Carnegie Mellon University
Software Engineering Institute

PROGRAM MANAGERS—THE DEVSECOPS PIPELINE CAN PROVIDE ACTIONABLE DATA

Julie Cohen

William Nichols

January 2023

Setting the Stage

As the SOCOM Commander, you are told that intelligence has discovered that an adversary has unexpected capabilities. Therefore, you need to reprioritize capabilities. You inform the program manager (PM) for your advanced aircraft platform that the lower priority capability for the whiz-bang software for sensor fusion that was on the roadmap 18 months from now will need to become the top priority and be delivered in the next 6 months. But the next two priorities are still very important and are needed as close to the original dates (3 months and 9 months out) as possible.

You need to know:

1. What options to provide the new capability and the next two priority capabilities with reduced capability can be provided without a change in staffing?
2. How many more teams we would need to add to get the whiz-bang software in the next six months and to stay on schedule for the other two capabilities? And what is the cost?

Introduction

The 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 system's life cycle. They need to be the leader of the program, understand requirements, balance constraints, manage contractors, build support, and use basic management skills. 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 Automated Cost Estimation in a Pipeline of Pipelines (ACE/PoPs) research work will show PMs how to translate raw development DevSecOps data into actionable program management information to inform the leadership decisions that need to be made in the 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 can help keep the overall program on track.

This white paper will explore the decisions that PMs 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. These data fall into the main categories of cost, schedule, performance, and risk. Each category will be discussed below. But these categories, and many PM decisions, are also impacted by other areas of concern, including staffing, process effectiveness, program stability, and the quality and data 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. All PMs track cost and schedule, but changes in staffing, program stability, and process effectiveness can drive changes to both cost and schedule. If cost and schedule are held constant, then these changes will manifest in the end product's performance or quality. Risks can be found in every category. Managing risks requires collecting data to quantify both the probability of occurrence and impact of each risk if it manifests into an issue.

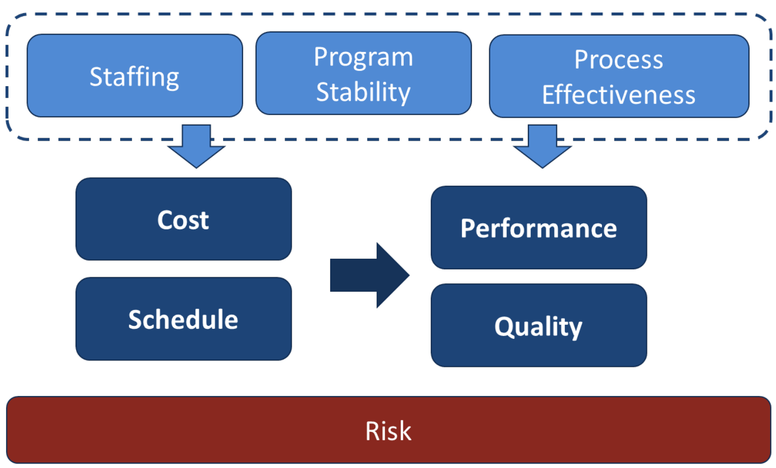

Figure 1: Notional Program Performance Model

Cost

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

Cost is by far the most understood metric. The units (dollars) are used 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, who needs accurate information to make decisions that depend on delivery timelines. Schedule changes can impact the delivery of capability to the warfighter. Schedule is also important when considering funding availability, need for test assets, commitments to interfacing programs, and many other aspects of the program. On programs with multiple software pipelines, it is important to understand not only the technical dependencies, ut also the lead and lag times between inter-pipeline capabilities and rework. Schedule metrics available from the DevSecOps pipeline can help the PM make decisions based on how software development and testing activities on multiple pipelines are progressing.

Performance

Functional performance is critical in making decisions regarding the priority of capabilities and features in an agile a 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 reassess. 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.

---

[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. 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. (https://www.dau.edu/glossary/Pages/GlossaryContent.aspx?itemid=28040)

Quality

Working software may not mean high-quality software. A PM needs insight as to the overall quality of the software. Poor quality software can impact both performance and long-term maintenance 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. Additional performance factors become more prominent in a pipeline-of-pipelines environment. Interoperability, agility, modularity, and compliance with interface specifications are a few of the most obvious ones.

Risk

Risk should always be a factor when the PM is making a decision. Risks generally threaten cost, schedule, performance or quality. The PM needs information to assess the probability and impact of the risks if not managed and possible mitigations (including the cost of the mitigations and reduction in risk consequence) 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, and issues with the DevSecOps environment and overall staffing.

What Else Should Program Managers Consider?

In addition to the traditional PM responsibilities of making decisions related to cost, schedule, performance, and risk, the PM must also consider additional contributing factors when making program decisions, especially with respect to software development. This section discusses some of the most common ones, including organization and staffing, process effectiveness and stability, and the required documentation is being produced on an acceptable schedule. Each of these factors can affect cost, schedule, and performance. These will be discussed in the following section.

Organization/Staffing

PMs need to understand the organization/staffing for both their own Program Management Office (PMO) team and the contractor's team (including 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.

An emerging trend within PMOs, is to have all Program partners (i.e., subcontractors, prime contractor, and government technical staff) identify members of their staff that have critical skills, capabilities, and experience regardless of their tasking. This provides the PM the ability to manage personnel and recruit across the Program at large.

Processes

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

It is important for a PM to ensure both PMO, contractor, and supplier processes are defined and repeatably executed. For a single pipeline, it is critical for all Program partners to understand the processes and practices of the upstream and downstream DevSecOps activities. This includes coding practices and standards as well as understanding the pipeline tooling environments.

For multi-pipeline programs, process inconsistencies (e.g., definition of 'done') and differences in the contents of software deliverables can create massive integration issues. For discrete pipelines within an organization, process standardization (e.g., adopting common Business Rules , e.g. for Jira) can improve overall process efficiency and productivity. Process issues can have both cost and schedule impacts.

Stability

In addition to tracking metrics for items like staffing, cost, schedule, and quality, a PM also needs to know if these areas are stable. Even if some metrics are positive (for example, the program is below cost), trends or volatility can point to possible issues in the future if there are wide swings in the data that are not explained by program circumstances. In addition, stability in requirements and long-term feature prioritization could also be important to track. While agility encourages changes in priorities, , then the PM needs to understand the costs and risks incurred. Furthermore, an agile principle is to "fail fast" which can increase the rate of learning the software's strengths and weaknesses. 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. One of the founding agile principles encourages the prioritizing code over documentation. At the same time DoD requires documentation for Acquisition programs. This creates a PM challenge to balance the spirit of agile by avoiding non-value-added documentation and yet capture necessary de-sign, architecture, coding, integration and testing knowledge in a manner that is easily digestible and understandable by the engineering staff tasked with sustaining the software (while also meeting DoD documentation requires). Automation helps balance these competing needs.

In a DevSecOps environment that documentation should be developed and reviewed incrementally. For any documentation 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, and so forth.

What Metrics Can Be Provided Through Your DevSecOps Pipeline?

Information can be obtained from tools that automate the DevSecOps pipeline and tools surrounding the pipeline that manage the work. DevSecOps pipeline tools that perform software development functions such as build, test, and deployment offer direct access to data pertaining to interim work products and overall stability of the code. Tools in the pipeline that support development like source control, static analysis, requirements management, issue tracking, and financial systems provide additional data to track and monitor growth and quality. A simplified single pipeline is shown in Figure 2. A program typically includes multiple pipelines with integration points.

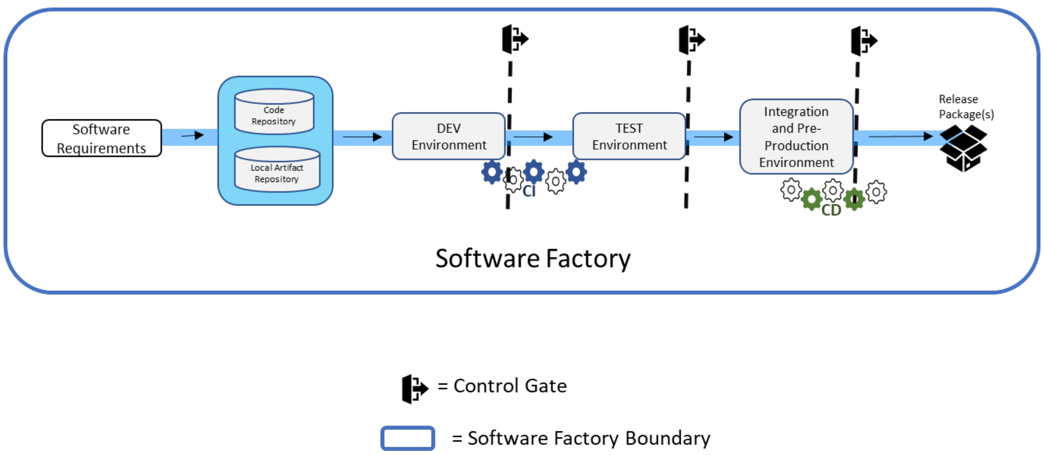

Figure 2: Simplified Software Factory Pipeline

Figure 3 provides an overview of how the main components of a pipeline-based measurement system could work. Measurements such as timestamps, duration, product size, and failure measurements are available at many points in the pipelines. Data is collected from several sources, including the work ticketing systems and the CI/CD pipeline. The requirements, financial system, and roadmap (or work breakdown) provide context for tracking the flow of software work items through the pipeline. Context allows the work to be aggregated into meaningful functional capabilities 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 of these metrics are collected automatically, others must be derived through analysis, which can often be automated.

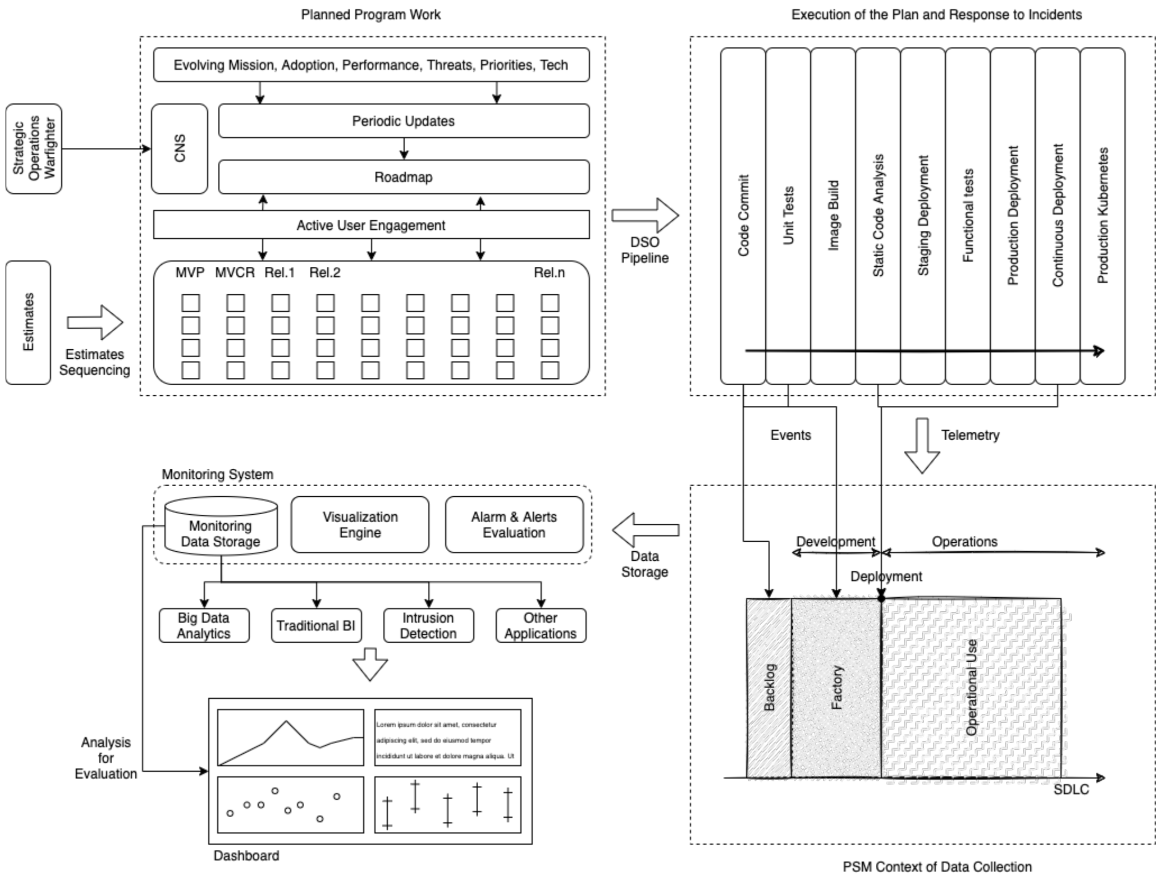

Figure 3: DevSecOps Data Collection Overview

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 ensure that data from differing projects or pipelines is properly aggregated.

Capability Delivery

In a DoD environment, the delivery of capability should occur on a schedule described by the strategic roadmap. The roadmap includes a minimum viable product (MVP), a minimum viable capability release, and subsequent releases. That is, while the program is ongoing, a capability is defined and has a definition of done. Each incremental delivery should measurably advance the objective of fielding a capability. This leads to metrics for percent complete and completion dates, but only if

1) the capability maps to the work items (stories, features, epics),

2) the work items are sequenced to support a delivery roadmap,

/API

3) the work is staffed and

4) the overall capability and work items have reasonable estimates

The PoPs environment incurs the additional challenge of aggregating deliveries from multiple pipelines, often with cross pipeline dependencies. For example, the diagram in Figure 4 describes a fictitious program including 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_{8-1} represents a lead time for rework discovered in integration to reenter the development pipeline at node 1.

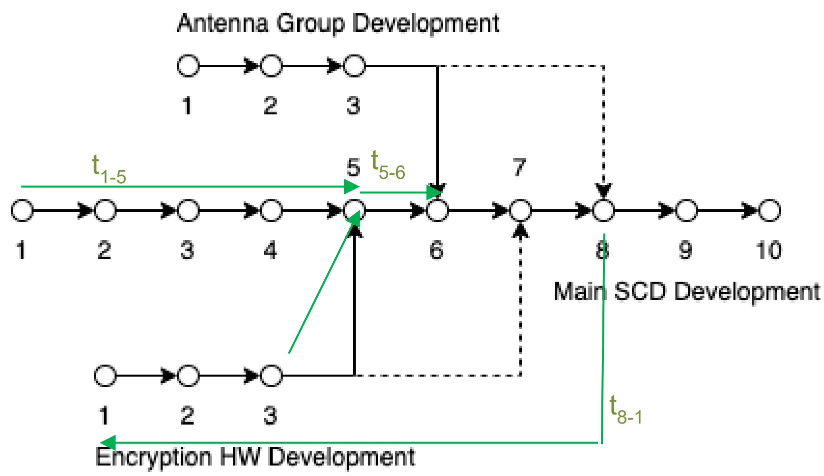

Figure 4: Multiple Interacting Pipelines for a Fictitious Cyber Physical Device

Nonetheless, the principles of tracking the percent complete and completion rates remain.

The DevSecOps pipeline can provide data to help PMs measure the progress toward capability delivery. To deliver a capability, work items that implement the capability (i.e., product requirements) must be identified and organized. The work items must be estimated and timestamped as they progress through the system. The most reliable way to measure completion is through delivery or deployment. In DoD the work items are required to be organized in a product-based work breakdown structure (WBS). The WBS is to be estimated, and tracked as the work items are completed and functionality is added to a delivery. A WBS organized around features and capabilities that are often distributed to teams as stories or epics that the teams will decompose into stories. Feature and/or capability completion at the end of an increment can provide a good measure of progress compared to the roadmap. High-level estimates must be in units that are meaningful to the PM, such as 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 WBS provided that the completed work items are identified as product features linked to the WBS. Additional measures can be used to ensure that the product has been built well, including build success rates, test success or failure.

Cost

The DevSecOps pipeline provides 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 provide EVM-like data even absent of a requirement for EVM. Cost is most evident from work applied to specific work items, which in turn requires information on staffing and the activities performed. For software developed using Agile processes in a DevSecOps environment, there are metrics available through the pipeline that can provide data on team size, actual labor hours, and worked completed versus 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 FTEs correlate very strongly with cost at the pipeline or factory level. And, at the team level, the DevSecOps cadence of planning increments and sprints (i.e. schedule This traceability does not naturally occur, nor will the metrics if not adequately planned and instantiated as well) 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.

A PM can use metrics on work completed versus planned to make informed decisions about potential cost overruns for a capability or feature. These metrics can also help a PM prioritize work and decide whether to continue work in specific areas or 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 versus 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, as they are built, and as they are 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 the roadmap estimates.

Schedule

The DevSecOps 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 WBS, features can be aggregated to show progress versus the plan for capability delivery as well. This traceability does not naturally occur, nor will the metrics if 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 small enough to detect schedule slips but large enough to avoid excessive plan churn as work is reprioritized. Be aware that 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 on a short-term scale and the plans must be updated whenever priorities change. 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 PM needs credible ranges for program schedule, cost, and performance.   Measures that inform predictability can be obtained from the pipeline bias and variability of estimates, throughput, and lead times.

It is important to remember when assessing schedule and progress for large multi-pipeline software development programs 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. 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. Work performed to complete individual stories (and/or sprints) is not guaranteed to generate working software. From the PM perspective, only completed software products that satisfy all conditions for done 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 the schedule performance of each pipeline to determine that work is progressing 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

A practical approach to address different pipeline performance parameters is to 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.[2] The specific pipeline throughput and lead time distributions must be placed into the context of that pipeline’s inventory of work and reported in units that are meaningful to the PM 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

[2] Alleman, G. B.; Henderson, M.; Hill, C. H. M.; & Seggelke, R. Making Agile Development Work in a Government Contracting Environment. Pages 114-119. In *Proceedings of the Agile Development Conference*. 2003. https://ieeexplore.ieee.org/document/1231460

pipeline time stamps. However, the information must be joined to requirements and task management systems. These considerations imply measurement needs that have not been explicitly considered in the DevSecOps literature. For illustration, consider the merge points from the example of a multipipeline system shown in Figure 4.

If you are a PM on a typical DoD program, schedule estimates for multiple pipelines can be performed manually, for example using earned schedule [Lipke 2003]. The authors of this paper have spoken to PMs 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 be assessed so easily. Fortunately, stochastics approaches such as Monte Carlo Network (MCN) 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

When dealing with multiple pipelines, the PM must aggregate the data carefully. Each team will have its 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 the progress of capabilities and features that may span one or more pipelines against the roadmap. Another important consideration for the PM is how to deal 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 understanding 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 as certification, authority to operate, training, and others can all impact the schedule. While some of these have stories that relate to them that will be tracked in the pipeline, many happen either fully or partially outside of the software CI/CD pipelines.

Technical Performance of the Capability Delivery

A necessary condition for a capability delivery is that all work items required for that capability have been deployed through the pipeline. Delivery, however, is insufficient for a capability to be considered complete. A complete capability must also satisfy the specified requirements and satisfy the needs in the intended environment   The development pipeline can provide early indicators for technical performance. is, therefore, connected to the delivered capability and schedule through the definition of done. That is, the schedule performance cannot be determined until the requirements for capability have been satisfied. Technical performance is normally validated by the customer. Nonetheless, technical performance includes indicators that can be measured through metrics available in the DevSecOps pipeline.

A indicator of technical 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. A caveat is that the PMO staff must assure that tests in fact test for the needed technical performance.

A second way to measure technical performance is to ask users for feedback after sprint demos and end-of-increment demos. Feedback from these demos can provide valuable information about the system performance’s ability to meet use needs/expectations. A third way to measure technical performance is through specialized testing in the pipeline. Stress testing that can test requirements for key performance parameters such as total number of users, response time with maximum users, and so forth. can help determine how the system will perform when deployed.

Risks

Risk results from uncertainty and includes potential threats to the product capability, operational issues such as cyber-attack, delivery schedule, and cost. The program must ensure that risks have been identified, quantified, and, as appropriate, tracked until mitigated. For the purposes of the PM, risk exposures and mitigations should be monetized. The risk mitigations should be prioritized, 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 should be included in reports to the PM.

Quality

The program must be satisfied that the development uses effective methods, issues are identified and remediated, and the delivered product has sufficient quality for not just the primary delivering pipeline but for all upstream pipelines as well. Prior to completion, individual stories must pass through a DevSecOps toolchain that includes a number of automated activities. In addition, the overall workflow includes additional tasks, design, and reviews that can be tracked and measured for the entire PoP.

Categorizing work items as “feature,” “bug,” “risk item,” “technical debt,” or “adaptation” can provide a lagging measure of program health.[3] 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 the 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 automatically run daily or with every code commit. These analyzers output weaknesses that were discovered. Timestamps from analysis execution and code commits. can be used to infer the time delay that was introduced to address the issues. Issue density, using physical size, functional size, or production effort, can provide a first-level assessment of the overall quality of the code. Large 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 may predict technical debt. For a PoPs, analyzing the upstream and downstream results across pipelines can provide insight as to how effective quality programs are on the final product.\

Automated builds provide 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 way to measure the build rework. For PoPs, build time following integration of upstream content directly measures how well the individual pipelines collaborated.

Test capabilities within the DevSecOps environment also provide insight into overall code quality. Defects found during testing versus 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. Defects provide a record changes needed for the code to operate properly. Since tests, inspections, and scans are imperfect, the number of defects found prior to release is a useful predictor of defects that escape into release. Tracking defect discovery and closure thus providing a PM with a wealth of knowledge as to the overall quality of

---

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

the software and the production process. Software that meets only minimal requirements for release may contain latent quality concerns that build up over time, creating a backlog of technical debt, which will be costly to rework later. Tracking defect insertion across pipelines, as well as, analyzing defects to identify commonly committed defects across independent pipelines can help the PM identify areas for improvement.

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. For a PoP, if the PM identifies a particular pipeline with high defect insertions, they can focus their attention 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 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 PM must be satisfied that cybersecurity practices are being performed, that issues are found, and that the discovered issues are addressed. Cyber issues include library changes, bugs associated with weaknesses, and cybersecurity-related work items. These should be counted, and effort and lead times should be measured. The risk lead times (age), burndown, and planned and actual mitigation effort should be tracked and reported because escaped cybersecurity weaknesses represent a risk. The DevSecOps pipeline can collect metrics related to cybersecurity such as effort applied to cybersecurity activities, weaknesses found during static or dynamic analysis, cybersecurity test failures, information on cybersecurity-related issues, and cyber security weaknesses that escape a pipeline. This data can be used to inform cybersecurity risk, assess if 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/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. If they are, many times the data is provided as an overall headcount or FTE number. The PM of a PoP rarely has quantitative insight as to critical skills positions and their retention rates.

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. If a PM believes there are staffing issues, pipeline data could provide metrics to show whether this might be an issue.

An emerging trend within PMOs, is to have all Program partners (i.e., subcontractors, prime contractor, and government technical staff) identify members of their staff that have critical skills, capabilities, and experience regardless of their tasking. This provides the PM the ability to manage personnel and recruit across the Program at large. Comparing the availability of not only total staff, but also specialized skills allows the program to determine the validity assumptions upon which schedules depend and take appropriate action.

Putting It All Together

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 can 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 program and during the lifetime of a program.

For example, early in a program, staffing may be a risk and the PM will want to assess based on the plan so that decisions can be made based on available staff. 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. With instrumented DevSecOps pipelines, those metrics are more readily available, and many can be provided 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.

Definitions

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 [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 prioritizing, planning, estimation, and reporting [https://resources.sei.cmu.edu/asset_files/TechnicalNote/2013_004_001_62918.pdf]

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. [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

[Kersten 2018]

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

[Lipke 2003]

Lipke, W. H. Schedule Is Different. *The Measurable News.* Volume 2. 2003. Pages 31–34.

Legal Markings

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.

DM22-0325

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