Henderson, M., Hill, C. H. M., & Seggelke, R. **Making Agile Development Work in a Government Contracting Environment**. (2003). *Agile Developmen*t

Contact Us

Software Engineering Institute
4500 Fifth Avenue, Pittsburgh, PA 15213-2612

Phone: 412/268.5800 | 888.201.4479
Web: www.sei.cmu.edu
Email: info@sei.cmu.edu