Carnegie Mellon University
Software Engineering Institute

F-35 JPO Software Metrics White Paper

Will Hayes, Pat Place, Julie Cohen, Nanette Brown, Keith Korzec and Chris Miller

December 2020

1 Introduction

This white paper describes the role of metrics in an iterative and incremental delivery of a complex cyber-physical system with software-dominant capabilities. Metrics offer *insight* into the program's realization of a functioning system as well as the performance of the processes in place. This enables *oversight* on the performance of the program as it delivers contracted capabilities. Knowledge of program context, and a shared understanding of the actions and outcomes driving the metrics, enable the insight required. Three specific areas key to incremental delivery are emphasized here: Assessing Progress to Plan, Assessing Quality, and Flow Based Metrics. This third area reflects the influence of Lean and Agile thinking and informs the first two areas.

Assessing Progress to Plan for iterative/incremental delivery typically involves assessments against near-, mid- and long-term plans – rather than a single integrated master plan. The heavy emphasis on customer/user collaboration supports more detailed insight about engineering progress, but the scope of the assessment tends to be narrow (and deep) with limited ability to forecast broader outcomes. Subject Matter Experts use structured feedback and/or score cards devised for engineering reviews and demonstrations against plans that govern the near-term time horizon. The availability of new product knowledge and process performance data with each iteration/increment enables confirmation of progress against the roadmaps for feature delivery and retirement of technical debt. The new insight is used by teams to prioritize and measure mid-term work. Finally, the contribution of the incremental deliveries toward the achievement of long-term plans for delivery of contracted capability is typically measured against criteria that derive from program or acquisition goals (e.g., major milestones like IOC/FOC). Each of these levels of oversight have an aperture and focus unique to their associated plan and the time-horizon. This succession of plans takes the place of a single integrated master plan. The orderly revision of a program's IMP is also supported by these three complementary perspectives.

Assessing Quality through examination of the product is a necessary, but insufficient, focus for conducting oversight; the development process must also be examined. The identification of defects in the product, and the *escape* of defects through activities meant to find them remains a well-established focus for metrics. More recently, the pace of delivery for software-dominant capabilities has greatly increased through the beneficial use of automation and tooling, which brings new ways to rapidly assess the quality of the product. In contrast to an exclusive focus on the delivered work products, metrics on the process itself offer insight about the performance of the enterprise and potential leading indicators for

future outcomes. Metrics that quantify code coverage and other key attributes of the testing process serve as measures of quality for the testing activities. Finally, the range of deliverables that result from iterative/incremental development offer thresholds for *containment* of defects within iterations, increments, releases, or major deliveries – each with their own threshold for triggering corrective action.

Flow-Based Metrics reflect a new focus on balancing priorities for performance, with an emphasis on timeliness and quality to enhance predictability. Analysis of lead times for things such as feedback from test activities help to characterize the efficiency of the process, while frequent integration and automation-driven testing bring early verification of quality. Analysis of cycle times helps us relate the number of work items in progress with the time it takes to complete the work and the rate at which results are delivered. Analysis techniques suitable for use with these metrics include cumulative flow diagrams, cycle time scatterplots and histograms.

Iterative development of software capabilities promotes learning and evolution through rapid implementation in small batches. Integrating each new batch into the system, using automation for builds and verification, the assessment of quality occurs early and grows with the product. With a user-centric focus, the emphasis is on building capability in a demonstrable manner - to increase feedback about the likely operational performance of the system. Assessment of progress, quality and the flow of results in the context of multiple planning horizons adds new perspectives for oversight, and this paper is an introduction to these perspectives.

2 Assessing Progress to Plan

The confidence of stakeholders is managed through timely communication of information that demonstrates accomplishment of near-term goals. The link between these accomplishments and the long-term plan for the program is an essential focus of program management. The technical adequacy of the detailed work is of primary concern as well, and serves as a necessary pre-condition to marking progress to plan. Technical adequacy is sometimes treated as an adjustable parameter in program performance, and one that is often sacrificed to enhance cost/schedule performance. In the extreme, this can quickly become a death spiral, as the cost of poor quality undermines the predictability of performance as well as the effectiveness of corrective actions.

Iterative development methods establish clear criteria for the acceptability of work at different levels of detail which serve as guards for the integrity of the progress claimed against plans. For major programs in the DoD, this can be seen in the hierarchy of plans established for different time horizons. Three levels of this hierarchy are described in the sections below, starting from the bottom and moving upwards.

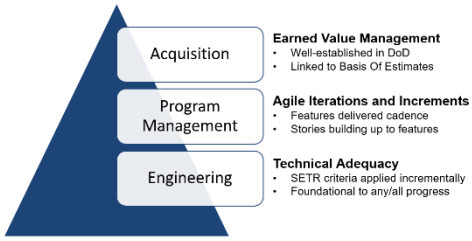

Figure 1: Three Perspectives

Section 2.1 is focused on Engineering: describing metrics based on meeting technical and engineering targets and performance goals. The technical adequacy of the detailed work is measured to ensure that timely quality work leads to capabilities maturation. These metrics are targeted to an audience of team leads, subject matter experts, and engineering managers.

Section 2.2 is focused on Program Management: for programs aspiring to a continuous delivery approach, these metrics focus on tracking product elements delivered by a pipeline. Measuring the progress of stories, features, and capabilities provides insight as to how well the program is progressing. These metrics are targeted to an audience of middle and upper management who oversee the delivery of 'finished goods' over time, as they are integrated into the system.

Section 2.3 is focused on Acquisition: these metrics provide insight about the goals of the Acquisition activity. Such measures are well established in the DoD and are directly linked to program estimates and contracts (e.g., required use of Earned Value Management). These metrics are typically targeted to an audience of executive leadership and upper management. They also focus on longer time horizons, not immediate project management.

The progress of work must be assessed from these three complementary perspectives: the engineering perspective, the program management perspective and the acquisition perspective. The engineering perspective is one of technical progress. It is a major focus in programs that advance state-of-the-art technology. The program management perspective assesses progress against the planned deliverables and timelines established for the program. Finally, the acquisition perspective comes from the goal of the enterprise for which the program was established. Different skills are required to assess progress in these three complementary areas.

2.1 Engineering Progress Metrics

Assessing the maturation of complex designs demonstrating the achievement of novel technical milestones requires an engineering perspective. Leading indicators used by engineers shepherding the maturation of technology derive from the technologies used and the nature of the problem. As such, generic indicators such as productivity and quality only partially determine progress as they typically don't account for the specific technologies in use. Engineering review of artifacts deriving from requirements analysis, design activities, and examples of implementation are necessary inputs. The completion status and timeliness of these reviews are a form of measurement used on every program.

2.1.1 Plan of Action and Milestones (POA&M)

The POA&M serves as an organizing structure and provides oversight personnel with a roadmap for the technical work to be accomplished on a given capability. Accomplishment of that work is seen through the delivery of artifacts and the conduct of review events. Early in the life of a capability, the POA&M will focus on artifacts of the Requirements Work Package (RWP) process; then the focus will shift to greater and greater detail found in implementation-level artifacts such as designs, features, and code modules. As the product (i.e., software) matures, the engineers will focus on the production of the Operational Flight Program based upon detailed technical output derived from Software Integration Labs (Simulation-based or HITL) as well as developmental/operational (flight) system test. The POA&M is the vehicle for defining the focus and decision criteria for marking progress.

2.1.2 Technical Reviews

The major deliverables listed in the POA&M for each capability will undergo technical reviews through an appropriate instantiation of the Systems Engineering Technical Review (SETR) process. Often these are conducted in an iterative fashion – to support rapid progression of the work in smaller work packages. Criteria for attaining the level of maturity implied by traditional milestones (e.g., PDR and CDR) are considered as the technical work underway produces artifacts recording the accomplishments[1] – which then serve as input to the next engineering activity. Caution must be exercised when using technical reviews as a measure to ensure true technical progress is being made and not merely the production of an artifact. Agile development would recommend use of early testing of the designs to provide a degree of rigor and objectivity; such testing can be achieved through a greater emphasis on digital engineering. Scorecards used in technical reviews may not always contain quantitative measures of performance, but the objectivity of this process is often superior to a vague numeric index based on the passage of time or consumption of resources. Action items and requests for information are tracked following the reviews, and satisfaction of these as well as timeliness are measured.

2.1.3 Demonstration Events

The technical pulse of capability maturation is made visible through demonstrations of the features and capabilities as they mature. Whether demonstrating a small piece of functionality in a test harness or

---

[1] Discussions with the capability leads indicate that the leads use informal contacts and anecdotal data in addition to other data to form their picture of progress.

running a full simulated mission in a robust lab, the developer provides deeper insight into accomplishments through demonstrations than would be obtained from a written report of their success. Objectives for specific functionality or performance are evaluated by appropriate subject matter experts in demonstrations. Some organizations collect scores for utility (or value) of the demonstrated functionality.

2.1.4 Verification Events

Judgements recorded by engineers in the Verification Results Report (VRR), or concurrence on the proposed closure of defects, are used to summarize engineering judgment. Whether represented as test points completed, requirements sold off, or deficiency reports closed, these measures confirm attainment of technical requirements and are essential for confirming progress.

2.2 Program Management Progress Metrics

The program management perspective, in contrast to the engineering perspective above, is informed by a roadmap of milestones, review events, and deliverables that are designed to provide a steady source of information to confirm progress or trigger action. Completion status for units of work (e.g., stories, features, use cases, capabilities, etc.) and their satisfaction of relevant completion criteria, along with the associated schedule and cost data, are the most common sources of these progress indicators. The Definition of Done provides the mechanism to define all the criteria that must be satisfied before the item being developed can be considered complete. However, this requires a robust Definition of Done, agreed by both government and contractor, to ensure that work is, in fact, complete.

It should be noted that when a developer is using all the capabilities of an application lifecycle management tool (such as Jira or VersionOne) the data needed to supply most metrics in this section is available in the tool and, indeed, the metrics can be generated automatically, though there may well be need of additional tools to provide summary-level data given the size of the program.

2.2.1 Completion of Features within a Program Increment

Measuring the number of features that met the Definition of Done during a program increment (PI) provides a tangible indicator of actual progress accomplished and is typically displayed via a burndown chart. When combined with other data, this provides additional insights, such as those below:

- Comparing the number of features completed to the number committed to during PI Planning provides insight into the estimation process.
- For a given capability,[2] comparing the number of completed features to the expected number of features provides a rough estimate of the completion of the capability (not all features are of equal difficulty and not all features required for the capability may be evident at the start).

The satisfaction of acceptance criteria, as well as the assessment of Definition of Done typically indicates a meaningful delivery of functionality greater than ‘the unit level’ (implying a level of integration).

---

[2] This same approach can be used for other containers of work such as features, epics, use cases, and even the POA&M.

2.2.2 Completion of Stories within a Sprint

The completion of stories for a given feature provides a more granular look at progress, but often, by the time the data is available to a government audience, it is too late to take meaningful action. While such data is typically of more use to the teams performing the work, the government can gain insight by looking at the consistency of the data. If there are wild fluctuations, then it is likely that there is instability in the development process and that progress will be impeded.

2.2.3 Work in Progress Limit Violations

Work in Progress (WIP) is a term used to define the number of items in any given state. Although it may seem counter-intuitive, flow of work through the process is improved when there are limits to the number of items in any given state. Following an agile process, new work cannot be accepted into a given step of the process until the number of items is below the WIP limit for that step. Tracking when WIP limits are violated helps identify external constraints that impact progress. A quick analysis of the root cause can often reveal opportunities to increase the amount of progress going forward. Indeed, if there is a significant number of such violations it is likely that the limits are simply not being observed.[3]

2.2.4 Traceability Matrix Report

A traceability matrix report allows the program office to ensure that only work directly linked to a capability is being performed. Note that the work itself should include enabler features such as upgrade to the development or test environments (e.g., the addition of a new test harness). This also provides a data integrity check to support the measures identified above, as the tally of stories and features completed, as well as the completion status of the contracted capabilities may be inferred from this report.

2.2.5 Defects Reported / Escaped

Defects identified after "completion" often indicate either a weakness in the Definition of Done or lack of adherence to that Definition of Done. A larger number of defects that have escaped erodes confidence in the completion data and accompanying metrics and, ultimately, slows progress as attention must be paid to correcting the defects. The section on Assessing Quality offers more on defect reporting.

2.3 Acquisition Progress Metrics (Earned Value Applicability)

An Earned Value Management System (EVMS) is a program management tool for tracking costs, schedule, and scope against an initial plan. The structure of the data generally aligns with the Work Breakdown Structure (WBS) and master schedule.

Historically, the EV structure was focused on Computer Software Configuration Items (CSCI). This method, however, does not readily support an agile development effort. For agile developments, as recommended by OUSD [OUSD AT&L 2018], using the product backlog to populate the EVMS provides tracking with a technical perspective.

---

[3] Much like speed limits which tend to be observed much more accurately in the presence of the police.

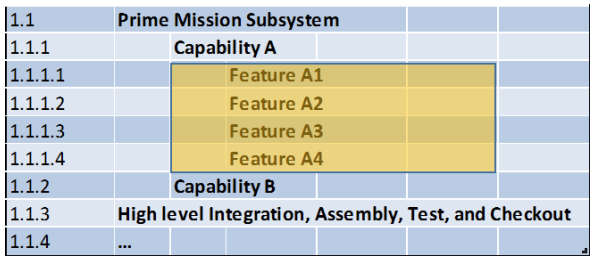

Figure 2: Possible Agile Breakdown, Not Prescriptive [OUSD AT&L 2018]

In order to facilitate delivering working software, it is critical to assign sizing constraints to each level of the requirements breakdown, from the POA&M down to the features (and possibly, also, the stories). Once these constraints are assigned, however, structuring the WBS to encourage flow and progress becomes much easier. Sizing constraints are important for two reasons: first to normalize the work packages, ensuring that they fall within a fixed size range so that they are comparable and second so that the level of detail doesn't make the WBS unmanageable.

One approach that shows promise is to use the "just in time planning" aspect of agile development in conjunction with the EVMS rolling-wave strategy to incorporate the most current program performance and product knowledge. Features committed to during the PI Planning are updated in the EVMS to reflect the latest cost/size information immediately after the planning event.

When features are small enough to be completed within a Program Increment (typically a maximum of 45% of the increment's duration), progress may be easily documented via the 0/100 method once the features satisfy both the acceptance criteria and the Definition of Done.

An additional benefit of structuring the EVMS by feature is that it allows the contractor's internal tracking tools to supply the data necessary to update the technical progress and costs.

When the EVMS reflects the features level of the product backlog, the agile metrics identified throughout this document provide evidence that supports the standard EVMS metrics.

Finally, using EVM provides a rigor to the capture and use of typical agile metrics that is beneficial for large-scale programs.

3 Assessing Quality[4]

Without quality, faster progress is meaningless. Indeed, as quality goes down, there can only be the appearance (through careful misuse of progress measures) of faster progress. Since quality is largely

[4] Although this paper makes a separation between assessing progress and assessing quality, the two concepts cannot be divorced that easily. For example, the qualities of extensibility and maintainability directly affect cycle time with respect to making changes.

subjective, at best only indicators of quality can be measured. Two types of quality indicators are discussed in this section.

Section 3.1 is focused on Artifact quality: describing metrics primarily based on attributes of artifacts that make up deliverable products, or artifacts used to capture and communicate representations of the end product (e.g., designs, user instructions)

Section 3.2 is focused on Process quality: describing metrics primarily based on attributes of the process used to develop or deliver those artifacts

The division isn't wholly binary, as some indicators, for example, those related to escape and defect counts, point to both artifact and process quality. The metrics are grouped according to their stronger indication; for example, the number of unplanned drops more strongly indicates problems in the development process in contrast to the number of escapes that more strongly indicates problems with the artifacts.

The focus of this section is on how to assess quality rather than how quality is defined; there are many articles and books (e.g., Fenton) that can aid in understanding the different dimensions of quality and choose the dimensions most appropriate at any given time [Fenton 2015]. If we accept the assertion that quality is subjective, then any assessment that uses objective measures can, at best, provide indicators of quality.

Examining quality indicators in context is necessary for drawing the correct conclusions with respect to the quality of the artifacts being produced. For example, an error occurs when actual outputs don't fall within the boundaries of the expected outputs for a given test. Obviously, the software may have a defect, but it is also possible that the test was incorrectly administered or that the test places incorrect expectations on the results. Careful analysis of all the indicators is needed to identify the root cause of the potential issue.

3.1 Assessing Artifact Quality

The terms *artifact* or *item* have been used throughout the document to describe the *things* produced by the development process. For the most part the term, as applied, denotes software, or hardware, or subsystems or whole systems. However, the more general term has been used in order to accommodate other important artifacts such as engineering drawings or other documents. The following discussion, motivated primarily by testing, is naturally biased toward more technical artifacts such as software and systems.

3.1.1 Counting Escapes

An escape is typically defined as a defect in an artifact that is not detected until the artifact has crossed some boundary.[5] For example, a bug introduced in the development of a user story would be considered a defect if detected in unit testing, but would be considered an escape if not detected until, say, the user story is integrated with other user stories to form a feature.[6]

If an issue in some artifact is determined to be an escape, then there is a need to determine if, and how, that escape could have been detected before the artifact crossed the boundary that turned the defect into an escape. A corollary to this analysis is to determine whether it is cost effective to detect the issue prior to the boundary. A count of the changes to the development process to prevent such escapes can be used as an indicator of organizational maturity.

The escape count, by itself, isn't very helpful; while higher numbers indicate lower quality than lower numbers, what's more important is how the escape counts trend over time. Further, escapes need to be considered in light of test coverage including code coverage, functional coverage, and interface coverage. These considerations are necessary since it is easy to obtain a low number of escapes by performing very little testing at intermediate steps; though these escapes will become evident in the operational system.

All escapes are bad, but some are worse than others, so escapes are categorized in various ways. Escape metrics should be reported according to the categorization scheme in place.

Note that, for many systems, it is typical that test results are available immediately thus preventing defects from becoming escapes. However, if test results are delayed because of the need for lengthy manual analysis or the late addition of more sensitive analysis tools, the artifact may have already "crossed the boundary" before the test results are available. In such cases, the error should be considered a defect and not an escape.

3.1.2 Percentage of Passing Tests

One of the most versatile indicators, and yet hardest to categorize, is a measure of the percentage of tests that pass in any given environment. The versatility stems from the fact that it can be used as a measure of engineering progress, of the development process quality, and of the quality of the artifacts. As its name suggests, this is the percentage of passing tests in comparison to the total number of tests.

When the percentage increases, engineers have confidence that the artifact is satisfying more of the tests and, by extension, more of the requirements, and can be seen as a measure of progress. Similarly, when there is a low initial percentage rising to a high percentage, there is some confidence that artifact is of increasing quality as it is passing its tests. Each change prompts the question "what happened?" for

---

[5] This distinction between defect and escape is sometimes also defined as private and public defects.

[6] In a development practicing continuous integration; the error may well be detected before it is considered to be "done" and therefore would be classed as a defect and not an escape.

which the answer will provide insight into quality. Finally, a saw-tooth graph (**Error! Reference source not found.** 3), where there are dramatic drops in the percentages followed by a steady rise is frequently an indicator that tests are created and then code is developed to satisfy the tests. However, this final picture is unlikely to be seen unless highly detailed metrics are captured.

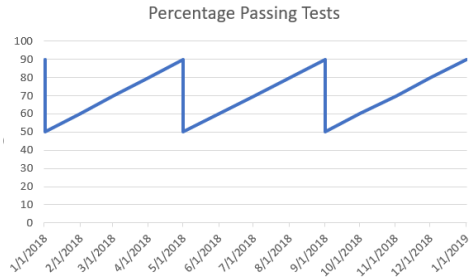

Figure 3: Saw-tooth graph of passing tests

Trends in this metric must be compared to other metrics, such as the numbers of tests, to gain appropriate insight into which type of quality is being observed. Tests should be developed at the same cadence as the artifacts, ideally just ahead of the artifacts. Develop too far ahead and tests may be created for functionality that is never developed; develop tests too far behind and the artifacts will be of unknown quality for an extended time.

3.2 Assessing Development Process Quality

Unfortunately, analysis and testing are typically performed after (or at periods during) development and are, therefore, lagging indicators of quality. Further, leading indicators of quality are dependent on the specifics of the development process and depend on some level of uniformity across the organization. Of course, the lack of such uniformity is also an indicator of poor process quality.

Note that an organization with a high-quality development process will find opportunities to increase the frequency at which quality assessments occur and, at the same time, extend the reach of the quality assessments. In modern software development organizations, developers complete some section of code and check it into the software repository indicating that they consider the task complete. This check-in triggers analysis and unit tests that provide immediate feedback to the developer. Assuming that the code passes the unit tests, the unit is integrated with other units to create an assembly that is similarly tested and feedback sent back to the developer. This process continues, creating ever larger units of functionality that are tested to the greatest extent possible, until all automated testing has been accomplished. Even when automated testing is complete, tests can continue – albeit at a slower pace.[7]

3.2.1 Number of Unplanned Drops

One of the most obvious indicators that quality is deteriorating is the number of unplanned drops. Consider the existing process where a capability is planned to be released in a specific drop, but when the drop is made the capability has critical issues that require a second drop (or more than two drops) to be

[7] This discussion of quality assessment leads to the conclusion that greater investment in the test environment will lead to higher quality products. However, investing in test environments should be guided by expectations of the benefits that will accrue from the early detection of the types of defects any given environment can detect. For example, the cost of increasing the fidelity of the hardware (or hardware simulation) in a given test environment should be compared to the cost savings of the defect detection that the higher-quality hardware can perform.

made in order to deliver a usable version of the capability. The number of unplanned drops is a simple indicator that can be tracked with respect to planned drops.

While easy to track, this is a very broad (and lagging) indicator of quality. Root cause analysis is needed to determine the basis for the lack of quality. Likely causes include planning more work than can be accommodated during development; addition of more work (e.g., new requirements or urgent defect fixes) during the development period; and insufficient testing prior to the final integration.

3.2.2 Code Coverage

This is the percentage of the total code that has been executed in response to the tests. Low percentages indicate either the code contains unreachable branches "dead code" or that the existing test suite isn't exercising all of the code. Code coverage percentages are typically accompanied by a report on the source lines of code that were and were not executed. Only by analyzing such reports is it possible to determine whether more tests or less code are needed.

Code coverage works best for unit testing, but also can be employed on integrations that can be suitably instrumented. Since the tools needed to determine code coverage slow down overall execution, this metric is likely to be employed only in test labs and for tests where speed of execution is not being determined. It is rare to achieve 100% coverage (as required by standards such as DO-178C) with tests, however, numbers greater than 70-80% should be expected from well written code and tests. With object-oriented programming and a great deal of inheritance, the percentages of code coverage are typically lower.

3.2.3 Test Metrics

There are a number of metrics all relating to testing that are discussed here as a group rather than separately; these are focused on the number and quality of tests in the test suite. None of these measures, by themselves, provides much useful information, although the aggregation of these measures provides an overall picture of the testing process. These should be reported periodically and team leaders should be looking at the trends from iteration to iteration, as they provide an indication of a focus on testing throughout development, whereas more senior leadership will likely watch the trends across program increments.

1. Number of new test cases created
2. Number of existing tests that are modified
3. Number of new test cases that are automated
4. Total number of tests
5. Percentage of tests automated
6. Number of tests per feature

Obviously, these numbers would need to be reported with respect to the different testing environments. Further, given the differences between the testing environments, the number of replicated tests (e.g.,

those in the simulated environments that are repeated in the hardware-based environments) would provide assurance that testing appropriately accounts for the higher fidelity environments

The discussion assumes that once tests are created and automated that they will be consistently used for regression testing. This implies, of course, the need to automate testing given the increasing number of tests. For these measures to be meaningful, each test should exercise different parts of the system. While not a perfect guide, the addition of a test should be visible in terms of the lines of code executed.

3.2.4 Total Containment Effectiveness (TCE)

TCE is a calculation of the effectiveness of the processes developing the code in terms of catching defects where they are introduced, rather than discovering them in later stages of development where they are called escapes. As such, TCE is an indicator of quality, since the higher the value of TCE, the fewer defects escaped the development. TCE is defined across the entire development lifespan as:

$$TCE = \frac{escapes}{escapes + defects}$$

Equation 1: Containment Effectiveness

Although TCE is usually considered across the whole lifespan, there are opportunities to use the concept in smaller units. For example, we can consider TCE for each internal engineering drop (where these are used) as well as for each production drop. Trends in TCE across each of these series of drops will provide indicators of overall code quality.

There is a natural progression for testing from simulated environments to hardware-based environments to flight testing; this progression provides opportunities for calculating TCE based on each test environment. Because the test tools and approaches differ across the test environments, it might be impossible to find some defects in some environments and these should not, therefore, be counted as defects unless, of course, they are not detected in the environment where they could be detected.

3.2.5 Definition of Done Containment

A key concept found in most agile development processes is Definition of Done; this is an agreed-upon set of items that must be completed before any given work item can be considered to be complete. It is important to understand that the Definition of Done varies by the type of work item being considered. At the team level, Definition of Done typically includes that stories satisfy their acceptance criteria, acceptance tests are passed, any necessary new automated tests are created, code follows defined standards, automated code analysis shows no serious issues, and any associated models or documents are updated. However, when stories are combined to create features, a new Definition of Done is needed for the features. It's not enough that every story has satisfied its acceptance criteria; the integration of those stories must pass acceptance criteria defined for the feature. There will be other elements of the Definition of Done for features including, for example, that the feature is demonstrated at the System Demo and the absence of any must-fix defects. This idea of adjusting Definition of Done extends to larger and larger assemblies.

It's clear that satisfaction of Definition of Done defines a boundary from one type of work to another and thus provides an opportunity to capture a defect containment metric. As discussed above, the metric depends on the number of defects caught prior to Definition of Done being satisfied and the number of escapes found in the same artifact after it has been signed off as done.

Definition of Done containment uses the same calculation as TCE (Equation 1) except that the calculation is performed at the boundaries defined by the different Definitions of Done. Containment metrics are open to manipulation in terms of deliberately introducing and then catching defects prior to the determination of done. Low values of Definition of Done Containment point to either poor enforcement of the Definition of Done or that the Definition of Done does not consider the needs of those that use the artifact. For example, if information assurance scans will be applied later in the process and artifacts typically fail those scans, the Definition of Done for the artifacts should be amended to include those scans as a prerequisite for the artifact to being "Done". The large number of problems likely present in a major development effort would cause a sprint-level "Definition of Done" containment metric to flood those collecting the data. Higher level, such as feature or higher-level Definition of Done Containment will be of more use. In a waterfall development, Definition of Done Containment devolves to being a Phase Containment metric.

4 Flow-Based Metrics

Flow-based metrics are rooted in Lean principles and focus on reducing time to value, accelerating learning and reducing waste. Lean metrics are typically built around cycle time, which is the total time it takes for a specified task, activity, or process to complete from start to finish. Cycle time encompasses both active "touch time" (the time during which a work item is actually being handled or modified in some way) and delay time (the time the work item spends in a wait state in between processing steps).[8] Cycle time metrics may be implemented across the entire workflow or segments of the workflow. Successful implementation of cycle time metrics requires a clear and unambiguous definition of what initiates and what terminates the process or activity, as well as the entry state and exit state of the work item being processed.

Note that much of the following section is based upon the work of Daniel Vacanti in his book *Actionable Agile Metrics for Predictability* [Vacanti 2015]. The book provides an in-depth treatment of the application of flow-based metrics, including both the underlying principles and the mechanics of their implementation.

Section 4.1 Provides examples of how cycle time metrics may be applied to generate insight and support process improvement.

[8] Note that cycle time, lead time, and flow time are all terms that are commonly in use in the Lean community with meanings that may vary by author. For the purposes of this paper, we will use "cycle time" as defined above.

Section 4.2 Summarizes the relationship between cycle time and Work in Progress (WIP) as expressed in Little's Law [Little 1961]. This relationship is important because management of WIP is a critical tool in achieving cycle time goals.

Section 4.3 Discusses visualization techniques that support the comprehension and analysis of cycle time data.

4.1 Examples of Cycle Time Metrics

This section provides examples of cycle time metrics and considerations for their use. The examples show the application of cycle time metrics to "pain points," which impact program flow and "time to value." The metrics proposed by the Defense Innovation Board[9] include a number of examples of cycle time metrics. A very small set of examples is addressed in this white paper.

4.1.1 "Verification Feedback" Cycle Time

This is the interval of time between "test complete" and "analysis complete" (i.e., when the full results of the testing activity can be consumed by those who will address the findings). There may be value in assessing data from the "quick look" report that comes out of flight test, but when the program defaults to using only those results to drive quality, the analysis effort does not have the ROI envisioned.

Consideration of this metric is most applicable during flight test, but lab testing and integration testing in so-called 'lower venues' should also be considered. The time lag should be measured in hours or days. Test results delayed for weeks or months quickly lose relevance, and engineers are forced to proceed without them. The lessons from use of this metric might drive innovations in the testing and reporting process (e.g. prioritization of targeted results, introduction of automation to serve identified needs).

4.1.2 "Test Progression" Cycle Time

This is the interval of time between "code commit" and each successive layer of integration and test that lead up to the completion of the capability maturation. This metric represents a quantification in the delay of information regarding the technical viability and quality of the system being developed. In general, the shorter these time periods, the better it is for the prospects of program success. However, there will always be a practical lower limit to these time intervals.

Due care must be taken to avoid a narrow focus on an individual observation rather than considering the performance of the overall system – accelerating one capability at the expense of all others is a typical pattern of sub-optimization with hidden system-level costs. An accounting of the number of defects found in each level of integration and test is an essential backdrop for reasonable use of this metric–as trading quality for speed is counterproductive.

---

[9] The publication can be found at https://media.defense.gov/2019/May/02/2002127284/-1/-1/0/DEFENSEINNOVATIONBOARDMETRICSFORSOFTWAREDEVELOPMENT.PDF

4.2 Cycle Time Management and Little's Law

Little's Law states that the average cycle time of a process is a function of the average work in progress (WIP), divided by the average throughput of the process [Little 1961]. A proof of this relationship was established by Dr. John Little in 1961 as an outgrowth of his work on queuing theory. While Little's Law is based upon a formal proof, it aligns with our intuitive understanding that the more things we try to do at once, the longer they will all take. Understanding the impact of WIP on flow and the judicious implementation of WIP limits are key tools in reducing cycle time; such understanding can only be gained through careful use of appropriate metrics.

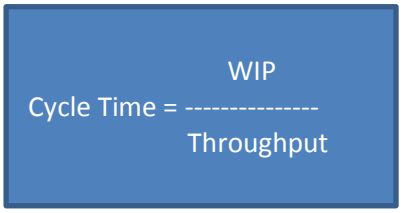

Figure 4: Representation of Little's Law

4.3 Visualizing and Understanding Flow

The use of visual displays of data to understand status and interpret process dynamics is an important aspect of using program metrics. The following sections discuss visualization techniques that support the comprehension and analysis of cycle time data. However, these types of displays may, and should, be used for displaying metrics described above.

4.3.1 Cumulative Flow Diagram

The Cumulative Flow Diagram (CFD) provides an intuitively useful visual depiction of process performance. This simple chart (Figure) reveals a "time signature" for the progression of work from the beginning to the end stages of the process as work happens. CFDs provide direct support for visualizing the three elements of Little's Law.

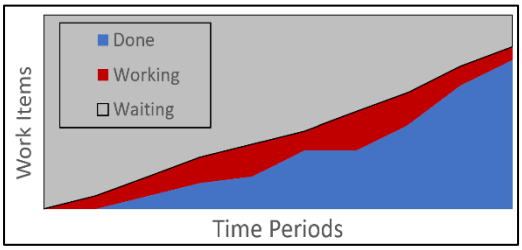

Figure 5: Sample Cumulative Flow Diagram

The red band in the graph above depicts ***work in progress*** over time. The height of the red band at any given horizontal slice in time will reflect the number of things in progress at that time. The average width of the red band across time will reflect the average ***cycle time*** over that time. Finally, the average slope of the line separating the red band from the blue zone in the bottom right is the rate of work completing, or ***throughput*** of the process.

4.3.2 Cycle Time Scatterplots and Histograms

Scatterplots are a basic visual display of data where more than one dimension of interest is quantified. A novel application of this graph proposed by Daniel Vacanti is the Cycle-Time Scatterplot [Vacanti 2015]. In the graph shown here (**Error! Reference source not found.**), each point represents a completed work product. The vertical dimension represents the cycle time for completing that work

product, and the horizontal axis represents the date and time when the work completed. With this graph, we can look for trends in cycle time across the duration of time in the chart.

The sample project data shown by the blue dots depicts a pattern of steadily increasing cycle times. This might be the result of dependencies between the work items, and items started early are paused as work on another item begins. In contrast, the orange dots show a consistent cycle time over the time period charted, which supports the analysis of data in a manner that does not presume the same level of dependencies between work items as we see in the blue project.

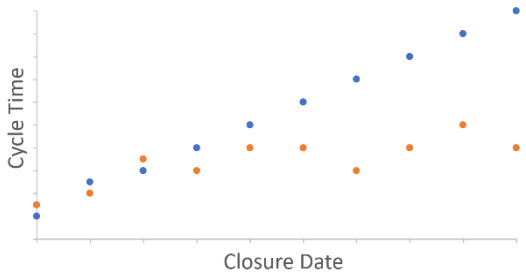

Figure 6: Sample Cycle Time Scatterplot

Histograms are another basic visual display of data commonly used for metrics. Building on the insight from cycle time scatterplots, Vacanti illustrates the use of histograms to establish benchmarks and thresholds for diagnostic purposes.

In the graph shown here (Figure 7), the horizontal axis represents the range of cycle times seen in the data. The height of each bar along the axis depicts the number of items completing in that amount of time. The pattern of these bars illustrates the fact that the majority of work items complete in relatively short cycle times, while some small number of items require longer times to complete.

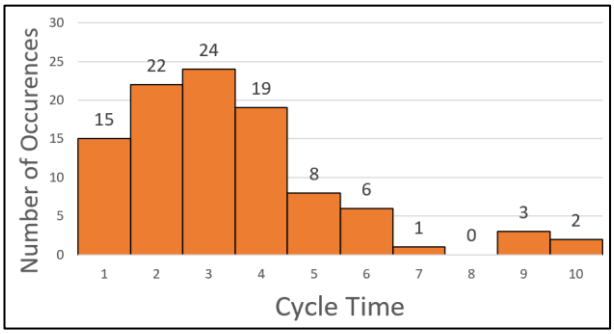

Figure 7: Sample Cycle Time Histogram

Looking more closely at the data, we can establish that only "X percent" of the cycle times seen thus far have been greater than "Y hours." Such a demarcation of a threshold, which is relatively rarely exceeded, may be used for both forecasting and diagnostic purposes. Consider the following fictitious examples:

1. If less than 25% of the review meetings finish in less than 4 hours, then allocating 4 hours or less to a series of planned review meetings would be unwise – without a mitigation tactic.

2. If 80% of work items complete in less than 1.5 days, starting work on an item with 2 days left in the iteration is a reasonable ambition.

5 Closing

This white paper provides a discussion of the major considerations for measuring progress, quality and flow, in contrast to the typical focus on utilization of resources, in the context of iterative/incremental development of major software-intensive systems.

The focus on technical adequacy embodied in the iterative approach offers targets for measurement and leading indicators of a very different nature. While using measured time and cost as proxies for value measurement is appealing, the objectivity of the clock or the dollar amount, even expressed in minutes or pennies, does not yield more information about the value of the thing it represents. This implied precision is an illusion. A program that remains precisely on schedule and within budget while building a useless product that does not address the needs of the user, is a failed program.

Ideally, the metrics available to the government will be those used by the contractors to manage the work; when the contractor isn't collecting metrics such as those described above, consider incentivizing the development of a robust metrics program. With a robust metrics program, data will be gathered as work progresses and analysis will be based on baselines and benchmarks established by the development organizations.

Those charged with oversight need to make a conscious decision with respect to the metrics, focused on completion of work, that will best serve the program. They need to ensure that a balance between progress (from the engineering, program management, and acquisition perspectives) and quality metrics is maintained[10]. Such metrics should be instituted across the program, supported by data made available by the contractor. Much of the data should already be available, the task at hand is collecting that data in a uniform manner and then using the information derived from it to make decisions with respect to the program. Deploying automation for the collection and analysis of data may also result in more time for acting on the insight provided by metrics.

[10] While measures of effort and money expended over time are useful, they are not indicators of delivery of value.

6 References/Bibliography

URLs are valid as of the publication date of this document.

[Fenton 2015]

Fenton, Norman & Bieman, James. *Software Metrics: A Rigorous and Practical Approach, Third Edition*. Chapman & Hall. 2015.

[Jones 2020]

Jones, Cheryl, et al. *Practical Software and Systems Measurement Continuous Iterative Development Measurement Framework.* Version 1.0. 2020. http://www.psmsc.com/CIDMeasurement.asp

[Kruger 2004]

Kruger, J., Wirtz, D., Boven, L., & Altermatt, T. "The effort heuristic." Journal of Experimental Social Psychology 40: 91-98. 204

[Little 1961]

Little, John. D. C. (1961). "A Proof for the Queuing Formula: L = λW." *Operations Research.* 9 (3): 383–387. INFORMS. June 1961.doi:10.1287/opre.9.3.383.

[OUSD AT&L 2018]

Office of the Under Secretary of Defense (Acquisition, Technology and Logistics). *Agile and Earned Value Management: A Program Manager's Desk Guide*. OUSD AT&L (PARCA). 2018. https://www.acq.osd.mil/evm/assets/docs/PARCA_Agile_and_EVM_PM_Desk_Guide.pdf

[Vacanti 2015]

Vacanti, Daniel S. *Actionable Agile Metrics for Predictability*. Leanpub. 2015. https://leanpub.com/actionableagilemetrics

The following are recommended as additional resources for better understanding of Vacanti's work.

[Vacanti 2015]

Vacanti, Daniel. Actionable Agile Metrics for Predictability. Tutorial Presentation: (39 min). Lean Kanban Central Europe (LKCE15), Munich, Germany. November 16-17, 2015. https://vimeo.com/146545310

[Vacanti 2020]

Vacanti, Daniel. Creating a Cycle Time Histogram in Excel. [blog post]. *Modern Kanban.* Steve Schmitz, blog publisher. April 2020. https://modernkanban.com/how-to-create-a-cycle-time-histogram-in-excel/

[Vacanti 2018]

Vacanti, Daniel. Actionable Agile Metrics with Daniel Vacanti. [audio-only podcast]. (47minutes). Agile Uprising Podcast. May 27, 2018. **https://www.youtube.com/watch?v=fXJvGnKaEKI**

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

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.

DM21-0069